JAVASCRIPT | ARDUINO | SELENIUM | DEEP LEARNING | C++

programmez.com

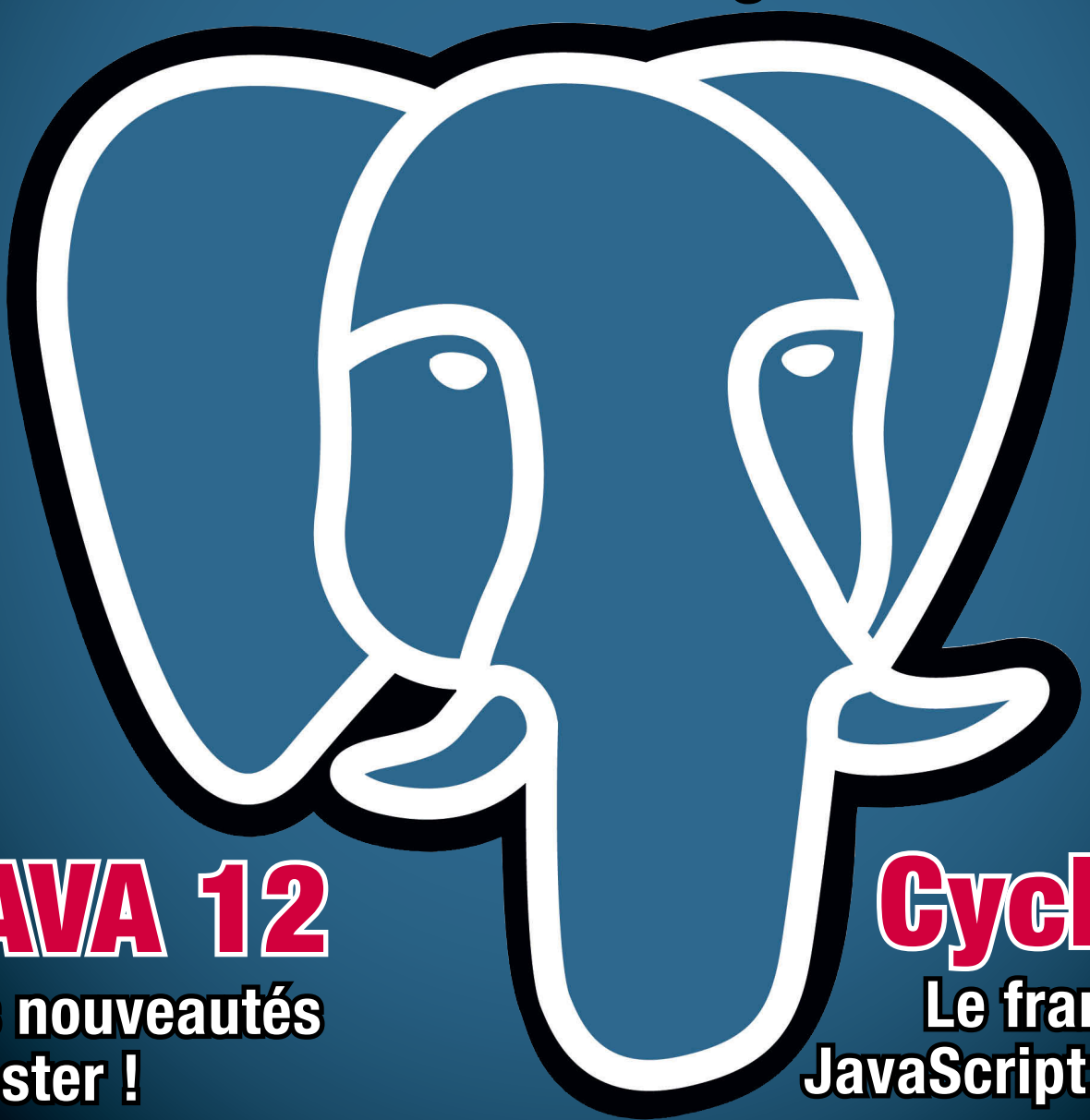
Programmez!

Le magazine des développeurs

228 AVRIL 2019

POSTGRESQL

La base de données à redécouvrir d'urgence !



M 04319 - 228 - F: 6,50 € - RD

JAVA 12

Des nouveautés à tester !

CycleJS

Le framework JavaScript badass

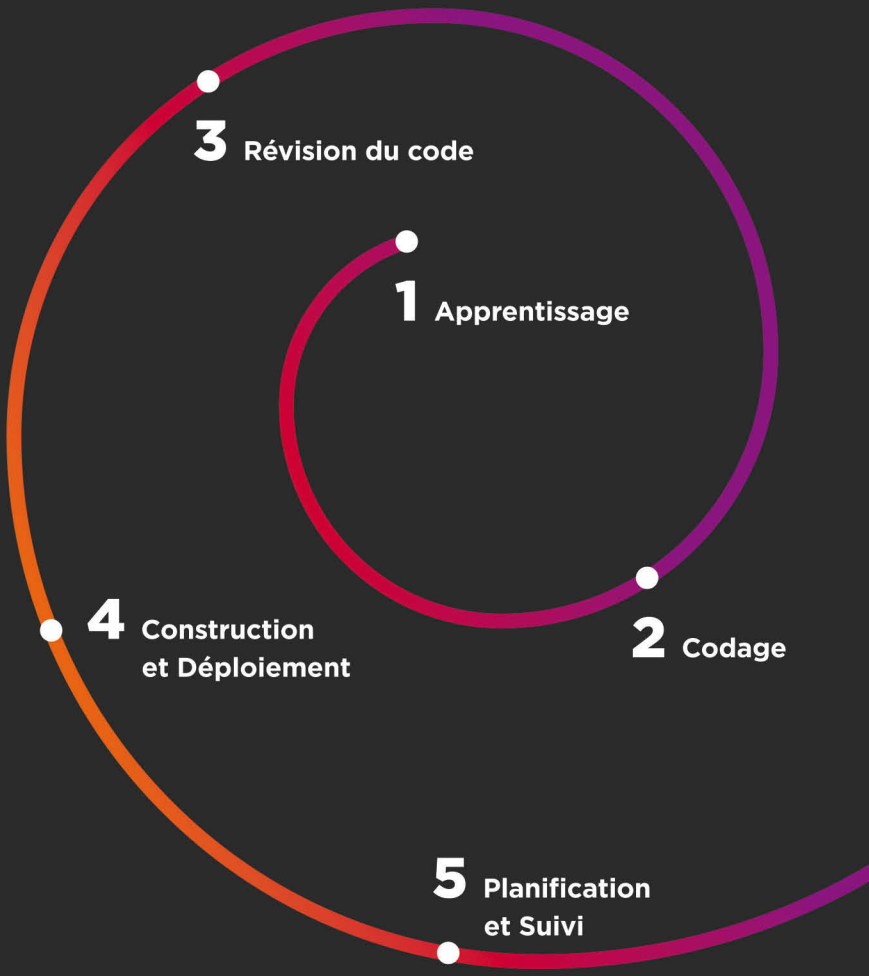
LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

Donnez-vous les moyens de réussir vos projets de développement logiciel

Chez JetBrains nous pensons que pour être performant, il est essentiel de disposer des outils de travail adaptés aux spécificités et besoins de son métier.

Nous proposons des produits qui couvrent **toutes les étapes du cycle du développement logiciel**, mais aussi les **dernière technologies** et la plupart des **langages de programmation**.



jetbrains.dev

6 millions de développeurs dans le monde utilisent nos outils dans les plus grandes entreprises.

citibank SIEMENS VALVE salesforce

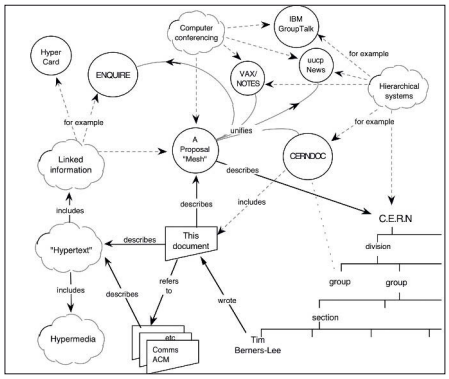
hp The New York Times Pinterest airbnb Expedia

EDITO

30 ans de web : finalement le web c'est vieux ! (et pas forcément mieux)

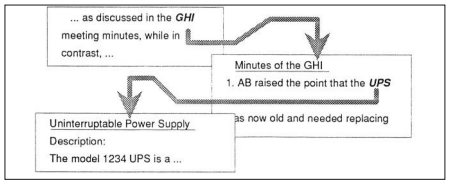
Mars 1989

Sir Tim Berners soumet sa 1ere proposition : aux origines du WWW



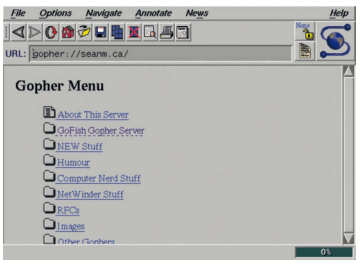
Novembre 1990

Sir Tim Berners, avec Robert Cailliau, soumet « WorldWideWeb : proposal for a HyperText Project »



Décembre 1990

Le 1er navigateur prend vie + le 1er serveur web tournant sur un NeXT Cube



Janvier 1993

La 1ere pré-version du navigateur Mosaic est disponible !

1999

Le protocole WAP ouvre les téléphones mobiles au monde du Web et de l'email !

Programmez ! célèbre les 30 ans du WWW !

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !

Programmez! #229, dès le 3 mai 2019

C# 8.0 : *les nouveautés*

Le cloud & le développeur : *comment le cloud impacte le code et le développeur ?*

PureOS veut se la jouer façon Canonical

Dans une récente publication, Purism, le constructeur à l'origine des ordinateurs portables open source annonce ses projets pour son système d'exploitation basé sur Debian PureOS. Celui-ci entend développer son OS afin de permettre à celui-ci de fonctionner aussi bien sur des ordinateurs portables que sur des smartphones. Un rêve que Canonical avait pendant un temps entretenu avec son Ubuntu Touch, mais qui a depuis été relégué aux oubliettes. Purism parviendra-t-il à réussir là où Canonical s'est cassé les dents ? L'éditeur semble en tout cas y croire dur comme fer.

Bug Bounty :
La Poste suisse comble ses premières failles

La Suisse a donné le coup d'envoi à son Bug Bounty visant à éprouver le système de votation en ligne mis au point par la Poste et Scytl. Les résultats ne se sont pas fait attendre. Plusieurs chercheurs universitaires en cryptographie se sont penchés sur le code source et ont découvert une faille qui permettait à un attaquant de manipuler le système de vote. Il s'agissait d'une faille identifiée pour la première fois en 2017, mais que le prestataire Scytl n'avait pas corrigée correctement. Des correctifs ont été promis et la faille sera rapidement corrigée. La Poste suisse a en revanche nettement moins apprécié que plusieurs chercheurs diffusent publiquement le code source du dispositif de vote, qui devait uniquement être consultable par les participants du Bug Bounty.

Vivendi et Ubisoft :
la saga touche à sa fin

Vivendi a annoncé avoir cédé les dernières parts qu'il détenait dans le capital d'Ubisoft. Ce faisant, le groupe de Vincent Bolloré met un point final à une saga qui avait débuté en octobre 2014, date à laquelle la société de divertissement avait annoncé le rachat d'actions au sein d'Ubisoft. Le groupe avait poursuivi ses acquisitions de manière très agressive au fil des mois, suscitant les craintes des dirigeants historiques du groupe qui redoutaient de voir Bolloré prendre le contrôle de la société. Le bras de fer s'était soldé par une victoire des Guillemot, la famille des fondateurs d'Ubisoft, et Vivendi a donc finalement revendu l'intégralité de ses parts. Mais le groupe reste l'actionnaire majoritaire de Gameloft, la société sœur d'Ubisoft consacrée au mobile. On ne se débarrasse pas de Vivendi aussi facilement.

Microsoft Store : nous, c'est les marges

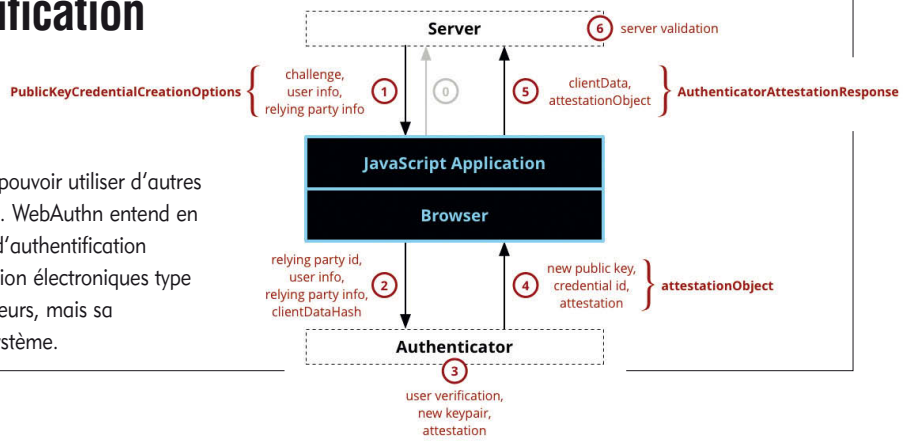
Si face aux App Store de Google ou d'Apple, Microsoft fait grise mine, ce n'est pas faute d'essayer. L'éditeur a en effet tenu sa promesse faite en 2018 et réduit sa commission sur les achats à 5 %. Ce qui signifie que l'éditeur de l'application touche 95 % du prix payé par l'utilisateur qui achèterait son application sur le Windows Store. Ce faisant, Microsoft prend le contre-pied de la concurrence, qui s'arroge généralement plutôt 30 % de commission sur les transactions effectuées via leurs plateformes.

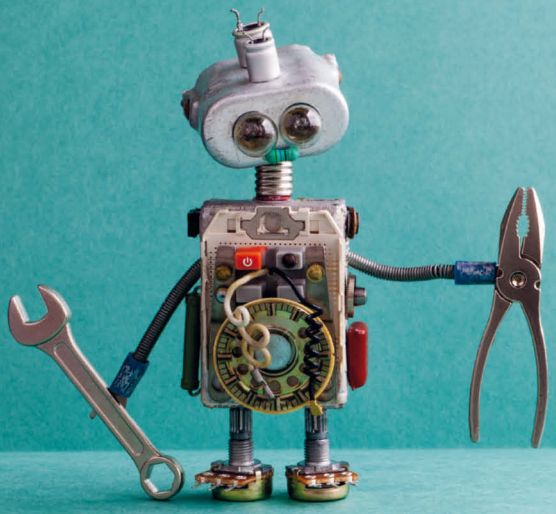
Sur le cloud, Iliad veut (aussi) casser les prix

Iliad est présent sur le marché du cloud à travers Scaleway, une filiale de sa division Online.net dédiée à l'hébergement. La société a annoncé en début d'année son intention de présenter « une soixantaine de nouveaux produits » dans les deux ans et commence fort en dévoilant une offre de GPU dans le cloud assorti de tarifs particulièrement bas : 1 euro de l'heure ou 500 euros par mois pour disposer d'un GPU Nvidia Tesla P100. Si Scaleway reste un acteur mineur face à des acteurs comme Google ou Microsoft, la filiale sort les crocs. Et après tout, casser les prix a plutôt bien marché pour Free.

WebAuthn finalisé, l'authentification web a son standard

L'API Web Authentication (WebAuthn) vient d'être officialisée par le W3C. Celle-ci vise à étendre les outils d'authentification pris en charge par les navigateurs afin de pouvoir utiliser d'autres méthodes que le traditionnel couple identifiant/mot de passe. WebAuthn entend en effet permettre aux navigateurs de supporter des méthodes d'authentification reposant sur la biométrie ou sur les dispositifs d'authentification électroniques type Yubikey. La norme était déjà supportée par plusieurs navigateurs, mais sa finalisation devrait conduire à une large adoption par l'écosystème.





6 & 7 AVRIL

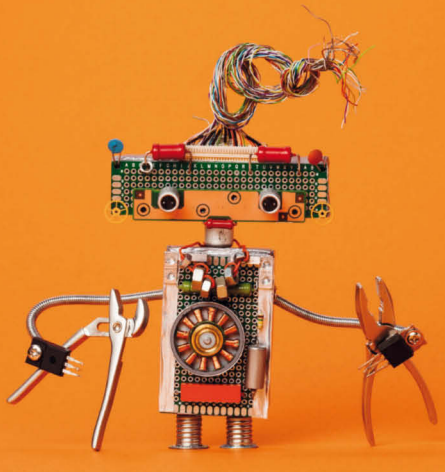
MakemeFest NANTES
PENSER - FAIRE - TRANSMETTRE

FOIRE INTERNATIONALE NANTES

DU 25 AU 29 AVRIL

MakemeFest ANGERS
PENSER - FAIRE - TRANSMETTRE

foire d'Angers



DU 30 MAI AU 2 JUIN

MakemeFest CHOLET
PENSER - FAIRE - TRANSMETTRE

FOIRE EXPO CHOLET

APPEL AUX MAKERS

Présentez votre projet à des milliers de visiteurs !

Inscription sur makeme.fr

DERNIER APPEL !

A partir du 5 mars : .NEXT On Tour / France

Dans toute la France, Nutanix mettra cette année l'accent sur les déploiements cloud hybride et multicloud, notamment en présentant les dernières solutions esquissées lors de son évènement .Next Europe qui s'est tenu à Londres en novembre dernier.
Les événements .NEXT on Tour de Nutanix sont l'opportunité pour les équipes IT de découvrir les dernières technologies cloud hybrides et leurs effets sur la transformation d'entreprise et de fixer le cap de leurs migration vers le cloud.
La dernière est Paris le 9 avril !
Agenda et informations :
https://www.nutanix.com/next/on-tour/

Avril

2 : AWS Summit Paris

Chaque année, AWS fait son show à Paris : toute une journée avec des sessions plénières, des sessions techniques et la présence de l'écosystème d'AWS.

A partir du 2 avril : tour de France de PC Soft

PC Soft, éditeur de WinDev, organise son tour de France, le WinDev TechTour 24.
Les dates :
2 : Strasbourg
3 : Genève
4 : Marseille
9 : Lyon

Pour en savoir plus :
https://www.pcsoft.fr/pcsoft/tdftech/2019/index.html

9 : Journée française des tests logiciels

Cet événement a su s'imposer au fil des années comme le rendez-vous fédérateur de la communauté française des tests logiciels. En effet, avec plus de 1 000 participants l'année dernière, 43 sponsors et 16 conférences thématiques mettant à l'honneur les témoignages de grands comptes et organisations, la JFTL est aujourd'hui un lieu d'échanges privilégié de l'ensemble des acteurs de ce marché, qui peuvent à cette occasion, rencontrer les experts les plus reconnus du secteur et échanger leurs expériences entre professionnels.
http://www.cftl.fr/JFTL/accueil/

Du 17 au 19 : Devoxx France

Le rendez-vous des communautés Java ! Un must à ne pas rater. https://www.devoxx.fr

23 & 24 : Android Makers

La communauté Android a rendez-vous au Beffroi de Montrouge près de Paris. 70 sessions, 4 tracks, +800 personnes attendues. Plusieurs stars du monde Android seront là. Citons Romain Guy, figure bien connu d'Android.
L'événement est co-organisé par le PAUG (communauté de Paris) et BeMyApp.

Mai

Du 15 au 17 : Riviera Dev / Sophia Antipolis

http://rivieradev.fr

16 & 17 : Newcrafts / Paris

Pour la sixième année consécutive, ne manquez pas LE rendez-vous indépendant et international des développeuses et développeurs professionnel.le.s passionné.e.s autour des technologies, pratiques et architectures modernes! https://ncrafts.io/
@ncraftsConf

{newcrafts}

23 & 24 : Mixit / Lyon

https://mixitconf.org/fr/

Juin

4 : Paris Container Day / Paris

Le 4 juin prochain aura lieu la quatrième édition du Paris Container Day. Le Paris Container Day est la conférence pionnière en France dédiée à l'écosystème des conteneurs et de ses bonnes pratiques. Pour celle nouvelle édition, Les participants pourront assister à une vingtaine de conférences techniques, retours d'expérience et des fast tracks. Le thème de cette année est "Standards & Craftsmanship". Organisée par Zenika. https://paris-container-day.fr

5 : Google Cloud Summit Paris

Venez découvrir la plateforme cloud de Google le 5 juin prochain à la Porte de Versailles. L'éditeur fera les choses en grand : +30 sessions sur tous les sujets chauds du moment, 4 salles en parallèles ! Tous les experts Google et les meilleurs partenaires seront présents sur scène. Bien entendu, de nombreuses entreprises viendront parler de leur projet Google Cloud Platform !
Site : https://cloudplatformonline.com/2018-Summit-Paris-speakers.html

7 : Best of Web / Paris

Une conférence qui réunit le Best Of des présentations données pendant l'année, réunis sur une journée à Paris ! Les organisateurs de meetups se réunissent pour proposer une journée avec les meilleurs talk qui ont été donnés dans leurs meetups respectifs. Contrairement à ce que le nom pourrait laisser entendre, on n'y parle pas exclusivement de web. On y parle de tous les sujets présents parmi les meetups fédérés, dont Human Talks, Duchess France, Scala UG, Software Crafters, ... qui ne sont pas des communautés spécialisées dans le web.

Venez échanger avec les communautés ! +500 personnes attendues.

Pour accéder aux formations du jeudi, il faut avoir un billet pour la conférence du vendredi et s'inscrire pour une somme symbolique de 10€ afin de réserver sa place. Il y a au maximum une quinzaine de participant pour que le formateur puissent travailler dans les meilleures conditions.

Les places sont relativement accessibles pour une conférences : 50€ le billet regular, et moins encore pour les billets early.
http://bestofweb.paris

Embarcadero Dev Tour!!!

Evenements à Lyon le 14 Mai, Marseille le 16 Mai. Bordeaux le 21 Mai et Nantes le 23 Mai.

Avec la participation des MVP Embarcadero :
Paul Toth, Patrick Premartin et Stephane Vander Clock.

Et de l'école Simplon :
Nous invitons l'école Simplon qui forme des développeurs Delphi. Ils nous parleront des programmes généralistes et spécifiques mis en place au sein de leur école.
Plus d'informations sur: https://www.barnsten.com/fr/evenements

---

La Région Auvergne-Rhône-Alpes

GRANDE ÉCOLE DU NUMÉRIQUE

SIMPLON.VE

11 au 14 juin : INFORSIF / Paris

Depuis 1982 le congrès INFORSID (INFormatique des ORganisations et Systèmes d'Information et de Décision) rassemble chaque année des chercheurs et des professionnels afin d'échanger sur les roblématiques d'ingénierie et de gouvernance des systèmes d'information.
Site : http://inforsid.fr/Paris2019/

14 : DevFest Lille

Le DevFest Lille Saison 3, est la 3ème édition d'un évènement complet de conférences sur le web, le mobile, le cloud et les objets connectés. Plus de 700 participants attendus cette année au Kinépolis de Lomme, le plus grand cinéma d'Europe. Une nouvelle saison pleine de surprises pour ce Devfest Lille sous le thème des Séries TV qui se déroulera le 14 juin prochain.
Site : https://devfest.gdglille.org

24 au 28 juin : COMPAS 2019 / Bayonne

Compas est la Conférence francophone en informatique autour des thématiques du parallélisme, de l'architecture et des systèmes. L'édition 2019 aura lieu du 24 au 28 juin à l'IUT de Bayonne.
Site : https://2019.compas-conference.fr/

27 & 28 : Sunny Tech / Montpellier

https://sunny-tech.io

Le coin maker

Makeme Fest Nantes :
du 12 au 14 avril sur la Foire de Nantes.
Makeme Fest Angers :
du 25 au 29 avril sur Foire d'Angers.

Septembre

11 &12 : Cloud Foundry Summit Europe

Cloud Foundry organise sa conférence européenne en Hollande. Deux jours pour comprendre, apprendre et échanger autour de Cloud Foundry, des outils, des technologies et des partenaires. Cloud Foundy est une couche PaaS que l'on retrouve chez Pivotal, Red Hat, IBM.
Informations :
https://www.cloudfoundry.org/event/eusummit2019/

16 & 17 : Agile en Seine 2019

La grande conférence sur l'agilité revient à Paris. Agile en Seine se veut une conférence multi-pratiques, ouverte à tous, dont l'objectif est de mettre en avant toute la diversité de ce que l'on nomme communément « Agilité » : du Scrum à l'agilité à l'échelle, en passant par le Design Thinking, le Kanban, le Lean Startup, le Lean, l'entreprise libérée, l'Holacracy …
Informations : https://www.agileenseine.com

CONFÉRENCES ORGANISÉES PAR XEBIA

27 juin : 2ème édition du DataXDay

https://dataxday.fr/

DataXDay est l'unique conférence Data qui réunit DataScientists, Data Engineers et Data Architects autour d'une quinzaine de conférences techniques. Venez échanger autour de la Data Science du PoC à la mise en production, des architectures Microservices et Serverless, de Craftsmanship, de sécurité, de Cloud, etc. Rendez-vous le 27 juin prochain.

7-8 Octobre : 4ème édition de la FrenchKit

La FrenchKit sera de retour pour une quatrième édition les 7 et 8 octobre prochains. La FrenchKit est la première conférence iOS et macOS en France. Durant ces 2 journées, certains des développeurs les plus reconnus de la communauté internationale traiteront un large éventail de sujets, des API Cocoa à Swift.

28 novembre : XebiCon, Build The Future

https://xebicon.fr/

A travers une soixantaine de conférences, la Xebicon vous donnera les clés pour tirer le meilleur des dernières technologies. Conférence techniques, retours d'experience clients, hands-on, venez partager et échanger sur les nouveautés technologiques autour du Cloud, de la Data, des nouveaux standards d'Architecture, des nouveaux langages Front-End, de l'IoT, de la culture DevOps, des transformations agiles à l'échelle ou encore de la mobilité.

Concours Prologin

Le concours national d'informatique Prologin est ouvert à tous les jeunes de 20 ans et moins. Ce concours entièrement gratuit est l'occasion pour les jeunes passionnés d'informatique de démontrer leurs talents et de rencontrer d'autres passionnés d'informatique. Prologin est organisé par des étudiants de l'EPITA, de l'École Normale Supérieure, ainsi que de l'École Polytechnique. La grande finale se déroulera en mai prochain à l'EPITA Paris : 36 heures de code pour les 100 meilleurs candidats ! Site : https://prologin.org

Girls Can Code !

Les stages Girls Can Code! c'est une semaine d'initiation à la programmation pour les collégiennes et lycéennes. On y apprend les bases de la programmation avec le langage Python, et de quoi se lancer dans des petits projets de son choix ! Les stages sont gratuits, entièrement organisés par des bénévoles de l'association Prologin.
Pour en savoir plus : https://gcc.prologin.org/

Merci à Aurélie Vache pour la liste 2019, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Véronique Loquet
@vloquet

Retour sur la conférence RSA de San Francisco

Pour sa 28e édition la conférence rassemblait une audience internationale de plus de 42 500 âmes. L'écosystème de la cybersécurité, des milliers de développeurs, hackers, CEO, grandes entreprises, start-up, investisseurs, ou agents fédéraux... sont venus débattre autour d'une foule de thématiques visant à créer un espace numérique plus sûr.

Toutes les études s'accordent. Le risque cyber dans le monde augmente, tant par son ampleur que sa fréquence. Les surfaces d'attaques ont explosé. Les offensives contre les entreprises ont plus que doublé au cours des 5 dernières années. Parmi la vague de cris d'alarme, celui de Ginni Rommety, PDG d'IBM, déclarant que : « La cybercriminalité est désormais la plus grande menace pour toutes les sociétés du monde ».

Sous l'influence des réseaux sociaux

Leur ascendant peut conduire à des conséquences désastreuses, notamment pour la stabilité de nos sociétés civiles. Dans le bruit des annonces, de nombreux reproches s'adressent directement aux réseaux sociaux.

Les scandales en cascade fondés sur la collecte massive de données, le non-respect de la vie privée, la diffusion de fake news et les algorithmes biaisés, posent la question des choix éthiques de cette industrie. Cindy Cohn, à la tête de l'EFF - Electronic Frontier Foundation, participait à un panel avec une représentante de Google.

Elle martèle le devoir impérieux de modifier la culture des entreprises, de s'entourer d'équipes pluridisciplinaires, avec par exemple des experts en sciences sociales, pour permettre de prendre des décisions éthiques lors de développement de projets technologiques ; permettre à ces experts d'oeuvrer ensemble et non isolément, afin d'anticiper les effets de leurs projets sur l'humanité. Elle déplore en outre le peu d'engagement des entreprises de la Silicon Valley au service du bien commun. Que cela puisse concerner les dispositifs de surveillance gouvernementaux, le vote électronique, les compteurs intelligents, les équipements médicaux, ou encore les scanners d'aéroport ; la démarche éthique est un process à part entière qu'il serait bon d'imposer.

Tensions géopolitiques

Elles sont bien présentes avec les accusations de hauts fonctionnaires rattachés au département de la sécurité intérieure. Qu'il s'agisse du piratage de la Russie lors des élections américaines de 2016, ou des tentatives de déstabilisation économique répétées de la Chine, accusée de vols de propriété intellectuelle et d'infiltration...

Alors que Huawei s'apprête à déployer son infrastructure 5G partout sur la planète, le gouvernement américain considère ce projet comme un danger pour la sécurité nationale. Il dénonce la proximité de Huawei avec le gouvernement chinois, et les répercussions d'un tel leadership si ses réseaux 5G devaient alimenter l'ensemble des équipements connectés, smart-cities, véhicules autonomes...

Les américains craignent l'hégémonie chinoise et la menace d'une capacité offensive sans précédent pour l'espionnage, et le sabotage d'infrastructures vitales. Si Huawei n'était pas présent à RSA cette année, son compatriote le géant du Cloud Alibaba concurrençait Amazon avec force.

L'immense show ne manque pas de thèmes à débattre, dont celui de l'absence d'une moitié du genre humain... **Où sont les femmes ?**

Depuis plusieurs années RSA s'intéresse à la sous-représentation des femmes dans les domaines des technologies de l'information, et en particulier dans le secteur de la cybersécurité. Confrontée à une pénurie de talents, l'industrie compte moins de 10% de femmes dans ses rangs.

Plusieurs sessions se sont penchées sur ce phénomène, et en aparté il fut question de trouver des solutions pour attirer les jeunes filles des collèges et lycées. Mandy Galante, Consultante au SANS Institut et enseignante en secondaire, précise que : « les filles douées en sciences et en mathématiques exercent ces matières avec passion durant le cycle du collège, mais s'orientent vers d'autres chemins lorsqu'elles arrivent dans le secondaire. »

Saluons la performance des organisateurs de la RSA, les panels entièrement masculins sont désormais en voie de disparition. C'est un positionnement volontariste dans le but de créer un environnement propice au dialogue inclusif.

La conférence offre des forums et lieux dédiés, allant jusqu'à la garde d'enfants. Le temps sera sans doute long pour parvenir à la parité, cependant les mentalités évoluent, les entreprises et les associations se mobilisent pour promouvoir la diversité. La bonne nouvelle est que chacun d'entre nous peut agir à son niveau. •

UN POINT SUR LES SALAIRES 2019

On parle beaucoup de la pénurie des profils techniques et de la pression sur les salaires. Le cabinet Hays a dévoilé début de l'année son étude annuelle sur les salaires. Pour la partie étude et développement, on constate toujours un grand écart entre les profils et l'opposition île de France et province.

Les principaux chiffres sont (fourchettes hautes) :

Développeur	0-3 ans	3-5 ans	5-8 ans	+8 ans
Java	42	47	57	60
.Net	41	46	55	60
PHP	38	43	52	60
Mobile	40	50	60	60
C / C++	40	45	50	60
Lead Tech	50	55	65	75
Intégrateur web	38	40	50	65
Webmaster	34	35	40	55

Plusieurs éléments nous gênent dans ce tableau. Tout d'abord, l'imprécision sur le profil développeur mobile. Oui, aujourd'hui, le dév mobile doit pouvoir être polyglotte (Android / iOS), mais tout de même. Le salaire estimé au-delà de 8 ans d'expérience pose problème, notamment par rapport aux autres profils dévs. D'une manière générale, le plafonnement à 60 000 € bruts nous perturbe :

1 – beaucoup de développeurs rêveraient de ce niveau.
2 – à ce niveau d'expérience, et selon les postes, le salaire risque d'être très fluctuent. Un expert C++ peut prétendre à plus.
3 – nous sommes encore largement en dessous d'autres pays.

Webmaster est aujourd'hui un profil dont il faut se méfier et qui doit disparaître. Même remarque pour intégrateur web. Ce sont des profils qui veulent tout dire et ne rien dire ! Une mise à jour de la nomenclature ne ferait pas de mal. Où sont les profils Python, JS, Angular, etc. ?

On constate que les dévs plafonnent à 60 000 alors que les profils CTO, architecte affichent 75 000 €. N'est-il pas tant de revaloriser le développeur ?

Le développeur est-il fidèle ?

Codingame a publié une étude sur les développeurs et le recrutement. Elle se base sur 9 000 personnes. Les points clés sont : Python est LE langage en vogue, IA / Machine Learning / jeux sont les grandes tendances. Mais attention : le salaire, la vision de l'entreprise sont des éléments importants.

Recruter c'est bien, garder c'est encore mieux !

Une des difficultés est de savoir retenir / garder les développeurs. C'est un défi qu'il ne faut pas négliger. 72,3 % des répondants seraient prêts à changer de postes / d'entreprise si une opportunité se présentait.

55,5 % seraient prêts à changer d'entreprise dans les 3 ans. La majorité peuvent partir entre 2 et 3 ans. Entre 4 et 5, ils sont 21,4 % ! Ce qui reste énorme.

Cela ne veut pas dire que tous les développeurs passeront à l'acte mais cela démontre tout de même un problème. Quels sont les facteurs pouvant inciter un développeur à voir ailleurs ? La première motivation est le salaire puis l'opportunité de progresser. Suivent : juste le changement d'air, aller dans une entreprise avec d'autres valeurs. Le mauvais management et une relocalisation géographique arrivent 6e et 7e.

Dans le détail, l'étude pointe trois grands éléments qui motivent :

- 1 le salaire ;
- 2 la flexibilité sur les horaires (et la possibilité de faire du télétravail) ;
- 3 les challenges techniques / technologiques.

L'étude met en avant le salaire moyen d'un développeur :

- 1 Etats-Unis : 100 515 $
…
- 2 Allemagne : 67 611 $
…
- 8 France : 53 515 $
…
- 19 Portugal : 30 204 $

Il est toujours délicat d'opposer les salaires moyens car il existe de nombreux écarts sur le calcul du salaire net, le coût des assurances / mutuelles, le niveau de vie, etc. Disons que cela donne une idée des possibilités. Clairement, un effort reste à faire pour revaloriser le développeur.

Au-delà ce qui peut perturber le développeur ou l'inciter à partir ce sont les problèmes d'organisation, les problèmes de planification, une vision floue ou absente sur les projets / les technologies, sans oublier les plannings irréalistes.

Oui on cherche des développeurs mais pas non plus n'importe quoi

Dans les media, on entend souvent « on manque de développeurs ». Nous l'avons dit à de nombreuses reprises que ce soit dans le magazine ou en conférence, oui, nous manquons de développeurs mais pas sur tous les profils. Et finalement, ce qui manque ce sont de bons développeurs. Celles et ceux qui apportent une valeur technique, une maîtrise. D'ailleurs, l'étude ne dit pas autre chose : un des défis est de trouver des candidats avec le bon profil. Il n'est pas rare d'avoir 7 à 8 CV / profils écartés sur 10 reçus. Une des difficultés pour l'entreprise et les équipes techniques est de savoir si le candidat a le profil réellement recherché. Mais une des questions est de savoir si on se focalise (encore) sur le diplôme ou sur les qualités réelles de la personne. Trop se focaliser sur le diplôme c'est clairement le risque de passer à côté du bon profil !

Peu de surprises sur les langages mis en avant : Javascript, Java, Python, C++, C, C#, PHP.

Plus gênants, les ressources humaines des entreprises cherchent toujours et encore le mouton à 42 pattes : le développeur full-stack. Aujourd'hui, les plateformes sont tellement complexes, avec une multitude de technologies, qu'il est impossible de tout maîtriser. Comment espérer réellement trouver un dév fullstack ? On évoque le profil développeur back-end. Avouons que ce profil est pour nous étrange et trop fourre-tout. Enfin, le 3e profil recherché est DevOps (sic !).

Où chercher les profils techniques ? Le réseau n°1 de l'étude est Linkedin puis son propre réseau, la famille, les amis, les communautés, etc. Mais on s'aperçoit aussi que le développeur est mitigé sur le fait de se déplacer : la même ville / région est plébiscitée mais l'international peut être aussi une motivation !

François Tonic

NUC8i5BEK : un NUC Core i5 nouvelle génération homogène

Intel propose une solide gamme de barebones, les fameux NUC. Le fondeur offre des machines de qualité : bien construites, personnalisables, avec une belle connectique. Oui ce n'est pas forcément le modèle le moins cher mais sans aucun doute l'un des meilleurs du marché !

Nous utilisons à la rédaction une génération ayant déjà 4 ans. La taille reste identique, seule l'épaisseur change. Le format de ce NUC est toujours aussi agréable. Le contenu de la boîte se réduit au NUC, à l'alimentation, au support VESA. Nous aimons beaucoup la robe noire. Côté connectique, ce NUC propose une panoplie séduisante :

- HDMI 2.0a plein format (ouf, car sur les anciens modèles nous avions un mini HDMI) ;
- Ethernet 1 Gb ;
- 2 USB 3.1 ;
- 1 port Thunderbolt / USB C supportant DisplayPort 1.2 et USB 3.1 Gen2 ;
- encoche sécurité ;
- lecteur SD ;
- En face avant, 2 USB 3.1 et prise audio.

A l'intérieur, nous trouvons un processeur i5 8e génération, un GPU Iris Plus Graphics 655, 2 slots mémoire format DDR4 (jusqu'à 32 Go), un slot M.2 pour le stockage. Le NUC peut recevoir une mémoire Optane. On dispose aussi du WiFi et du Bluetooth 5.

Le modèle que nous avions en test était équipé de 8 Go de mémoire et d'un disque SSD d'origine Intel de 256 Go. Le CPU était un i5 8259U 4 coeurs. Ce processeur a été introduit en 2018 ; il se destine avant tout aux machines mobiles et non aux desktops. Il supporte maximum 32 Go de mémoire, d'où la limitation de la RAM.

Pour quels usages ?

Nous avons évalué les performances brutes de la machine avec NovaBench. Le processeur obtient un score de 692, 243 pour le GPU intégré et 21 FPS en Direct3D11. Le disque flash affiche de très bons débits : 1232 Mb en écriture et 1114 Mb en lecture. Ce qui est très homogène.

Si on en juge Novabench, nous serions sur une machine milieu de gamme pour de la bureautique, Internet, un peu de jeux (mais sans forcer sur la bête). La machine étant homogène, peu de surprise. Idem pour le développement : la machine suffira. Par contre, on évitera le gaming intensif, la 3D ou encore la réalité virtuelle / augmentée. Pour ces usages, mieux vaut la version i7, chargée en RAM et SSD. La partie GPU est le point faible, comme souvent dans ce type de machine. Il sera toujours possible d'ajouter un boîtier eGPU qui coûtera le double de la machine.

La connectique est séduisante, mais l'unique port Thunderbolt 3 / USB C est une faiblesse surtout quand on commence à connecter un SSD haute performance, un eGPU. Sur la limitation à 32 Go, due aux processeurs, ce n'est pas un problème au vu de l'usage de la machine.

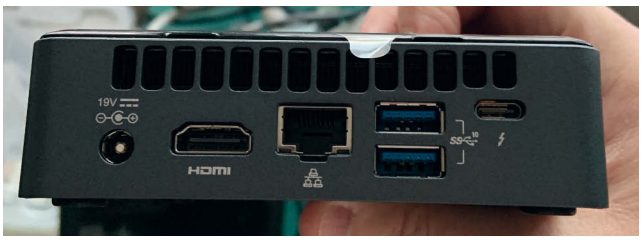
Pas si silencieux

Si en usage « standard », le NUC saura être discret et silencieux, c'est une autre histoire quand vous chargez la machine.

Immédiatement, la ventilation s'entend et le dégagement de chaleur n'est pas anodin. Prévoyez un bon dégagement autour du boîtier pour assurer une bonne aération. Dommage.

Côté tarif, le boîtier nu coûte environ 430 € + 70 € pour 8 Go de RAM (2 barrettes) + 70 € pour 256 Go (SSD, format M 2, modèle Intel 760p NVMe). Pour -500 €, on dispose d'une machine homogène. La version i7 coûte environ 100 € de plus. Mais là encore, la RAM est limitée à 32 Go.

Notre note : 14/20



Les +

- Ouverture rapide (nécessite un outil)
- Mémoire & stockage amovible
- Taille réduite
- Thunderbold 3 à 40 Gb
- Support VESA dans la boîte
- Silencieux (ou presque)
- Tarif
- Stockage SSD (de notre machine de tests)

Les –

- 1 seul port Thunderbolt
- Alimentation externe
- Format du câble d'alimentation
- Dégagement de chaleur
- Bruit quand on charge le NUC
- GPU

Roadmap 2019 des langages

Chaque mois, Programmez ! vous proposera la roadmap des sorties des langages, frameworks, etc.

1ER SEMESTRE 2019

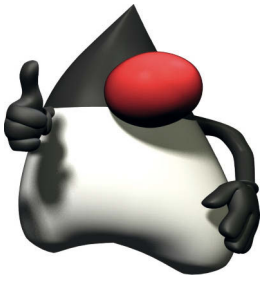
Kotlin 1.3.30

Date de sortie : mars / avril

Les principales nouveautés attendues / annoncées : après une version 1.3 plutôt réussie, les versions se succèdent. La 1.3.30 arrivera dans quelques semaines. Il est demandé à la communauté d'évaluer plusieurs éléments : vue séparée des variables dans la fenêtre outil, trace Async dans les coroutines, mode interactif pour les fichiers Scratch, possibilité de faire des commentaires TODO sur plusieurs lignes.

Java 12

Date de sortie : mars/avril



Les principales nouveautés attendues / annoncées : voir dossier dans ce numéro

Au-delà : Java 13 doit être déployée en septembre prochain. Cette version n'est pas LTS, support long terme, et elle sera en fin de vie à la sortie de la version 14, prévue en mars 2020. A l'heure où nous écrivons, nous savons encore peu de choses sur cette version. On aura une unification de plusieurs méthodes appartenant à la classe GraphicsEnvironment :

- getCenterPoint retounera les coordonnées du centre d'un affichage primaire, sur toutes les plateformes ;
- getMaximumWindowBounds : retournera les bounds de l'affichage.

Sur macOS, le Swing Motif Look and Feel est déprécié et ne sera plus supporté. Le code source sera retiré dans une future version de la JDK.

Angular 8

Date de sortie : mars / avril

Les principales nouveautés attendues / annoncées : Angular évolue très vite et les versions se succèdent à une cadence élevée, tous les 6 mois environ. La v8, qui arrive maintenant, introduit plusieurs améliorations :

- Sur la CLI : possibilité d'utiliser ES5 et ES2015+ avec une amélioration sur le chargement et le build du code JS ;
- Pré-version de l'Opt-in Ivy ;
- Web Worker : amélioration sur les performances et la parallélisation des applications ;
- Mise à jour des dépendances, notamment outils.

Au-delà : la v9 est prévue pour cet automne.

Rust

La communauté annonce la prochaine version du langage, la 1.34 pour le 11 avril. La 1.35 arrivera fin mai.

Pour suivre la roadmap : https://forge.rust-lang.org

SWIFT 5

Date de sortie : printemps

Les principales nouveautés attendues / annoncées : initialement attendue pour fin 2018, Swift 5.0 arrivera dans quelques semaines. Les équipes terminent de stabiliser les branches. Cette version est cruciale pour le langage car il s'agira de la première version à stabiliser l'Application Binary Interface (ABI). L'ABI décrit l'interface entre les applications et l'OS, entre l'application et les bibliothèques, etc. Cette stabilité ABI permettra d'avoir une compatibilité entre les applications et les bibliothèques, utilisant différentes versions du langage. La v5 promet aussi de réduire la taille des binaires générés. On aura droit à un nouvel attribut (@dynamicCallable), des nouveautés sur la gestion des packages ou encore dans le compilateur.

Au-delà : la version 5.1 arrivera très vite après la 5.0. Il ne faut pas s'attendre à d'importantes évolutions.

Ruby on Rails 6

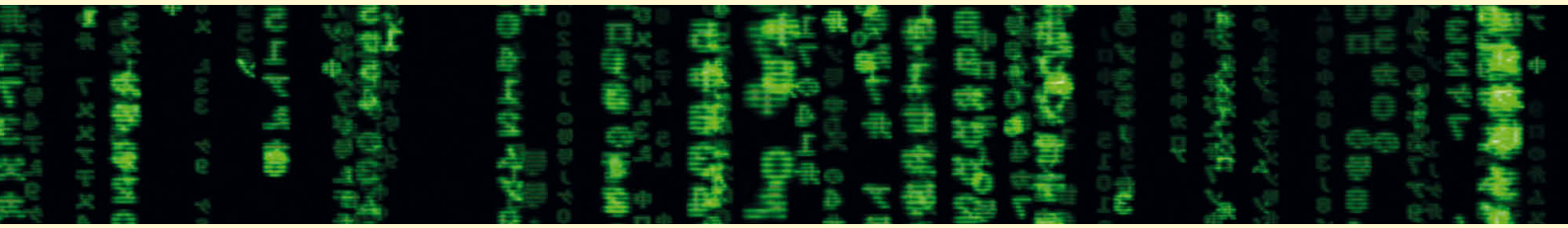
Date de sortie : printemps

Les principales nouveautés attendues / annoncées : parallèlement au langage, Rails continue à évoluer de son côté. Les grosses nouveautés attendues concernent les actions sur les mailbox, les évolutions sur les fonctions de tests, particulièrement sur le testing action cable et les tests parallèles.

MEETUP PROGRAMMEZ! #4

SPÉCIAL

C++ 17 et au-delà



25 avril à partir de 19h

Intervenant :

Christophe Pichaud
Architecte Microsoft chez
'M Applications by Devoteam'

Où

Arolla
21, rue du bouloi
75001 Paris
Batiment D au fond à gauche

Métros / RER :
Métro 1 : station Louvre-Rivoli
Métro 4 / RER A, B & D : Châtelet - Les Halles

Inscription et informations sur www.programmez.com

Nos prochains meetups :
28 mai : *Java 12*
25 juin : *quantique, vers l'infini et au-delà*

Meetups organisés par [Programmez!] Le magazine des développeurs

2E SEMESTRE 2019

Goland 1.13

Date de sortie : août/septembre

Les principales nouveautés attendues / annoncées : cette version doit activer le mode module par défaut tout en déconseillant le mode GOPATH. On aura aussi des nouveautés sur les génériques, la gestion des erreurs.

Au-delà : Go 2 sera LA grosse évolution du langage Go même si les équipes ne veulent pas faire une version de rupture mais des évolutions au fil des versions. Un des changements sera la manière de définir les évolutions et comment la gouvernance fonctionne. L'idée est que la communauté soit plus impliquée. La compatibilité avec Go 1.x sera un des aspects cruciaux. Pour le moment, le planning de Go2 reste à préciser.

C# 8.0

Date de sortie : été / automne

Les principales nouveautés attendues / annoncées : cette nouvelle évolution du langage phare .Net doit arriver avec .Net Core 3.0. Parmi les nouveautés attendues :

- Les types de références nullables doivent en finir avec les exceptions `null`. Pour cela, elles vous empêchent de mettre `null` dans des types de référence ordinaire comme `string`. Par défaut, ces types seront non nullables ! Cela pourrait avoir un impact sur les codes existants ;
- Les flux asynchrones ;
- Types range et index ;
- Expressions Switch.

Au-delà : il faut s'attendre à des itérations périodiques comme pour C# 7. Pas de détails pour le moment.

Python 3.8

Date de sortie : octobre

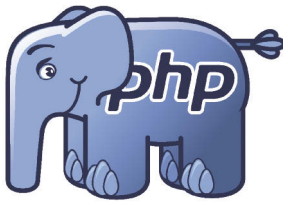
Les principales nouveautés attendues / annoncées : plusieurs nouveautés devraient plaire aux développeurs. Tout d'abord, on bénéficie du nouveau paramètre pythonpycacheprefix. Il sera à préférer au __pycache__ présent dans les sous-répertoires des dossiers sources. On disposera aussi de la méthode as_integer_ratio(), dans le type int. Le contrôle-C, SIGINT, sera modifié pour éviter les problèmes liés à l'exception KeyboardInterrup. Plusieurs modules auront droit à des améliorations comme asyncio avec ProactorEventLoop ou encore dans gc avec de nouveaux paramètres dans le get_objects().

Pour plus de détails : https://docs.python.org/dev/whatsnew/3.8.html

Au-delà : en toute logique, la prochaine version majeure sera la 3.9. Pour le moment, pas grand-chose n'a été communiqué. Cette version ne devrait pas arriver avant 2021.

PHP 7.4

Date de sortie : décembre (?)



Les principales nouveautés attendues / annoncées : la 7.4 doit apporter le préchargement (preloading), au cœur de PHP pour pouvoir améliorer les performances. Ce mécanisme permet de charger l'ensemble des fichiers PHP dès le démarrage et ainsi améliorer les accès et assurer une disponibilité constante de ceux-ci. Par contre, en cas de changement des fichiers, il faudra redémarrer le serveur. On notera aussi la disponibilité des propriétés typées, l'extension FFI (Foreign Function Interface) pour appeler du code C, le nouveau opérateur Null Coalescing, l'apparition de nouvelles méthodes (__serialize & __unesrialize).
On notera aussi le retrait de PEAR ou la dépréciation de ext/wwdx.
Attention : la 7.4 n'est pas encore totalement figée. Des changements peuvent intervenir.

Au-delà : après la 7.4, difficile d'y voir clair. Il était question d'une v8. Pour le moment, rien de précis.

Ruby 2.7

Date de sortie : ?

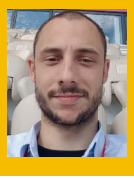
Les principales nouveautés attendues / annoncées : le langage Ruby continue à évoluer. La 2.6 est sortie fin 2018 notamment avec un nouveau compilateur JIT. On bénéficie aussi d'un nouveau module RubyVM ::AbstractSyntaxTree. Il analyse une chaîne de caractères et retourne les nœuds d'arbre syntaxique. Un travail d'optimisation a été réalisé.

COURANT 2020

C++20

Date de sortie : 2020

Les principales nouveautés attendues / annoncées : les spécifications de C++20 sont désormais figées et un premier document complet sera disponible cet été. Cette future version du langage mettra en avant deux nouveautés : les modules et les coroutines. Les modules constituent une nouvelle alternative aux fichiers d'en-tête qui apportent un certain nombre d'améliorations, notamment en isolant les effets des macros et en permettant des compilations évolutives. En 35 ans c'est première fois que C ++ ajoute une nouvelle fonctionnalité permettant aux utilisateurs de définir une limite d'encapsulation nommée.
Une coroutine est une unité de traitement qui s'apparente à une fonction (ou routine) avec cette différence que si une sortie du corps d'une fonction met fin à l'exécution de celle-ci, la sortie de la coroutine suspend seulement son traitement qui peut ensuite reprendre, l'état du traitement à la sortie étant conservée. Une coroutine peut être vue comme un morceau de programme qui conserve son état, mais qui n'a pas de thread d'exécution. Les coroutines peuvent notamment être utilisées pour des itérateurs et des générateurs.



Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Java / Android
https://www.ssaurel.com

Java 12 : une nouvelle version majeure qui a tout d'une mineure !

Tout juste 6 mois après la sortie de Java 11, Oracle vient de sortir Java 12. C'est la nouvelle version majeure de Java qui a tout d'une version mineure tant son contenu aura peu d'impact sur le commun des développeurs Java. Le point essentiel de cette nouvelle mouture étant avant tout qu'Oracle se tient, pour le moment, à son nouveau de cycle de release annoncé en 2018. Si cette nouvelle version n'a rien de majeur, il est toujours important de rester informé des évolutions du JDK. C'est que nous vous proposons de faire au travers de cet article.

niveau 100

Comme on pouvait l'imaginer à l'annonce du nouveau cycle de release du JDK annoncé par Oracle en 2018, il n'y aura plus la même excitation lors de la sortie de chaque version majeure du JDK tous les 6 mois. Certaines intègreront des nouvelles fonctionnalités importantes issues de projets au long cours menés en parallèle. D'autres se contenteront d'apporter des évolutions au JDK n'ayant pas d'impacts directs sur les programmeurs Java mais qui restent néanmoins appréciables.

Tout ceci n'est pas vraiment grave car l'essentiel était avant tout qu'Oracle montre que la plateforme a bel et bien repris sa marche en avant. Sur ce point, Java 11 et Java 12 sont donc un succès important pour le futur de Java. Cette version 12 de Java s'articule autour de 8 grandes nouveautés que nous allons détailler.

Un Garbage Collector à faible temps de pause

Les Garbages Collectors sont décidément à l'honneur avec les dernières versions de Java puisqu'après avoir introduit le Z Garbage Collector (ZGC) dans Java 11, Oracle introduit cette fois Shenandoah. Ce dernier est un Garbage Collector à faible temps de pause utilisable à titre expérimental pour le moment.

Shenandoah utilise un algorithme conçu spécifiquement pour réduire les temps de pause lors du processus de Garbage Collection. Pour ce faire, il va réaliser des tâches d'évacuation de manière concurrente avec les threads Java en cours d'exécution. De fait, avec Shenandoah, les temps de pause deviennent indépendants de la taille de la heap. Ceci signifie que vous aurez des temps de pause constants identiques que la heap soit de 200 MB ou de 200 GB.

Ce nouveau Garbage Collector a été créé pour les applications valorisant la réactivité et les temps de pause très courts et surtout prévisibles. Le but ici n'étant pas de corriger tous les problèmes de pause de la JVM. Les temps d'arrêt réalisés pour des raisons autres que la Garbage Collection, tels que les problèmes de temps de retour au point de sécurité ou la surveillance de l'inflation de la heap sont hors du périmètre de Shenandoah.

Si vous souhaitez tester Shenandoah sur vos applications Java, il faudra utiliser les options suivantes au lancement de la JVM :

```
-XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC
```

Microbenchmark Suite

Cette fonctionnalité s'adresse avant tout à ceux prenant part au développement du JDK. Elle consiste en l'ajout d'une suite d'outils permettant de faire des micro-benchmarks directement au sein du code source du JDK. Ainsi, si votre but est de simplement télécharger Java 12 afin de l'utiliser pour exécuter vos applications, il est fort probable que vous ne trouviez aucune utilité à Microbenchmark Suite. Dans le cas où le sujet vous intéresserait particulièrement, sachez que Microbenchmark Suite est basée sur la suite d'outils Java Microbenchmark Harness (JMH).

Pour lancer cette suite, il faudra que vous récupéreriez le code source du JDK afin de pouvoir le construire directement sur votre ordinateur. La suite se passe en ligne de commande dans un terminal :

```
$ cd jdk-src
$ sh make/devkit/createJMHBundle.sh
$ ./configure --with-jmh=build/jmh/jars --enable-headless-only
$ make test TEST="micro:java.lang.reflect"
```

... après plusieurs lignes de sortie console ...

```
Test selection 'micro:java.lang.reflect', will run:
* micro:java.lang.reflect

Running test 'micro:java.lang.reflect'
# JMH version: 1.21
# VM version: JDK 12-internal, OpenJDK 64-Bit Server VM, 12-internal+0-adhoc.ubuntu.jdk-src
# VM invoker: /home/ubuntu/jdk-src/build/linux-x86_64-server-release/images/jdk/
bin/java
# VM options: --add-opens=java.base/java.io=ALL-UNNAMED
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Average time, time/op
# Benchmark: org.openjdk.bench.java.lang.reflect.Clazz.getConstructor

# Run progress: 0.00% complete, ETA 01:31:40
# Fork: 1 of 5
# Warmup Iteration   1: 19.849 ns/op
# Warmup Iteration   2: 19.067 ns/op
```

```
# Warmup Iteration   3: 20.044 ns/op
# Warmup Iteration   4: 20.050 ns/op
# Warmup Iteration   5: 20.061 ns/op
Iteration   1: 20.037 ns/op
Iteration   2: 20.019 ns/op
Iteration   3: 20.070 ns/op
Iteration   4: 20.052 ns/op
Iteration   5: 20.024 ns/op
```

.. l'outil continue à réaliser de nombreux tests …

Extension des expressions switch

Voilà enfin une nouveauté qui aura un impact sur les développeurs Java au quotidien. Elle concerne les expressions *switch* qui sont étendues avec Java 12. Cette nouvelle fonctionnalité est en preview. En effet, des fonctionnalités affectant le langage Java lui-même ou la JVM pourront désormais être intégrées dans une version majeure afin de permettre aux développeurs de la communauté de la tester dans des conditions réelles et de faire leurs retours ensuite à Oracle. Les implémentations sont pleinement spécifiées et implémentées mais pas forcément permanentes.

Dans ce cas précis, l'extension du *switch*, qui devient une expression, vise à simplifier la vie des développeurs au quotidien tout en préparant l'introduction future du pattern matching en Java qui s'appuiera sur les expressions *switch*. Concrètement, les changements concernant le *switch* sont de deux natures :

Introduction de la nouvelle syntaxe *case L ->* qui supprime l'obligation d'utiliser une instruction *break* tout simplement parce que seule l'instruction ou le bloc d'instruction se situant après la flèche sont exécutés.

Le *switch* peut désormais être une expression. Il peut donc avoir une valeur ou bien retourner une valeur.

Tout ceci nous donne les exemples d'utilisation du switch étendu suivants :

```
public class SwitchJava12 {

 public static void main(String[] args) {

  // Ici, on considère que args[0] correspond au jour de la semaine
  final int day = Integer.valueOf(args[0]);

  // Utilisation du switch traditionnel
  switch (day) {
   case 1:
   case 2:
   case 3:
   case 4:
   case 5:
    System.out.println("Jour de la semaine");
    break;
   case 6:
   case 7:
    System.out.println("Week-end");
    break;
   default:
    System.out.println("Invalide");
  }

  // Utilisation de la nouvelle syntaxe case L ->
  // Plus besoin d'utiliser un break lorsqu'il
  // n'y a qu'une instruction derrière la flèche
  switch (day) {
   case 1, 2, 3, 4, 5 -> System.out.println("Jour de la semaine");
   case 6, 7 -> System.out.println("Week-end");
   default -> System.out.println("Invalide");
  }

  // Utilisation d'un switch en tant qu'expression
  final String attr = switch (day) {
   case 1, 2, 3, 4, 5 -> "Jour de la semaine";
   case 6, 7 -> "Week-end ";
   default -> "Invalide";
  };

  System.out.println(attr);

 }
}
```

En repartant de cet exemple des jours de la semaine, on peut utiliser un *switch* en tant qu'expression avec la syntaxe *case L ->* et un bloc d'instructions placé à droite de la flèche :

```
public static boolean isWeekend(String day) {
  return switch (day) {
    case "lun", "mar", "mer", "jeu", "ven" -> false;
    case "sam", "dim" -> true;
    default -> {
      if (day.startsWith("s") && day.startsWith("d")) {
        System.out.println("Cela semble être un jour du Week-end");
        break true;
      }

      System.out.printf("Jour inconnu : %s%n", day);
      break false;
    }
  };
}
```

Les switchs étendus étant en mode preview dans Java 12, vous devrez les activer à la compilation et à l'exécution afin de pouvoir les tester dans vos programmes. Pour cela, il faudra procéder de la sorte :

```
$ javac -d classes --enable-preview --release 12 Test.java
$ java -classpath classes --enable-preview Test
```

JVM Constants API

Un fichier de classe Java possède un pool de constantes stockant les opérandes pour les instructions bytecode dans la classe. Le contenu du pool de constantes décrit soit des artéfacts d'exécution tels que des classes et des méthodes, soit des valeurs simples telles que des chaînes et des entiers. Toutes ces entrées peuvent servir d'opérandes pour l'instruction bytecode *ldc* qui charge les constantes à utiliser par d'autres instructions. Ces entrées peuvent également être utilisées dans la liste d'arguments statiques d'une

méthode bootstrap pour l'instruction bytecode *invokedynamic*. Au runtime, l'exécution d'une instruction ldc ou invokedynamic provoque la résolution de la constante en une valeur de l'un des types Java standards tels que *Class*, *String* ou *int*.
La fonctionnalité JVM Constants API introduit un modèle de constantes au sein du pool de constantes. Cette API peut être utile pour les programmes et les bibliothèques de code qui traitent les instructions de bytecode et manipulent les fichiers de classe. Contrairement aux switchs étendus, cette nouveauté n'aura pas d'impacts directs sur la plupart des développeurs Java. Elle profitera aux langages basés sur la JVM tels que Scala ou Groovy mais également aux outils manipulant les classes et les méthodes directement au sein du bytecode. Cette nouvelle API est située dans le package *java.lang.constant*.

One AArch64 Port, Not Two

Java peut fonctionner sur les architectures ARM. Le JDK contient même deux ports ARM 64 bits actuellement ! Pour les familiers avec la structure des dépôts de code source du JDK, les principales sources de ces ports se situent précisément dans les répertoires suivants :

```
src/hotspot/cpu/arm
open/src/hotspot/cpu/aarch64
```

Tout ceci est quelque peu redondant, il faut bien l'admettre. L'évolution "One AArch64, Not Two" visait donc à supprimer toutes les sources liées au port arm64 afin de ne conserver que le port ARM 32 bits ainsi que le port aarch64 64bits.
La suppression du port arm64 permettra à tous les contributeurs du JDK de focaliser leurs efforts sur une seule implémentation 64 bits pour ARM. Elle élimine également le travail supplémentaire requis pour maintenir deux ports. Il est évident que cette évolution intéressera assez peu la plupart des développeurs Java. Néanmoins, elle fait partie des nouveautés de Java 12 et à ce titre, elle méritait d'être mentionnée.

Default CDS Archives

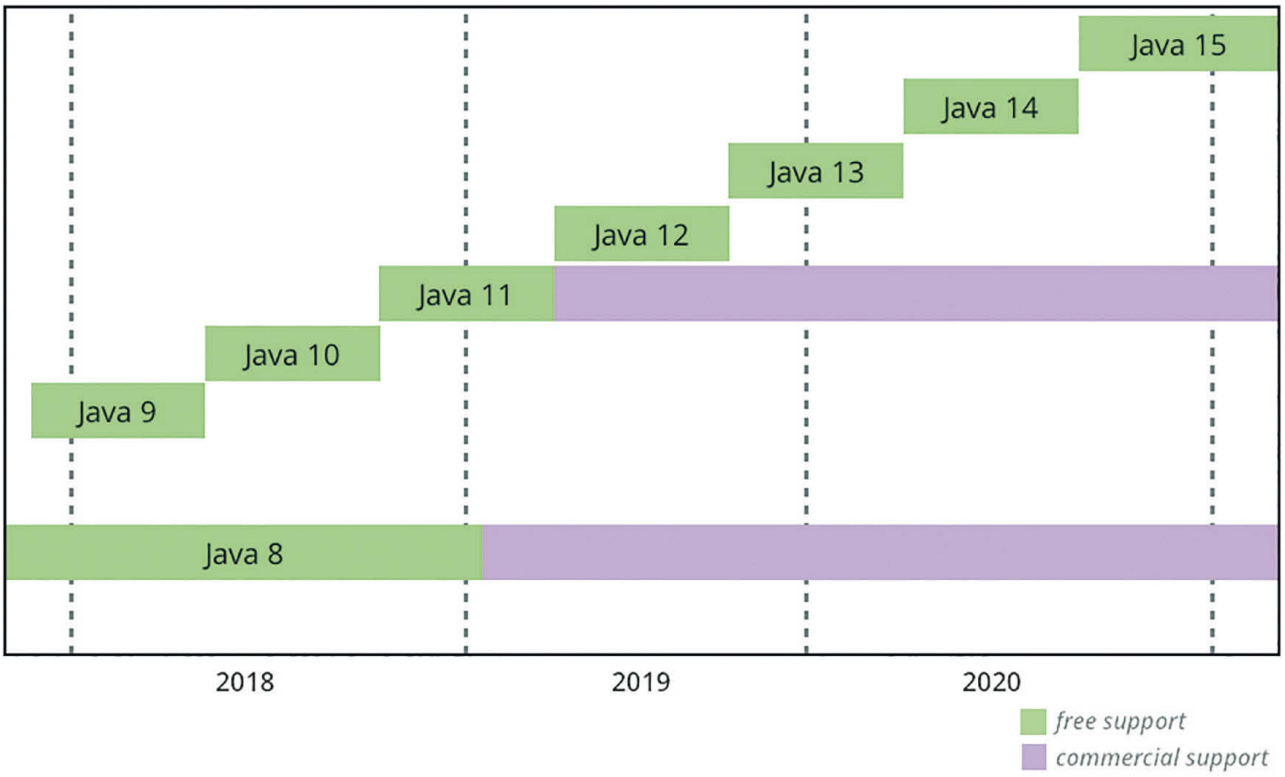
CDS est l'acronyme de Class-Data Sharing. CDS permet de prétraiter un ensemble de classes dans un fichier d'archive partagé. Pourquoi la JVM en a-t-elle donc besoin ? Durant le chargement des classes, la JVM effectue tout un tas d'opérations qui peuvent apparaître comme magiques. Elle va analyser la classe, la stocker dans une structure interne, effectuer quelques contrôles, résoudre et lier les symboles, etc. Ensuite, la classe est enfin prête à travailler. Toutes ces étapes prennent un temps certain.
En outre, chaque instance de la JVM charge généralement les mêmes classes système comme *String*, *Integer*, *URLConnection*, etc., incluses par défaut dans Java. Toutes ces classes exigent de la mémoire. Quand nous avons une archive partagée qui contient des classes prétraitées, elle peut être mappée en mémoire au moment de l'exécution. Ainsi, le temps de démarrage et l'empreinte mémoire peuvent être réduits.
Actuellement, une image JDK inclut une liste de classes par défaut dans le répertoire lib. Cette liste de classes par défaut est générée au moment de la compilation. Si nous voulons profiter de CDS avec la liste des classes par défaut fournie dans le JDK, nous devons exécuter *java -Xshare:dump* comme une étape supplémentaire.
Cette introduction sur le concept de CDS faite, nous pouvons passer au fonctionnement de Default CDS Archives. Cette fonctionnalité va modifier la construction du JDK pour exécuter *java -Xshare:dump* après avoir lié l'image. L'archive CDS résultante sera créée dans le répertoire lib/server, ce qui signifie qu'elle fera partie du JDK. Par conséquent, la fonction CDS sera activée automatiquement au démarrage d'une application (rappelez-vous que *-Xshare:auto* était activé par défaut pour la VM serveur du JDK 11). Si vous souhaitez désactiver CDS, vous devrez utiliser l'option *-Xshare:off*.

Collectes mixtes annulables pour G1

Le domaine de la Garbage Collection est clairement à l'honneur avec Java 12. Vous connaissez probablement le Garbage Collector G1 utilisé dans la JVM Hotspot. Si c'est le cas, vous devez savoir que G1 met en pause les threads d'application durant le phase "Arrêt du monde" au moment où il commence la phase de collecte. G1 peut être configuré avec un objectif de temps de pause maximum. Cela se fait en utilisant l'option *-XX:MaxGCPauseTimeMillis*. Ensuite, G1 va essayer d'atteindre cet objectif de temps de pause lors du processus de Garbage Collection via diverses techniques. Malheureusement, il arrive parfois que G1 ne respecte pas l'objectif de temps de pause. Cela peut se produire lors de collectes mixtes quand G1 essaie de collecter à la fois des régions jeunes et des régions vieilles du tas. Cette évolution de G1 vise à le rendre plus intelligent via une mise à jour lui permettant de pouvoir interrompre les collectes mixtes si elles dépassent l'objectif de pause.
En pratique, si G1 découvre que l'heuristique de sélection de l'ensemble de collecte sélectionne de façon répétée le mauvais nombre de régions, il va passer à une méthode plus incrémentale de collecte mixte : division de l'ensemble de collecte en deux parties, une partie obligatoire et une partie optionnelle. La partie obligatoire comprend les parties de l'ensemble de collecte que G1 ne peut pas traiter de manière incrémentale (par exemple, les régions jeunes) mais peut également contenir de vieilles régions pour une meilleure efficacité.
Une fois que G1 a fini de collecter la partie obligatoire, il commence à collecter la partie optionnelle à un niveau beaucoup plus granulaire à condition qu'il reste du temps. La granularité de la collecte de cette partie optionnelle dépend du temps restant, tout au plus dans une région à la fois. Après avoir terminé la collecte de n'importe quelle partie de l'ensemble de collecte optionnel, G1 peut décider d'arrêter la collecte en fonction du temps restant.
Au fur et à mesure que les prédictions redeviennent plus précises, la partie optionnelle d'une collecte devient de plus en plus petite, jusqu'à ce que la partie obligatoire comprenne à nouveau l'ensemble de la collecte. Si les prédictions se révèlent à nouveau inexactes, les prochaines collectes se composeront à nouveau d'une partie obligatoire et d'une partie optionnelle.

Renvoi rapide de la mémoire inutilisée de G1

Pour terminer ce tour d'horizon des nouveautés de Java 12, nous nous attardons encore une fois sur une nouveauté liée au Garbage Collector G1. Actuellement, G1 ne renvoie la mémoire du tas Java que lorsqu'un cycle de Garbage Collection complet se produit ou



pendant un cycle simultané. Mais les deux évènements peuvent ne pas se produire très souvent avec G1. Par conséquent, il se peut que la mémoire ne soit pas retournée au système d'exploitation pendant une longue période.

Java 12 met à jour le Garbage Collector G1 pour détecter l'inactivité de l'application et retourner la mémoire du tas Java au système d'exploitation lorsqu'elle est inactive. Pour atteindre l'objectif de retourner un maximum de mémoire au système d'exploitation, G1 essaiera, pendant l'inactivité de l'application, de continuer ou de déclencher périodiquement un cycle simultané pour déterminer l'utilisation globale du tas de Java. Cela lui permettra de renvoyer automatiquement au système d'exploitation les parties inutilisées du tas de Java.

Afin de justifier cette mise à jour, les architectes du JDK précisent que le fait que G1 ne renvoyait pas assez souvent la mémoire était particulièrement désavantageux dans les environnements de conteneurs où les ressources sont payées à l'usage. Même pendant les phases où la JVM n'utilise qu'une fraction des ressources mémoires qui lui sont affectées en raison de son inactivité, G1 conservait tout le tas de Java. Il en résultait que les clients payaient toutes les ressources en permanence et que les fournisseurs de cloud computing n'étaient pas en mesure d'utiliser pleinement leur matériel alors que certaines de leurs ressources auraient pu être employées ailleurs.

Cette option de G1 s'avère donc essentielle puisqu'elle permettra aux sociétés exécutant des applications Java dans des conteneurs de réduire leurs dépenses liées à l'usage qu'elles font des ressources matérielles.

Un mot sur Java 13

Java 12 tout juste sorti, il est déjà de temps de s'intéresser au contenu de Java 13 dont la sortie est attendue dans 6 mois conformément au nouveau cycle de release d'Oracle pour le JDK. Pour le moment, seules deux fonctionnalités ont été proposées officiellement pour Java 13. Là encore, il s'agit de nouveautés autour de la JVM n'impactant pas directement le travail quotidien des développeurs. Il s'agit de Java Compiler Intrinsics for JDK APIs qui vise à permettre au compilateur Java d'utiliser des stratégies de traduction alternatives, telles que *invokedynamic*, dans le but d'améliorer les performances de certaines méthodes du JDK.

La seconde a pour but de rendre accessible en continu les données de l'outil JDK Flight Recorder afin de fournir une API pour la consommation continue des données de JFR sur disque. Espérons que des fonctionnalités plus orientées développeurs Java soient ajoutées. On parle ainsi d'une introduction des littéraux de chaînes de caractères qui devaient faire partie de Java 12 mais qui ont finalement été retirés au tout dernier moment. Pour le reste, il faudra encore attendre quelques semaines pour en savoir plus.

Conclusion

La plupart des nouvelles fonctionnalités de Java 12 sont plus ou moins liées à la JVM et son fonctionnement interne. Elles n'impacteront pas les développeurs Java dans leur travail au quotidien mais elles sont toujours nécessaires pour poursuivre le travail d'amélioration constant de la plateforme Java. Les switchs étendus constituent la seule vraie nouveauté orientée développeurs de Java 12 mais malheureusement elle n'est qu'en preview. Gageons qu'elle soit intégrée pleinement à Java 13 et que le pattern matching s'appuyant sur les switchs étendus soit également de la partie. Rendez-vous en septembre 2019 pour avoir la réponse. •

Mathieu Eveillard
Arolla

CycleJS : un framework et une architecture

Beaucoup a été fait ces dernières années pour faciliter l'écriture de Single Page Applications (SPA). Diverses librairies et frameworks ont apporté de la structure et du systématisme dans l'écriture de composants et la gestion de l'état applicatif. Cependant, l'utilisation d'outils tels que React et Redux sur des applications de grande ampleur, de l'ordre de 100 000 lignes de code, révèle la difficulté à structurer le code applicatif en modules fonctionnels, en charge d'un périmètre de données. CycleJS apporte des réponses sur ce point précis, en plus d'autres qualités que nous nous attacherons à détailler dans cet article.

niveau 200

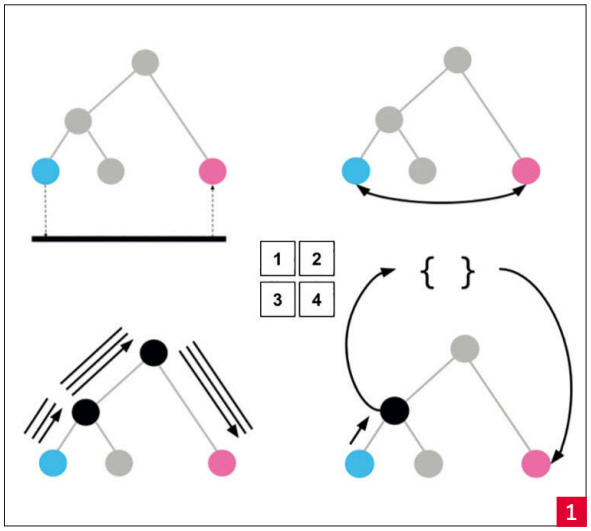
Préambule

Les considérations à venir s'entendent dans le cadre d'une architecture AJAX (Asynchronous JavaScript and XML), caractérisée en premier lieu par une inversion de contrôle : contrairement aux applications Web "classiques", que l'on désignera ainsi faute de mieux, c'est le code JavaScript exécuté par le navigateur Web qui contrôle l'application, initiant de lui-même les requêtes à destination du serveur. Une fois passé le chargement initial, cette architecture ne requiert plus que l'échange de données unitaires, intégrées dans l'application grâce au code JavaScript. Il en résulte une plus grande fluidité et une plus grande interactivité avec l'utilisateur.

La communication entre composants : le défi de toutes les SPAs

Il est courant qu'un composant A permette la saisie d'une donnée et qu'un composant B doive afficher cette même donnée.

En pareille situation, les deux composants vont devoir communiquer. Au moins quatre modalités d'échange ont été inventées à cette fin. 1



1

La communication par bus de données [1] était le mode privilégié par Backbone.js. Le composant A publie un évènement, c'est-à-dire un objet contenant la donnée d'intérêt. Tout composant à l'écoute de ce canal et de cette nature précise d'évènement est ainsi notifié et accède à la donnée. C'est un patron "observer". Des bus distincts peuvent être mis en place quand il est nécessaire de restreindre l'accès à certaines données.

Le framework Angular reprend cette idée, mais pour en faire une communication de point à point, de composant à composant, au travers de services [2]. Angular permet d'autres modes de communication, notamment entre composants parents et enfants, mais cette communication par service est sans doute ce qui le distingue des autres librairies et framework front-end. L'écueil est que, l'application grandissant, on arrive rapidement à un enchevêtrement de fils, autrement appelé "anti-pattern spaghetti". Dans le pire des cas, le manque de systématisme peut conduire à une désynchronisation des composants, qui travaillent sur des versions distinctes d'une même donnée.

Avec React [3], deux composants communiquent grâce à leur ancêtre commun, c'est-à-dire le premier composant qui les relie. Pour faire "remonter" une donnée, un composant enfant invoque une fonction de rappel ("callback props") passée en paramètre par son parent. Pour faire "redescendre" une donnée, le composant parent la passe en paramètre ("props") au composant enfant en question. Bien que systématique, ce mode de communication se révèle extrêmement fastidieux à écrire. En premier lieu parce que le transfert de données doit être mis en œuvre explicitement au niveau de chaque composant relayant la donnée, mais aussi parce que ce "câblage" doit être réalisé autant de fois qu'il y a de données à faire transiter.

Ce constat, qui allait de pair avec un risque d'erreur important, a conduit la communauté JavaScript à rechercher des solutions alternatives, et c'est ainsi qu'est né Redux [4]. Cette architecture repose sur un objet unique représentant l'état applicatif partagé par tous les composants, c'est-à-dire un sous-ensemble des données devant être échangées (un sous-ensemble seulement parce que la communication directe entre composants parents et enfants reste possible, comme illustré par le schéma). La mise à jour de cet objet se fait par l'intermédiaire d'actions (event sourcing) et de fonctions

pures, les "reducers". La contrainte introduite par ces stéréotypes est la garantie d'un plus grand systématisme, en même temps qu'une meilleure traçabilité, qui facilite le travail du développeur.

Parler de "gestion de l'état applicatif" est problématique

En centralisant ces données, Redux a introduit la notion de gestion de l'état applicatif ("state management"), qui n'existait pas auparavant. Cette architecture, censée aller dans le sens de la simplification, laisse pourtant le développeur face à de nombreuses questions : tout composant doit-il avoir accès à cet objet ? Comment structurer cet objet ? Sur ce sujet, l'expérience prouve qu'il est préférable de le normaliser, comme une base de données(1).

Mais, à bien y regarder, la centralisation de l'état applicatif introduit un problème plus profond, car elle revient à déresponsabiliser les composants. Ces derniers ne sont finalement plus en charge que de l'interaction bas-niveau avec l'utilisateur et n'implémentent que très peu de comportements. Nous sommes bien loin de composants autonomes, responsables du cycle de vie d'une donnée, capables d'interroger eux-mêmes le serveur pour récupérer la donnée ou la mettre à jour suite à l'interaction avec l'utilisateur. Une vraie modularité se traduirait par des composants jouant le rôle de source de vérité pour une donnée.

Ces fonctionnalités, que les composants ne prennent plus en charge, doivent bien être mises en œuvre quelque part dans l'application.

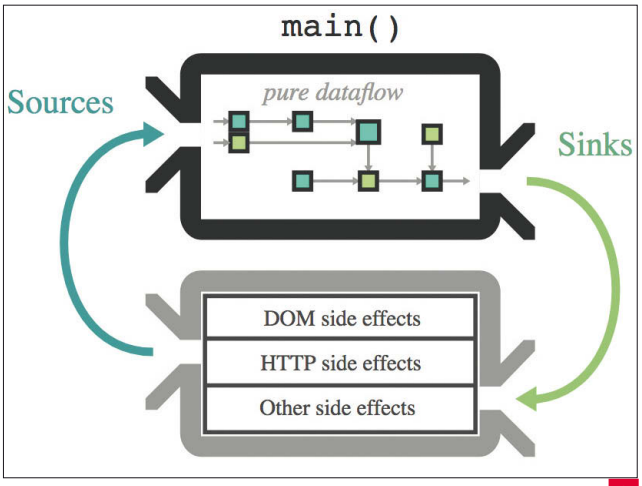
De fait, l'écosystème React / Redux a vu fleurir bon nombre de librairies (redux-thunk, redux-observable, redux-saga pour n'en citer que quelques-unes) censées permettre la gestion de ces "effets de bord", et plus largement d'orchestrer l'application. Autrement dit, les composants ont perdu le contrôle de l'application.

Ainsi, l'écosystème React / Redux a crû en complexité, multipliant les stéréotypes applicatifs. La plus petite application nécessite l'écriture de composants, d'actions, de reducers, de sélecteurs et, pour peu qu'elle gagne en ampleur, de "thunks", "epics" ou autres stéréotypes exotiques censés faciliter les choses. Partant, la testabilité pose problème : autant chaque stéréotype est testable unitairement, autant il est difficile de tester leur intégration sans passer par des tests end-to-end, dont le temps d'exécution est un facteur limitant. Mais, peut-être plus problématique encore, la notion de module fonctionnel a disparu : regrouper l'ensemble des stéréotypes agissant sur une même donnée pour en faire un module est une mission quasi impossible à relever.

CycleJS

La caractéristique première de ce framework (https://cycle.js.org) est qu'il permet l'écriture de composants autonomes, capables de faire tout ce qui est nécessaire pour gérer une donnée : effectuer des requêtes HTTP en lecture ou en écriture, lire ou écrire dans le DOM et communiquer avec son composant parent ou ses composants enfants. Ainsi, chaque composant pourrait être monté en tant qu'application à part entière, d'où l'aspect fractal.

Ce framework se veut par ailleurs réactif et repose sur des observables. CycleJS supporte RxJS, Most et xstream, cette dernière



2

librairie ayant été conçue spécifiquement pour les besoins du framework. Plus simple que RxJS, xstream implémente les principaux constructeurs d'observables (of, merge, combine) ainsi que les principaux opérateurs (map, filter, fold) et ne propose que des observables "chauds". Une littérature abondante existe(2), qui permettra au lecteur d'approfondir cette notion, mais au premier abord l'analogie suivante pourra aider : une séance de cinéma serait un observable chaud parce que la séance commencera, que je sois ou non dans la salle, tandis que la lecture d'un Blu-ray est comparable à un observable froid parce qu'elle ne commencera qu'au moment où moi, spectateur, déciderai de l'insérer dans le lecteur. 2

Mais revenons à CycleJS, pour préciser qu'un composant est une fonction qui prend en arguments des "sources" et renvoie des "sinks". Sources et sinks sont des objets qui permettent l'interaction du composant avec le monde extérieur. Les sources en permettent la lecture. Ainsi, la source du DOM permet d'accéder à des observables tels que le flux des valeurs saisies par l'utilisateur dans un champ ou le flux des clics sur un bouton. Les sinks permettent quant à eux l'action du composant sur le monde extérieur. Par exemple, le composant renvoie un flux de nœuds décrivant le DOM souhaité et le driver le met à jour (mécanisme de DOM virtuel avec algorithme de différenciation). Tout composant est "réactif" en ce sens qu'il renvoie des flux pour agir sur le monde extérieur en réaction aux flux qui lui sont passés en entrée.

Dernière caractéristique, majeure : les composants sont des fonctions pures. Cela signifie premièrement que deux appels successifs à un composant avec les mêmes sources fourniront les mêmes sinks (idempotence). Cela signifie d'autre part que l'appel à un composant se fait sans incidence sur le monde extérieur (absence d'effet de bord). Si des actions sur ce dernier sont souhaitées, cela se traduira dans les sinks renvoyés par le composant et interprétés par les drivers du framework, seuls autorisés à mettre en œuvre ces effets de bord.

Une conséquence de l'utilisation de fonctions pures est l'absence d'état à l'intérieur des composants. Ceux-ci se contentent d'effectuer des transformations sur des flux de données, comme nous

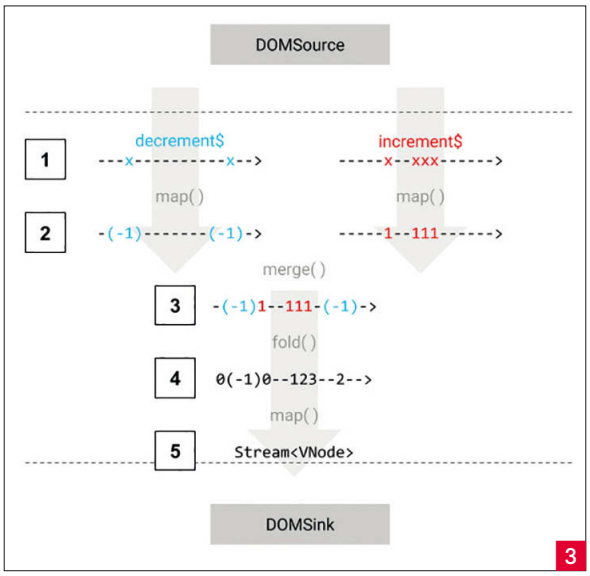
(1) https://bit.ly/2ApdqxM

(2) https://bit.ly/2ruN3Td

allons le voir. Toutefois, il n'est pas exclu que les flux communiqués au composant au travers des sources ou renvoyés par ce dernier possèdent un état (flux à mémoire). Ce cas de figure ne change pas ce qui a été dit précédemment, car l'état est celui des flux manipulés par le composant, non celui du composant lui-même. Il est possible d'effectuer des transformations pures sur des flux à états. La distinction est importante.

Illustrons ce qui précède à l'aide de l'exemple d'un compteur actionnable au travers de deux boutons, l'un pour le décrémenter, l'autre pour l'incrémenter. Le composant implémentant ce compteur requiert une source du DOM, qu'il interrogera pour obtenir les observables correspondant aux clics de l'utilisateur sur chacun des boutons [1]. Une première étape consiste à associer un clic à une valeur, -1 ou 1 selon le bouton [2], avant de fusionner ces deux flux [3]. L'étape suivante consiste à créer un flux dont chaque évènement est le cumul des valeurs passées [4]. Ce flux est donc un flux à mémoire. La dernière étape est la création d'un flux de nœuds du DOM virtuel pour les valeurs successives du compteur [5]. 3

Le schéma précédent se traduit comme suit en TypeScript :



3

```
function main({ DOM }: Sources): Sinks {
const decrement$: Stream<any> = DOM.select('.decrement').events('click');
const increment$: Stream<any> = DOM.select('.increment').events('click');
const action$: Stream<number> = Stream.merge(decrement$.mapTo(-1),
increment$.mapTo(+1));
const count$: Stream<number> = action$.fold((x: number, y: number) => x + y, 0);
const vdom$ = count$.map(count =>
  div([
    button('.decrement', 'Decrement'),
    button('.increment', 'Increment'),
    p('Counter: ' + count)
  ])
);
return { DOM: vdom$ };
}
```

Interaction parent / enfant

Détaillons à présent le mécanisme par lequel deux composants échangent de l'information. Comme dans React, la communication se fait au travers du premier ancêtre commun. Cette mécanique met en avant la responsabilité du composant : s'il se contente de faire suivre l'information, qui monte ou descend le long de l'arbre des composants, c'est qu'il n'en est pas responsable. Si, en revanche, le composant stoppe la remontée de l'information et la fait redescendre à destination d'un autre composant, c'est qu'il en est responsable.

Cependant, là où React impose de déclarer des "props" et des "callback props" pour chaque donnée à faire transiter, CycleJS propose un mécanisme qui facilite les échanges entre composants parent et enfant. 4

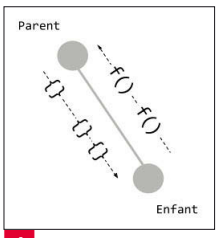
En effet, le composant parent communique au composant enfant via les sources un flux dont chaque évènement correspond à un état d'entrée ("state"). Cette notion renvoie aux données d'entrée du composant enfant et non à l'état interne de ce dernier, puisque nous avons vu que les composants n'ont pas d'état. S'il fallait faire un parallèle avec React, cette notion serait à rapprocher des "props". En sens inverse, le composant enfant renvoie au travers de ses sinks un flux de fonctions de réduction ("reducers"), construit notamment en réaction aux actions de l'utilisateur et dont la signature en TypeScript serait la suivante :

```
(state: ChildState) => ChildState
```

Un reducer est donc une fonction qui, appliquée à l'état d'entrée du composant enfant, renvoie un état modifié. C'est au travers de ce flux de fonctions que le composant enfant signifie comment il souhaite que les données qu'il manipule soient modifiées. À ce stade, il est important de noter que le composant enfant ne modifie pas lui-même les données, mais qu'il donne à son parent les éléments pour ce faire : c'est le parent qui invoque la fonction de réduction. Ceci offre la garantie d'un meilleur contrôle sur la mise à jour des données.

Il convient également de remarquer que, contrairement à Redux, la fonction de réduction de CycleJS ne fait pas intervenir d'action. Cela tient au fait qu'une fonction de réduction est une action, au sens où c'est une fonction spécialisée pour mettre à jour un état avec des données qui, dans Redux, proviendraient d'une action. Ces données ont été capturées dans le contexte de la fonction par closure. Exemple :

```
function makeUpdateNameReducer(updatedName$: Stream<string>):
Stream<Reducer<State>> {
return updatedName$.map(
  (name: string) =>
   function (state: State): State {
     return {
       ...state,
       name
     };
   }
);
}
```



4

Rentrons encore un peu plus dans le détail de cette interaction, sur base de l'exemple graphique suivant : 5

En [1], le composant parent reçoit un flux d'états parents que nous nommerons parentState$ (Stream<ParentState>). La convention veut qu'un flux soit mis en évidence par un $ terminal. L'état parent est un tableau de ronds rouges. Notons que, pour la lisibilité, le schéma représente chaque flux par un de ses évènements.
Une des responsabilités du composant parent est de construire le flux d'états qui sera passé au composant enfant au travers de ses sources [2]. Ce childState$ (Stream<ChildState>) est un rond rouge, le premier issu de l'état du composant parent. Il est communiqué au composant enfant, et l'on pourrait imaginer qu'il en soit de même pour les deux autres éléments du tableau. Le composant parent et les trois composants enfants implémentent donc collectivement une liste d'objets.
Le composant enfant renvoie une fonction de réduction qui, d'un rond rouge donné en entrée, fait un carré rouge [3]. Sa signature est la suivante :

```
(state: ChildState) => ChildState
```

Cette fonction de réduction enfant est utilisée par le composant parent pour construire sa propre fonction de réduction qui, pour chaque rond rouge, applique la fonction de réduction du composant enfant à qui elle en a donné la charge [4]. La fonction de réduction parent aura donc pour signature :

```
(parentState: ParentState) => ParentState
```

La construction de la fonction de réduction parent à partir des fonctions de réduction enfants est complexe à appréhender. Heureusement, le framework en propose une implémentation pour le cas particulier, mais fréquent de collections d'objets.

Testabilité

Le fait qu'un composant soit une fonction pure est un avantage certain pour l'écriture de tests, en TDD préférentiellement. Un composant implémente un comportement dans son ensemble, faisant intervenir différentes sources : le DOM, les requêtes HTTP et le router pour n'en citer que quelques-unes. Tester un composant peut vouloir dire "tester que le clic de l'utilisateur à tel endroit du DOM provoque bien l'émission d'une requête HTTP", ou bien "tester que le retour de la requête HTTP avec telle information dans le payload met à jour le DOM de telle façon".
Bien sûr, toutes les sources peuvent faire l'objet de bouchons, jusqu'à virtualiser le temps. L'exemple ci-dessous montre le test d'un composant ProductListItem qui, en réaction au clic de l'utilisateur, est censé renvoyer une instruction de mise à jour de l'URL par le router :

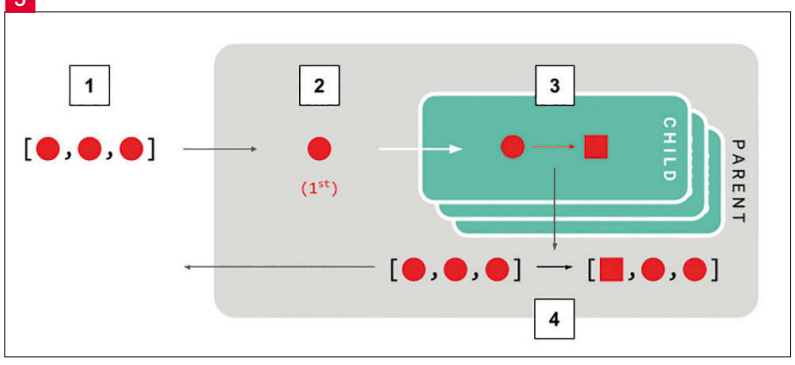
```
it('should update route when the user clicks on the product list item', done
=> {
  // GIVEN
  const time: MockTimeSource = mockTimeSource();
  const navigate$ = time.diagram('--x---------');
  const DOM: DOMSource = mockDOMSource({
    '[data-action="navigate"]': { click: navigate$ }  });
  const product: Product = {
    id: 'aef3c8a3-65ef-4242-8ac7-0d329ba6931e'
  };

  // WHEN
  const sinks: Sinks = ProductListItem({ DOM, state: { stream:
xs.of(product) } } as Sources);
  const actual$: Stream<HistoryAction> = sinks.router;

  // THEN
  const expectedValue: HistoryAction = '/product/aef3c8a3-65ef-4242-8ac7-0d329
ba6931e';
  const expected$: Stream<HistoryAction> = time.diagram('--e---------', {
    e: expectedValue
  });
  time.assertEqual(actual$, expected$);
  time.run(done);
});
```

Ces tests n'excluent nullement l'écriture de tests unitaires sur les fonctions non exposées qui implémentent la logique du composant, ni bien sûr l'écriture de tests end-to-end visant à tester l'intégration entre les composants ainsi que le bon fonctionnement de l'application compte tenu de son habillage graphique (CSS). Mais, déjà, ces tests ont du sens, fonctionnellement parlant.

Conclusion

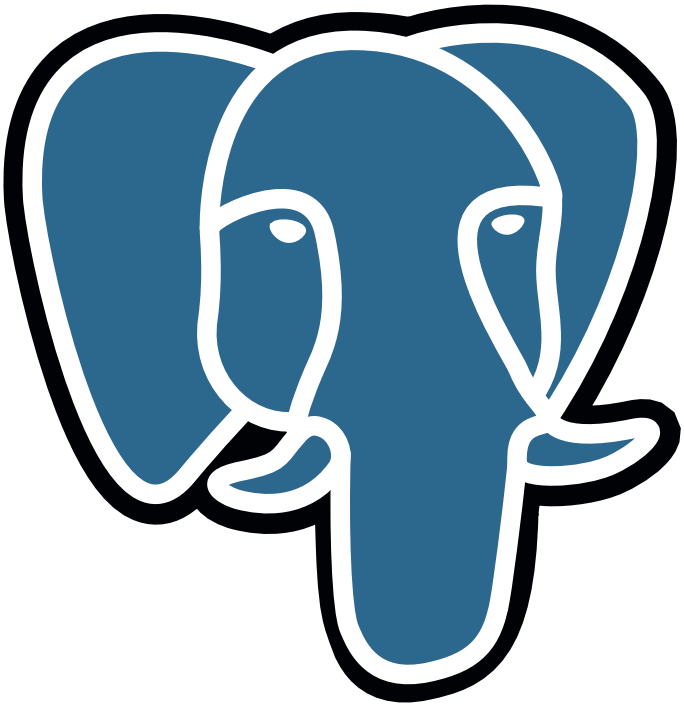
Le framework CycleJS permet donc l'écriture de Single Page Applications selon un paradigme réactif et fonctionnel. De ce fait, le code produit est hautement testable. Chaque composant est responsable d'une donnée applicative et peut être considéré comme une application à part entière, entièrement autonome, d'où l'aspect fractal précédemment évoqué.
Actuellement en version 7, CycleJS existe depuis 2015 et rassemble une communauté de près de 10 000 utilisateurs. •

5



Laetitia Avrot
Experte PostgreSQL. Lætitia est co-fondatrice du mouvement Postgres Women et membre du comité du code de conduite de la communauté PostgreSQL. Elle a écrit plusieurs patches. Durant l'automne 2018, elle a été mentor pour le Google Code-In Contest pour la communauté PostgreSQL. @l_avrot

PostgreSQL : qu'est-ce que c'est ?



PostgreSQL est un Système de Gestion de Bases de Données Relationnel-Objet (SGBDRO). Son travail consiste à gérer les accès concurrents aux données. C'est-à-dire de gérer les transactions (Atomicité et isolation), la Consistance de ces données et d'assurer leur Durabilité dans un contexte où n clients peuvent modifier ces données en même temps. Il s'agit des fameux principes ACID communs aux SGBDR.

niveau 100

Un peu d'histoire

Le projet a été initié par Michael Stonebraker dans les laboratoires de recherche de l'université de Berkeley en Californie. C'est là qu'il devient responsable d'un premier système de gestion de bases de données, INGRES (pour INteractive Graphics Retrieval System). Cependant, les limites du modèle relationnel commencent à se faire sentir et Michael Stonebraker démarre un nouveau projet : PostGres (pour Post-Ingres).

Les laboratoires de l'université de Berkeley sont particuliers, car au lieu d'être formés en silo sur une matière, ils sont interdisciplinaires, ce qui implique forcément beaucoup de coopération. De plus, ils sont créés pour une durée limitée, en fonction des projets, ce qui permet de s'entourer des talents nécessaires pour proposer des applications qui correspondent vraiment à un besoin. Généralement les logiciels issus de Berkeley sont sous licence BSD. Le projet PostgreSQL a sa propre licence qui se résume simplement à "Prenez le code, faites-en ce que vous voulez, mais ne nous faites pas de procès."

Entre 1987 et 1994, le moteur de Postgres est entièrement (ré)écrit 3 fois et en 1994, c'est la fin du projet de recherche, mais ce n'est pas la fin de Postgres. En 1995, deux étudiants de Michael Stonebraker, Andrew Yu et Jolly Chen, décident d'implémenter le SQL dans Postgres et de créer un projet Open Source. Le projet prend le nom de PostgreSQL, nom difficile à prononcer (que ce soit en français ou en anglais) et qui reste à ce jour la seule erreur du projet (selon Tom Lane, contributeur majeur de PostgreSQL).

Voici donc comment le prononcer : cela s'écrit PostgreSQL, mais le Q et le L sont muets. On prononce donc Postgress ([pɔsgʁɛs]). L'abréviation PG est également souvent utilisée dans la communauté.

La philosophie du projet

PostgreSQL s'est construit sur une note de spécification écrite par Michael Stonebraker (http://db.cs.berkeley.edu/papers/ERL-M85-95.pdf). Le deuxième point de cette note indique que le nouveau moteur devra être extensible, c'est-à-dire que l'utilisateur pourra définir ses propres types de données, ses propres opérateurs et ses propres fonctions. Depuis, il est possible de packager ses modifications dans une "extension". Ces extensions peuvent ajouter de très nombreuses fonctionnalités à PostgreSQL. Cela peut aller de la simple extension qui fournit la recette des pâtes à la carbonara (la seule vraie recette d'après son auteur italien) à une extension permettant du sharding horizontal pour gérer des Po de données.

Une autre valeur très importante du projet PostgreSQL, c'est la stabilité. Tout dans le projet est mis en œuvre pour favoriser la stabilité. Si un patch casse la rétrocompatibilité, son impact sera longuement jugé avant d'être accepté.

De plus, pour éviter la tentation (humaine) de ne pas corriger les bugs, aucune solution de contournement ne sera jamais proposée officiellement par le projet. Ainsi, le projet n'a pas de bug tracker car un bug chez PostgreSQL n'a pas une durée de vie très longue (98 % des bugs sont corrigés dans la journée). Pour finir, une version majeure de PostgreSQL ne pourra pas voir le jour si un bug est connu et n'a pas été corrigé.

C'est cette rigueur qui permet à PostgreSQL d'afficher aujourd'hui 99,999 % de disponibilité, même si, au début du projet, il a été plus long d'ajouter de nouvelles fonctionnalités.

Un Système de Gestion de Bases de Données Relationnelles, mais pas que

Au départ, PostgreSQL a été classé dans les Systèmes de Gestion de Bases de Données Relationnelles. En effet, à l'époque (vers 1996), le SQL était fortement lié à la notion de modèle relationnel. Ce n'est qu'à partir de SQL:1999 que la norme SQL a commencé à évoluer au-delà de la notion de modèle relationnel. C'était une énorme avancée, mais aussi un énorme risque : en permettant l'utilisation de types de données riches (non atomiques) comme les

tableaux, les tables ou les types composites, on permettait à la fois d'améliorer la modélisation et de la rendre pire, en fonction des compétences de l'équipe qui modélise.
La modélisation des données est un exercice difficile qui demande beaucoup d'expérience, de souplesse et de remise en question de principes qu'on croyait inébranlables. Sachez seulement que si cette étape est faite correctement, vous ne devriez pas rencontrer de difficulté pour écrire le code ensuite. Tout vient des données. Si elles sont bien modélisées, elles seront de bonne qualité et tout le reste sera facile.
Dans les faits, PostgreSQL est plus vu aujourd'hui comme un Système de Gestion de Bases de Données Relationnel-Objet, car il intègre l'héritage de tables et la surcharge des opérateurs en fonction des opérandes.

Une communauté ouverte et bienveillante

PostgreSQL, c'est aussi une communauté très importante. Pour éviter les soucis qu'ont pu rencontrer d'autres communautés Open Source (rachat, mainmise d'un groupe...), PostgreSQL s'est structuré autour d'un groupe de 5 à 7 personnes qui gèrent l'avenir de la communauté, la Core Team. Pour éviter le conflit d'intérêts, ces personnes travaillent (dans la mesure du possible) pour des sociétés différentes.
La communauté PostgreSQL accueille chaque personne qui souhaite apprendre ou donner un peu de son temps. Il n'y a pas de petite contribution. C'est la seule communauté que je connaisse où les gens sont capables d'écouter les arguments les uns des autres et de changer d'avis en fonction de ces arguments. C'est également la seule communauté où j'ai vu des personnes commiter des fonctionnalités qui ne leur plaisaient pas, mais pour lesquelles la majorité avait voté.
Je ne dis pas que c'est le monde des licornes et des arcs-en-ciel, j'essaye juste d'expliquer que la grande majorité des contributeurs est extrêmement bienveillante et essaiera d'expliquer son opinion tout en essayant de comprendre le point de vue de la personne en face.
Si vous avez besoin d'aide, demandez et quelqu'un trouvera le temps de vous aider. C'est comme cela que fonctionne l'aide sur le canal IRC #postgresql de freenode, sur le slack de la communauté et sur les différentes mailing-listes.

EN PRATIQUE

Après cette petite présentation théorique, voyons concrètement comment faire fonctionner PostgreSQL.

Installation et démarrage

Comme n'importe quel outil open source, vous pouvez bien sûr télécharger le code, le compiler et faire tourner votre programme. Cependant, je ne vous le conseille pas. En effet, pour compiler vous-même PostgreSQL, vous allez devoir installer par vous-même toutes les dépendances du projet à la main, ce qui est très pénible. De plus, si vous travaillez dans un environnement professionnel, sachez que l'automatisation de l'installation et la maintenance d'une instance ainsi installée seront beaucoup plus difficiles à gérer.
Par la suite, mes démonstrations se feront sur un Ubuntu Server (Bionic Beaver). Il est possible d'installer PostgreSQL sous Windows, mais je le déconseille largement (en production) pour deux raisons :

- La première, c'est que la version Windows de PostgreSQL n'est qu'un portage du code (même si ce portage est de qualité) et, tout comme je préfère lire une documentation dans sa version originale, je préfère faire tourner un code sur l'OS pour lequel il a été conçu ;
- La deuxième, c'est que la grosse majorité des serveurs PostgreSQL dans le monde tournent sous Linux. Il y a donc moins de risque de rencontrer un bug sur une version tournant sous Linux.

Installation du serveur

Pour installer PostgreSQL, il faut commencer par aller sur la page download de postgresql.org. Vous y trouverez les instructions pour chaque OS.
Ubuntu Server étant Debian-based, il utilise Advanced Packaging Tool, ou apt, si vous préférez. PostgreSQL se trouve dans les dépôts par défaut, mais vous risquez de ne pas avoir accès aux dernières versions. Pour cette raison, nous vous conseillons de toujours utiliser les dépôts du projet PostgreSQL.
Une fois sur la page download de postgresql.org, je clique sur le lien Ubuntu. Là, une liste déroulante me permet de choisir ma version (Bionic Beaver) et les instructions qui suivent sont adaptées pour ma version.
La première chose à faire, c'est d'ajouter les dépôts Postgres à apt. Pour ce faire, il suffit de créer un fichier /etc/apt/sources.list.d/pgdg.list et d'y ajouter cette ligne :

```
deb http://apt.postgresql.org/pub/repos/apt/ bionic-pgdg main
```

Puis, il faut ajouter la clé du dépôt à apt avec cette commande :

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
```

Enfin, on peut mettre à jour la liste des packages connus par apt :

```
sudo apt-get update
```

Il existe deux packages importants pour installer PostgreSQL :

- postgresql-client-xx (avec xx le numéro de version)
- postgresql-xx (avec xx le numéro de version)

Le package client n'installe que les outils nécessaires pour établir des sessions clientes à une base de données. La liste des outils clients de PostgreSQL est disponible dans la documentation.
Si vous souhaitez créer votre instance PostgreSQL, vous aurez besoin du package postgresql-xx.

```
sudo apt-get install postgresql-11 postgresql-client-11
```

Aparté sur les versions de PostgreSQL

PostgreSQL avait pris l'habitude de numéroter les versions majeures sur deux nombres (exemple : 9.6). Cela posait des problèmes à la communauté, car, les décisions étant prises en consensus, de longues discussions devaient déterminer si les changements de la nouvelle version étaient suffisants pour nécessiter un incrément du premier nombre ou du deuxième.

Pour éviter ces discussions finalement peu constructrices et très consommatrices en temps, la communauté a décidé de numéroter les versions majeures sur un seul nombre à partir de la version 10.
Si certains d'entre vous ont travaillé sous Oracle, vous connaissez l'adage "Il ne faut jamais installer une R1 en production".
Je vais vous demander de ne pas faire d'amalgame et de garder cet adage pour le monde Oracle. PostgreSQL est bâti sur un code solide comme un roc grâce à une rigueur dans le process de validation du code assez exceptionnelle. Lorsqu'une version sort en production (généralement en octobre), elle a déjà été testée sur des milliers de bases de données dans le monde, grâce à la communauté. Lorsque cette version sort, elle est prête pour la production et vous pouvez l'installer sans danger. (Attention, je ne dis pas qu'il ne faut pas tester un minimum une nouvelle version avant de l'installer en production.)
PostgreSQL ne publie pas de planning précis de sortie des versions pour se réserver le droit de retarder une version s'il existe le moindre doute quant à la stabilité de celle-ci.
De plus, le projet apporte une attention toute particulière à la rétrocompatibilité. Si la nouvelle version inclut un risque pour l'upgrade, cela sera écrit explicitement dans la note de version (qu'il est conseillé de lire avant chaque upgrade, quoi qu'il arrive).

Création du cluster

Sous PostgreSQL, une instance peut permettre d'accéder aux données de plusieurs bases de données (contrairement au monde Oracle, où, jusqu'à récemment, une instance était monobase, ce qui a entraîné beaucoup de confusions entre les termes "instance" et "base de données"). Reprenons donc le vocabulaire pour être sûrs de parler de la même chose.
Une instance est un ensemble de processus et de mémoire. On peut arrêter ou démarrer une instance. Cependant, une instance ne stocke pas les données sur disque.
Une base de données est un ensemble de fichiers (souvent binaires) qui permettent de stocker de manière pérenne des données. Arrêter ou démarrer une base de données n'a pas de sens : on ne peut pas démarrer un fichier.
Un cluster (dans le monde Postgres) est une zone de stockage sur disque dans laquelle on va retrouver les bases de données ainsi que des informations techniques nécessaires soit à PostgreSQL, soit à l'instance (gestion des transactions, crash recovery...). Le nom est issu de la norme SQL qui définit un "cluster de bases de données".
Avant de pouvoir démarrer une instance PostgreSQL, il faut donc créer un cluster. Par défaut, sur les distributions Debian-based, un cluster nommé "main" est créé. Vous pouvez bien sûr l'utiliser, mais vous pouvez aussi le supprimer pour lui donner un nom plus en accord avec votre politique d'entreprise, par exemple.
Grâce aux travaux communs entre la communauté PostgreSQL et la communauté Debian, des utilitaires ont été créés. Ils facilitent l'administration des instances sous Debian. Pour que ma démonstration soit utilisable sur d'autres distributions, je ne vais pas les utiliser. (Attention, cela ne signifie pas que vous ne devez pas les utiliser chez vous, juste que c'est moins universel.)

```
# Arrêt de l'instance avec Système D
systemctl stop postgresql
# Suppression du cluster
sudo rm -rf /var/lib/postgresql/11/main
# Création d'un cluster sous /var/lib/postgresql/rainbow
sudo -u postgres /usr/lib/postgresql/11/bin/initdb \
 --pgdata /var/lib/postgresql/rainbow \
 --data-checksum
```

L'option --data-checksum permet d'activer la mise en place de checksum sur chaque page pour pouvoir détecter une éventuelle corruption.

Démarrage du serveur

Il y a plusieurs manières de démarrer une instance PostgreSQL. La première consiste à utiliser les scripts fournis par le projet PostgreSQL :

```
# Démarrage d'une instance avec pg_ctl
# Le cluster doit être initialisé dans /var/lib/postgresql/rainbow
sudo -u postgres /usr/lib/postgresql/11/bin/pg_ctl \
 -D /var/lib/postgresql/rainbow start
```

La deuxième consiste à utiliser Système D (valable sur toutes les distributions Linux qui utilisent Système D) :

```
# Démarrage d'une instance avec Système D
# Pour le service postgresql défini dans
#  /etc/systemd/system/postgresql.service sous Debian
#  ou /usr/lib/systemd/system/postgresql.service sous Red Hat
sudo systemctl start postgresql
```

La troisième consiste à utiliser les scripts Debian (valable sur toutes les distributions Debian-based) :

```
# Démarrage d'une instance nommée rainbow avec les scripts Debian
#  pour une instance rainbow définie dans
#  /etc/postgresql/11/rainbow/pg_ctl.conf
sudo -u postgres  pg_ctlcluster 11 rainbow start
```

Comment s'y retrouver

Quand on arrive sur un serveur inconnu, c'est rare qu'on ait toutes les informations sur l'instance qui est censée tourner sur ce serveur...
Le plus simple pour savoir quelle instance tourne et comment, est de faire un simple ps -ef | grep postgres. Le résultat est de ce type :

```
ps -ef | grep postgres
postgres  3180  1566  0 17:34 pts/1    00:00:00
/usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql/rainbow
postgres  3182  3180  0 17:34 ?        00:00:00 postgres: checkpointer
postgres  3183  3180  0 17:34 ?        00:00:00 postgres: background writer
postgres  3184  3180  0 17:34 ?        00:00:00 postgres: walwriter
postgres  3185  3180  0 17:34 ?        00:00:00 postgres: autovacuum launcher
postgres  3186  3180  0 17:34 ?        00:00:00 postgres: stats collector
postgres  3187  3180  0 17:34 ?        00:00:00 postgres : logical replication launcher
loxodata  4238  3155  0 20:09 pts/1    00:00:00 grep --color=auto postgres
```

La ligne intéressante est la première ligne. C'est le process postmaster. Le parent de tous les autres processus qui apparaissent ensuite. On y retrouve l'option -D avec le chemin complet vers le cluster ensuite /var/lib/postgresql/rainbow.
La dernière ligne représente juste le processus de la commande qu'on vient de lancer. On peut la masquer en ajoutant || grep -v grep à la commande précédente, mais je ne connais personne qui s'embête avec ça sur des serveurs de production.

Les clients

L'étape d'après consiste à utiliser un client pour pouvoir lancer des requêtes SQL sous PostgreSQL.
Il existe de nombreux clients pour PostgreSQL. Des clients graphiques et des clients lignes de commandes. Le seul client officiel fourni avec PostgreSQL est un client lignes de commandes.
Je ne vais pas m'appesantir sur les clients graphiques, car ils sont relativement intuitifs et leur peu de fonctionnalités fait que vous en aurez vite fait le tour. Les clients graphiques ne se valent pas tous. Pour PostgreSQL, je vous conseille soit dBeaver, soit omniDB.
Quel que soit votre choix de client graphique, je vous conjure de ne l'utiliser que pour du SQL. Dès lors que vous voulez faire des opérations d'administration (y compris export/import de données), je vous conseille d'utiliser les outils fournis par la communauté pour limiter les risques pour vos données.
Si vous effectuez des requêtes plus d'une fois par semaine, je vous conseille largement de découvrir la richesse et le pouvoir des lignes de commandes.
Voici les raisons principales qui me font aimer les lignes de commandes :

- Il y a plus de fonctionnalités sur les outils lignes de commandes.
- J'utilise un multiplexeur de terminal (tmux) avec un plug-in qui stocke dans un fichier texte toutes les commandes entrées ainsi que toutes les réponses de ces commandes. Si je fais une erreur, je saurai ce que j'ai fait. De plus, quand je suis en intervention à 2 heures du matin, je n'ai pas besoin de faire le rapport d'intervention de nuit.
- Ces outils sont utilisables sur toutes les plateformes avec la même syntaxe. Y compris sur les plateformes de clients auxquelles je ne peux par exemple accéder que via un Citrix.
- Je peux scripter mes interventions pour m'éviter des erreurs.
- Les gens pensent que je pirate la CIA quand ils me voient travailler dans un café ou un TGV.

L'outil lignes de commande fourni avec PostgreSQL s'appelle psql. La documentation de psql imprimée en A4 fait 31 pages. C'est beaucoup pour un outil lignes de commandes, mais pas tant que ça pour un client de SGBDR. Apprenez à l'utiliser et vous ne le quitterez plus jamais.

Connexion

PostgreSQL est tellement sécurisé que par défaut, il n'est pas possible de se connecter à une instance depuis une autre machine. Pour le permettre, il va falloir ouvrir explicitement les flux.
La première chose à faire est de trouver les fichiers de configuration. Généralement, ils se trouvent sous le répertoire du cluster, mais ce serait trop simple s'il n'y avait pas d'exception ! Sous Debian, ils ont été mis par défaut sous /etc/postgresql/<version>/<nom_cluster>/. Pour savoir où ils se trouvent, le plus simple est de regarder le résultat de ps -ef | grep postgres. Si le process père comprend une option qui ressemble à config_file=..., vous aurez le chemin complet vers votre fichier de configuration principal.
Ce que j'appelle le fichier de configuration principal est aussi nommé postgresql.conf. Il s'agit du fichier de paramètres de l'instance.
Le deuxième fichier qui va nous intéresser est le fichier qui permet de définir qui a accès à quelle base et avec quelle méthode d'authentification. Ce fichier s'appelle pg_hba. S'il n'est pas dans le même répertoire que le fichier postgresql.conf, vous trouverez son chemin complet dans le paramètre hba_file dans le fichier postgresql.conf.

Autoriser l'écoute sur le réseau

Par défaut, le paramètre listen_addresses ne permet à l'instance de n'écouter que sur la boucle locale. Il faut donc changer ce paramètre pour permettre à l'instance d'écouter sur le réseau. On peut mettre dans ce paramètre soit '*', ce qui permet à l'instance d'écouter sur toutes les interfaces réseau fournies par la machine, soit la (ou les) adresses IP sur laquelle écoute l'instance.
La sécurité de l'instance n'étant pas gérée à ce niveau-là, il est possible de mettre '*' comme interface d'écoute pour une machine de production, sans que cela ne pose de problème.
Une fois ce paramètre changé, il va falloir redémarrer l'instance pour sa prise en compte (peu de paramètres nécessitent un redémarrage, mais celui-ci en fait partie).

```
sudo-upostgres
/usr/lib/postgresql/11/bin/pg_ctl -D /var/lib/postgresql/rainbow/ restart
```

Autoriser l'accès à l'instance

Comme indiqué précédemment, la sécurité de l'instance se trouve dans le fichier pg_hba.conf. Celles et ceux qui ont déjà configuré iptables avec un fichier texte ne vont pas être dépaysés. Le fichier pg_hba.conf fonctionne exactement de la même manière.
Il décrit un ensemble de conditions (type de connexion, quel user, quelle base de données, quelles adresses IP) qui permettent de déterminer quelle méthode d'authentification doit être appliquée (si la connexion est autorisée) ou si la connexion doit être purement et simplement rejetée.
La première ligne qui correspond sera celle qui est utilisée. Attention à bien définir l'ordre dans lequel appliquer ces règles.
Prenons comme exemple le fichier fourni par PostgreSQL :

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

Le fichier par défaut autorise les connexions que ce soit par socket ou par réseau pour tous les utilisateurs vers toutes les bases de données. S'il s'agit d'une connexion réseau, l'IP entrante doit forcément être la boucle locale (127.0.0.1/3 en IPv4 ou ::1/128 en IPv6). Dans ce cas-là, le type d'authentification demandé sera trust (ce qui signifie qu'on fait confiance, aucune demande de mot de passe n'est nécessaire). C'est à ça que servent les 3 premières lignes non commentées.
Les 3 lignes suivantes non commentées permettent une connexion de type réplication, toujours en local et toujours sans mot de passe. On pourrait se demander pour quelle raison la réplication est autorisée par défaut. En fait, cette connexion de type réplication permet également d'assurer la consistance des données pour les sauvegardes physiques à chaud.
Imaginons que je souhaite autoriser une connexion rainbow_dash à se connecter à la base de données friendship de mon instance depuis son serveur pony_place à condition qu'elle saisisse son mot de passe, j'ajouterai cette ligne au fichier :

```
host   friendship  rainbow_dash      pony_place      md5
```

(Pour connaître tous les modes d'authentifications possibles, je vous conseille de lire la documentation officielle.)
Puis, pour que cette modification puisse être prise en compte, je vais devoir dire à mon instance PostgreSQL qu'elle doit recharger ses fichiers de configuration (cela ne nécessite pas de coupure du service).

```
sudo -u postgres /usr/lib/postgresql/11/bin/pg_ctl -D /var/lib/postgresql/rainbow/  reload
```

Quelques cas d'erreur classiques

Pour vous éviter quelques recherches sur stackoverflow, voici un petit florilège des cas d'erreur les plus fréquents.

```
FATAL: authentification par mot de passe échouée pour l'utilisateur « x ».
```

Ce message signifie que le mot de passe que vous avez tenté pour vous connecter avec l'utilisateur x n'est pas le bon. Soit vous vous êtes trompé de mot de passe, soit vous ne vouliez pas qu'il vous demande de mot de passe et dans ce cas, il va falloir modifier à nouveau pg_hba.conf (ou sinon vous avez simplement oublié de recharger la configuration pour prendre en compte le changement). Il est possible également que le fichier .pgpass de l'utilisateur OS ne contienne pas le bon mot de passe.

```
FATAL: aucune entrée dans pg_hba.conf pour l'hôte « x », utilisateur « y », base de données « z », SSL actif.
```

Après passage complet de toutes les règles du fichier pg_hba.conf, aucune ne correspondait au cas qui s'est présenté, la connexion a donc été refusée. Il va falloir modifier le fichier pg_hba.conf (ou recharger la configuration si cela avait été oublié).
De manière générale, lors d'un problème de connexion, vous pouvez chercher d'où vient le problème en vérifiant cette petite check-list

- Une ligne de pg_hba correspond-elle au cas de connexion présent (vérifier le type de connexion, le nom de la base de données, le nom du user et éventuellement l'adresse IP d'origine) ?
- Le user existe-t-il (sous psql) ?
- La base de données existe-t-elle (sous psql) ?
- Est-ce que je tente la connexion sur la bonne instance (vérifier le port, le nom d'hôte) ?
- Est-ce que le user a les droits de login (select rolcanlogin from pg_roles where rolname='mon_user'?
- Est-ce que le user a le droit de connect sur cette base de données ?

Obtenir de l'aide

La documentation de PostgreSQL est bien écrite et bien détaillée. Comme le projet lui-même, elle est constamment améliorée par les modifications des différents contributeurs. La documentation a été entièrement traduite en français et la traduction est maintenue à jour. Cependant, à moins que vous ne soyez pas capables de lire de l'anglais technique, je vous conseille de vous référer systématiquement à la documentation originale, car, quelle que soit la qualité d'une traduction, il y a toujours une perte de sens entre un original et une traduction.
Voici les chapitres qui me semblent les plus importants et ce que vous trouverez dedans :

Chapitre	Contenu
II. The SQL Language	Chapitre générique sur le langage SQL ainsi que les types de données, les fonctions et les opérateurs intégrés.
III. Server Administration	Tout ce que les "ops" veulent savoir : installation, paramétrage, haute disponibilité, réplication logique, sauvegardes...
V. Server Programming	Partie plus orientée développement avec les fonctions et procédures SQL, le langage PL/pgSQL, les triggers et les règles.
VI.I. SQL Commands	La bible des commandes SQL et leur syntaxe sous PostgreSQL.
VI.II. PostgreSQL Client Applications	Les manuels de tous les utilitaires client fournis avec PostgreSQL.
VI.II. PostgreSQL Server Applications	Les manuels de tous les utilitaires serveur fournis avec PostgreSQL.
VII.52. System Catalogs	Description des tables et vues système du catalogue de PostgreSQL.
Appendix E. Release Notes	Ensemble des release notes des versions de PostgreSQL (mineures et majeures) -> À lire avant un upgrade.

Bien sûr, si vous avez des questions, vous pouvez aussi les poser à la communauté directement sur le canal slack, ou sur le canal #postgresql du IRC de freenode, ou, enfin, sur l'une des nombreuses mailing-listes de la communauté.

CONCLUSION

Cette petite introduction ne fait qu'effleurer ce qu'est réellement PostgreSQL. Dans les faits, c'est encore mieux que ça ! Utilisez-le, posez-vous des questions, lisez le code, rencontrez la communauté! Je sais que vous tomberez amoureux, comme nous tous !

•

+ de Tests = - de bugs*

Partie 2

Dans Programmez ! n°227, nous avions déjà dégrossi le sujet du test logiciel : son intérêt, les bonnes pratiques et comment les utiliser.
Oui, les tests sont importants.
Oui, les tests permettent de découvrir des problèmes de conception, des bugs et autres problèmes.
Non, le test ne fait pas tout.
Dire que l'on fait du test en faisant de l'intégration continue c'est comme si je disais que je code en C++ sans les mains. Un peu comme si tu valides un binaire avec 200 warnings en compilation.
Dans cette seconde partie, nous parlerons qualité des projets web, des tests unitaires et d'intégration, de l'automatisation des tests et enfin nous ferons un gros focus sur Selenium.
*Vous pouvez tenter le principe du « **it's not a bug, it's a feature** »***

**(si tout va bien) ** résultat non garanti.*

INFINITE SQUARE

Hamza Iqbal
Consultant web Front-End
https://blogs.infinitesquare.com/

Maîtrise de la qualité en projet web : le point de vue du développeur

Partie 2

Aujourd'hui, presque n'importe qui a la possibilité de créer son site web et même de lancer sa boutique en ligne à moindre frais. Mais qu'en est-il de la possibilité de faire des sites et des services en ligne de qualité ? Est-ce une notion applicable, mesurable et durable dans le temps ? D'ailleurs, qu'est-ce qui définit la qualité web et qui concerne-t-elle ?

niveau 100

Quels sont les objectifs du référentiel ?

Le référentiel Opquast (actuellement disponible en version 3) est une *check-list* de **226 critères** liés aux bonnes pratiques répartis en **19 catégories** (Alternatives, Code, Contact, Contenus, E-Commerce, Espaces publics, Fichiers et multimédia, Formulaires, Hyperliens, Identification, Internationalisation, Mobile, Navigation, Newsletter, Présentation, Serveur et Performances, Syndication, Sécurité et confidentialité et Tableaux).

Ce référentiel est une **base de travail**, un socle robuste et éprouvé pour tout projet web. La prise en compte de la qualité pendant la phase de rédaction du **cahier des charges** est une démarche ciblée, qui doit déterminer les risques majeurs en fonction des **attentes** liées au projet. Une **identification** et une **sélection** des bonnes pratiques clés est donc nécessaire pour le bon déroulement de toutes les phases du projet afin que le résultat réponde aux besoins. Cette sélection peut être complétée par d'autres spécifications détaillées (charte graphique, solutions techniques, règlementations, …). Le cahier des charges peut également être accompagné d'un référentiel de bonnes pratiques spécifiques au projet, élaboré avec consensus entre le commanditaire et la maîtrise d'oeuvre. 2

Le référentiel de bonnes pratiques est un **moins-disant**. Son but n'est pas de lister un maximum de règles, mais uniquement ce qui a du sens. Il ne se prononce pas sur tout et n'a pas vocation à se substituer à la loi, ni au savoir-faire des différents acteurs impliqués. Cependant, il agit en tant que garde-fou, de point de repère pour les intervenants. Il permet d'identifier et de prioriser les axes d'amélioration ainsi que d'anticiper leurs **coûts de réalisation** et de **maintien**. Il s'agit de **maîtriser** ce l'on fait et d'en garantir la pérennité.

Un référentiel est fait pour **durer**. Les bonnes pratiques ne sont pas mises à jour tous les mois en fonction des avancées technologiques ou des tendances graphiques. Les **attentes de base** des utilisateurs ne diminueront pas dans le futur et auront plutôt tendance à augmenter avec la multiplication des possibilités et des services ; c'est à ce moment-là que le référentiel pourra être enrichi.

Opquast, à travers sa démarche, pousse au pragmatisme, au réalisme et à l'amélioration progressive et continue. D'ailleurs, les propos de l'organisme sont assez clairs : LA qualité absolue **n'existe pas** et ne sera pas atteignable. Il y aura toujours des facteurs d'**insatisfaction** ou de **mécontentement** (parfois temporaires, parfois permanents) même malgré tous les efforts des intervenants. Cependant, les bonnes pratiques permettront de **résoudre** un maximum de cas problématiques et de **limiter** des risques importants et courants, le tout pour une majorité d'utilisateurs.

Il y aura toujours un équilibre à trouver entre les coûts de non-qualité et les coûts d'obtention de la qualité. De plus, la "surqualité" est également un facteur à anticiper correctement sous peine d'engendrer de gros problèmes liés à la gestion du temps et du budget alloués au projet.

La qualité web dans la pratique

Qui est concerné ?

En reprenant le **modèle VPTCS**, on peut rapidement comprendre que la qualité web couvre un grand nombre de **disciplines** : référencement, ergonomie, gestion de projet, création graphique, intégration, développement, sécurité, rédaction, juridique, marketing, communication, logistique, etc. Tous les experts de ces domaines, appartenant à la Maîtrise d'Ouvrage (MOA) ou à la Maîtrise d'Oeuvre (MOE), ont leur rôle à jouer et un **impact** non négligeable sur la qualité Web.

Ainsi, tous les intervenants pouvant prendre part au processus de conception, de réalisation et de maintien d'un site web sont concernés par la Qualité. Les bonnes pratiques Opquast étant génériques et transverses à tous les métiers, chacun peut s'en servir comme base de travail dans sa spécialité.

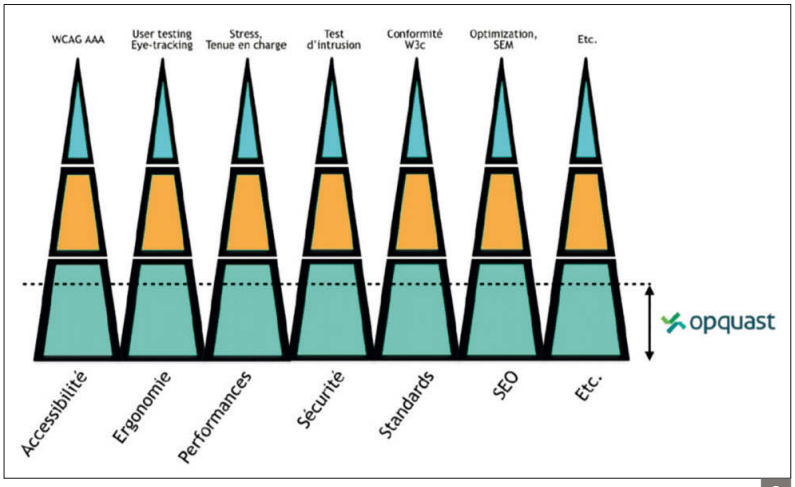
La qualité web est un **travail d'équipe**.

Quel rôle pour les profils techniques ?

Les profils techniques interviennent à plusieurs étapes du projet. Le terrain qu'ils couvrent est assez **important** puisqu'il va du code (intégration HTML/CSS, développement Front et Back) à la configuration serveur en passant par l'optimisation du référencement et de la sécurité.

Il est intéressant de noter qu'une **grande partie** des critères du référentiel Opquast est à destination des profils techniques. En effet, certaines catégories abordent exclusivement des aspects liés au code et à la sécurité. En complément, des compétences techniques seront également requises pour les critères liés à la navigation, aux formulaires, à la mobilité ou à l'internationalisation.

Même si le référentiel Opquast n'impose aucune technologie pour valider ses critères, les développeurs et autres administrateurs systèmes auront pour rôle d'utiliser tout leur **savoir-faire** pour résoudre les



Le référentiel Opquast comme socle de la chaîne de production web 2

problématiques posées avec le **choix technologique** le plus pertinent et le plus pérenne.

Comment former les professionnels ?

Deux associés d'une entreprise discutent. Le premier dit « *et si l'on forme nos employés et qu'ils s'en vont ?* » le second lui répond « *imaginons qu'on ne les forme pas et qu'ils restent ?* ».

De cette discussion vraisemblable découle une question bien réelle : Comment former les professionnels à l'application et au maintien des bonnes pratiques sur la Qualité Web ?

Opquast a donc continué son chemin en tant qu'organisme référent sur le sujet en proposant une **formation certifiante** : « Maîtrise de la qualité en projet Web ». Cette dernière a été créée avec plusieurs objectifs :

- **Informer** sur les fondements de la démarche Qualité Web et présenter les bonnes pratiques, leur utilité, leurs objectifs, leur raison d'être ;
- **Assimiler** le vocabulaire générique du Web et comprendre les enjeux de la Qualité Web et de son impact sur l'utilisation de sites et services web ;
- **Apprendre** à utiliser le référentiel de bonnes pratiques dans les différents contextes d'usage, d'abord par l'apprentissage de leurs intitulés, puis par la manière de tester et valider les différents critères ;
- **Maîtriser** la démarche Qualité Web : prévenir les risques, anticiper les coûts de non-qualité, consolider un cahier des charges avec les bonnes pratiques, intégrer les objectifs qualité dans le processus de conception et de production, impliquer et optimiser le rendement des différents spécialistes.

Selon les profils, les expériences ou les métiers, les attentes ne sont pas les mêmes face à la certification. C'est pourquoi Opquast a découpé les résultats de la certifications en **5 niveaux** : Novice, Intermédiaire, Confirmé, Avancé et Expert. 3

Le niveau découle du score obtenu à l'issue d'un examen de **125 questions**. Le score maximal est de **1000 points** (Le guide des scores : https://www.opquast.com/certification/scores/)

La certification ne nécessite aucun pré-requis technique, seulement une connaissance de base du fonctionnement du web. Le passage est donc ouvert à tous les intervenants de la chaîne de production Web quel que soit leur niveau. Toutefois, il est intéressant de noter que les scores les plus élevés sont très souvent atteints par des profils techniques tels que les intégrateurs et les développeurs Web.

Pour terminer, il est important de signaler que la certification Opquast « Maîtrise de la qualité en projet Web » est inscrite à l'Inventaire National des Certifications Professionnelles. (https://inventaire.cncp.gouv.fr/fiches/2363/).

Conclusion

A travers cet article, nous avons pu définir et comprendre en quoi consistait la démarche Qualité Web. Une introduction au modèle VPTCS et au référentiel de bonnes pratiques d'Opquast nous a également permis de prendre conscience des enjeux et des risques liés à la non-qualité des services en lignes que nous réalisons.

Il a été intéressant de noter que beaucoup de bonnes pratiques sont liées à la qualité du code et nécessitent des compétences techniques en intégration, développement, sécurité et référencement.

Nous avons également pu constater que la certification proposée par Opquast est un moyen complémentaire aux profils techniques de valider leurs acquis et de mettre en oeuvre leurs compétences au service de la Qualité Web. •

Les différents niveaux de la certification Opquast 3

Anthony Giretti
MVP, MCSD
Developer, Blogger, Speaker
anthony.giretti@gmail.com
http://anthonygiretti.com

Tests unitaires et tests d'intégration sur ASP.NET Core 2.2.

Partie 2

Souvent négligés, les tests unitaires permettent de vérifier qu'une implémentation reste conforme à la règle de fonctionnement attendue, évitant les régressions en cas d'évolution du programme. Les tests d'intégration, eux, servent à vérifier ce que les tests unitaires ne peuvent pas faire : tester le fonctionnement de chaque couche du logiciel entre elles.

niveau 200

PRÉPARATION ET CRÉATION D'UN TEST INTÉGRÉ AVEC TESTSERVER

Préparation des tests en émulant un serveur

La première étape avant de tester WebAPI consiste à créer un serveur émulé à l'aide de la classe *TestServer* fournie par l'assembly: *Microsoft.AspNetCore.TestHost*.
Qu'allons-nous faire exactement ? Nous allons créer un client http à partir grâce à *TestServer :*

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.TestHost;
using System;
using System.Net.Http;
using WebApiDemo;

namespace IntegrationTestsDemo
{
  public class TestServerFixture : IDisposable
  {
    private readonly TestServer _testServer;
    public HttpClient Client { get; }

    public TestServerFixture()
    {
      var builder = new WebHostBuilder()
          .UseEnvironment("Development")
          .UseStartup<Startup>();

      _testServer = new TestServer(builder);
      Client = _testServer.CreateClient();

    }

    public void Dispose()
    {
      Client.Dispose();
      _testServer.Dispose();
    }
  }
}
```

WebHostBuilder est facile à configurer. Il vous suffit de définir l'environnement que vous souhaitez utiliser pour vos tests et de déclarer la classe de démarrage *(Startup.cs)* que vous souhaitez utiliser. Dans cet exemple je reprends la même configuration en utilisant le *Startup.cs* de ma Web API. Ensuite, il vous suffit d'instancier la classe *TestServer* qui prend en paramètre une instance de *WebHostBuilder* définie précédemment. C'est tout!

Création d'un test intégré

Pour la création du test, nous allons reprendre la même méthodologie précédemment décrite. Nous allons tester un contrôleur qui va jeter une exception et nous allons regarder si en intégration le comportement de la Web API est celui attendu : une erreur 500 avec un objet d'erreur correctement formaté.

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;

namespace WebApiDemo.Controllers
{
  [Route("api/[controller]")]
  [ApiController]
  public class DemoExceptionController : ControllerBase
  {
    public DemoExceptionController()
    {
    }

    [HttpGet]
    public IEnumerable<string> Get()
    {
      throw new Exception("bouhhhh quelle affreuse erreur!");
      return new string[] { "value1", "value2" };
```

```
    }

  }
}
```

Et maintenant notre test :

```
using FluentAssertions;
using System.Collections.Generic;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using WebApiDemo.Models;
using Xunit;

namespace IntegrationTestsDemo
{
  public class DemoExceptionControllerTests
  {
    [Fact]
    public async Task WhenGetMethodRaiseAnException_WebAPIShouldHandleItandAnswer
AProperErrorObjectAndStatusError500()
    {
      //Arrange
      using (TestServerFixture fixture = new TestServerFixture())
      {
        // Act
        var response = await fixture.Client.GetAsync("/api/DemoException");

        // Assert
        var responseData = await response
        .Content
        .ReadAsAsync<Error>();

        responseData
        .Should()
       .BeEquivalentTo(new Error { Message = "Unhandled error", Errors = new List<string>
(), Code = "00009" });

        response.StatusCode
        .Should()
        .Be((int)HttpStatusCode.InternalServerError);

        response
        .Content
        .Headers
        .ContentType
        .MediaType
        .Should()
        .Be("application/json");
      }
```

```
    }


  }
}
```

De la même manière encore que dans notre test unitaire, nous vérifions le contenu de la réponse, le code http retourné et le Content-type avec *FluentAssertions*.
Est-ce que la réponse retournée correspond bien aux attentes ?

```
var responseData = await response
        .Content
        .ReadAsAsync<Error>();

        responseData
        .Should()
       .BeEquivalentTo(new Error { Message = "Unhandled error", Errors = new List<string>
(), Code = "00009" });
```

Est-ce que le statut Http correspond bien aux attentes ?

```
response.StatusCode
    .Should()
    .Be((int)HttpStatusCode.InternalServerError);
```

Est-ce que le content-Type correspond bien aux attentes ?

```
response.Content
    .Headers
    .ContentType
    .MediaType
    .Should()
    .Be("application/json");
```

Conclusion

Vous avez vu comment faire facilement des tests unitaires et des tests d'intégration.
C'est une très importante fonctionnalité que vous devez implémenter lorsque vous développez un programme tel que Web API.
Cependant vous n'êtes pas obligé d'utiliser les mêmes outils que nous, surtout en ce qui concerne les tests unitaires.
Il y a des alternatives à *Xunit*, telles que *Nunit*, *MSTest V2*, vous pouvez également utiliser *NFluent* ou tout simplement utiliser les assertions de *Xunit* ou celles de *Microsoft*. (*Assert.Equals*, *Assert.Contains etc.…*). Enfin en termes de framework de mocking le choix est plus conséquent, il existe *Moq*, *Rhino.Mocks*, *FakeItEasy*.

Maintenant à vous de jouer !

Automatisation des tests logiciels : les enjeux de l'industrialisation

Partie 2

Jean-Baptiste Marcé *Co-founder, CEO/CTO CloudNetCare*

Dans la partie précédente, nous avons vu un aperçu des différents types de tests automatisables, de la responsabilité des différentes équipes (développeurs, testeurs, automaticiens, ...), des bénéfices de l'automatisation mais aussi des risques à appréhender.

Nous allons maintenant nous focaliser sur les enjeux actuels de l'automatisation grâce à nos retours d'expériences concrets et réussis d'industrialisation de tests automatisés. Nous évoquerons les avantages, les risques, et, surtout, les éléments à prendre en compte pour réussir un processus d'automatisation.

NOTRE FOCUS : L'INDUSTRIALISATION DE L'AUTOMATISATION DES TESTS FONCTIONNELS UI

Les tests automatisables qui ont le plus de « valeur » (au sens aptitude à détecter des défauts), car au plus proche du besoin de l'utilisateur final (cf. définition de la qualité par IEEE), sont aussi ceux qui sont les plus complexes à créer et à maintenir : les tests fonctionnels UI.

Lorsque la couverture en tests unitaires est importante, la nécessité d'avoir une couverture en tests fonctionnels UI est présente, mais peut être optimisée sur des parcours spécifiques. Si celle-ci est faible ou inexistante, la couverture en tests fonctionnels UI doit devenir le filet de protection de l'application.

Dans les processus de tests automatisés, nous distinguons trois phases distinctes et récurrentes :

Phase 1 : la création et la maintenance des scénarios de tests

Il s'agit de la définition des parcours de tests (scénarios / mots clés / jeux de données) et de leur maintenance au fil des itérations. Plusieurs méthodes existent :

- *Méthode « Capture + Replay » avec données statiques ou variabilisées*
  Cela consiste à enregistrer un parcours utilisateur en le jouant dans le contexte d'exécution. Les actions manuelles sont enregistrées pour être reproduites ensuite.
  Les données enregistrées peuvent rester statiques ou être variabilisées.
  Cette méthode est simple pour la création rapide de tests mais complexe à maintenir au cours des itérations.
- *Méthode structurée basée sur des scripts*
  Cela consiste à scripter les parcours et utiliser une sémantique dédiée qui décrit les actions.
  Cela est très puissant et modulable mais attention d'éviter les langages de scripts propriétaires !
  La maintenance demande à l'automaticien de l'expérience et des compétences dédiées.
- *Méthode par « mots-clés »*
  Cela consiste à découpler la complexité de l'exécution et celle des notions fonctionnelles à tester.L'automaticien peut générer des fonctions élémentaires avec une des méthodes précédentes et les rendre disponibles sous la forme de mots-clés (potentiellement avec des variables).
  Les mots-clés auront une sémantique fonctionnelle. Il suffit alors de d'enchaîner les mots-clés pour créer le parcours de test.
  Si le contexte technique évolue (mise à jour du navigateur qui demande une adaptation de script par exemple), l'automaticien modifie le script, mais le test reste inchangé (enchaînement des mots-clés).

Il existe bien entendu des outils qui prennent en charge toutes les méthodes et qui peuvent les combiner (exemple : la « capture » génère un script utilisable avec des mots-clés).

Phase 2 : l'exécution des tests

Il s'agit de la phase d'automatisation proprement dite : des robots exécutent les parcours sur des cibles (ex : Navigateur, smartphone, …) avec des jeux de données.

Il est impératif d'étudier le comportement de l'outil d'automatisation face aux problématiques suivantes :

- Parallélisme :
  Etudier comment le moteur d'orchestration gère le séquençage des tests : il en découle une durée d'exécution de l'ensemble de la couverture.
- Segmentation des devices (navigateurs / smartphones / tablettes …) :
  Au vu du nombre de versions de navigateurs existantes ou du nombre de téléphones différents, cette problématique est fondamentale et elle est liée au point précédent (parallélisme).
  Si la solution est déployée sur site, évaluer la configuration hardware / software nécessaire.
- Stratégie de gestion des faux positifs/négatifs :
  Comment l'outil gère-t-il les cas suivants (non exhaustifs) :
  - Indisponibilité temporaire d'un service tier (exemple : l'authentification) ;
  - Accès internet temporairement hors-service (si l'applicatif nécessite une connexion internet) ;
  - Un problème hardware sur un téléphone (pour un test d'app mobile).

Phase 3 : l'analyse des résultats et découverte des régressions

Il s'agit d'étudier le processus de restitution de la phase 2, la pertinence des résultats et la fiabilité des informations disponibles pour investigation.

Il faut mettre en place un processus d'analyse des régressions pour fiabiliser les cas avérés et fournir un maximum d'informations aux développeurs.

Il est important que tous les résultats soient historisés pour suivre l'évolution de la couverture au fil des itérations.

Les résultats peuvent être présentés dans des tableaux de bord dédiés ou remontés dans des outils de consolidation via des API.

Analyse des différents types de solutions disponibles et des enjeux :

Nous allons analyser les deux principaux cas usuels : le monde Desktop Web (navigateurs) et le monde des applications mobiles.

Le Monde Desktop / Web (Navigateur)

Principaux enjeux / difficultés de mise en œuvre

- Mise à jour des navigateurs en continu

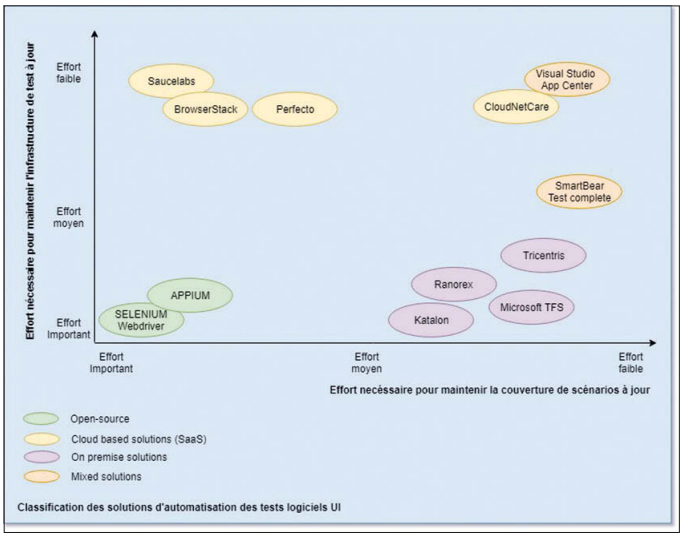
La fréquence de mise à jour des navigateurs est très rapide (environ tous les 15 jours) et celle-ci est automatique et obligatoire pour l'utilisateur final (sauf contextes spécifiques). Il faut donc en permanence mettre à jour l'environnement des robots avec les dernières versions. Il faut s'assurer que le pilotage de la nouvelle version du navigateur reste entièrement opérationnel et n'entraîne pas de faux positifs/négatifs.

- Gestion des OS (Windows/MacOS/Linux)

Les Updates de sécurité des OS sont fréquents, et, de la même manière que les navigateurs, ils sont automatiques et obligatoires. Il faut donc toujours s'assurer que l'environnement d'exécution des robots évolue en permanence.

- Reproductibilité des contextes

Il est important que chaque exécution soit initiée dans un contexte strictement identique et reproductible. Par exemple le navigateur doit systématiquement se retrouver dans un état initial identique à chaque nouveau test (Cache Vide, pas de cookies, …).
Le Cloud est particulièrement adapté à ce contexte.



Trois types d'outils d'automatisation

Les outils open source : principalement la communauté SELENIUM

• SELENIUM IDE (scripting)

Très populaire, il s'agit d'un Plug-in Firefox & Chrome qui permet d'enregistrer des parcours et de les sauvegarder dans un format *.SIDE* (Json). Il s'agit de scripts Json qui sont utilisables dans beaucoup d'outils d'automatisation. Cela est le standard des scripts de tests UI.

« Command-line Runner » permet d'exécuter des scripts .SIDE en parallèle via un service Node.JS sur site ou en mode Cloud.

• SELENIUM Webdriver / GRID

SELENIUM Webdriver est un ensemble de composants logiciels qui permettent de piloter des navigateurs Web. C'est le framework de référence pour automatiser les tests navigateurs. Il permet de coder des tests dans plusieurs langages de développement. L'automaticien est donc un développeur qui utilise son langage habituel (Java, C#, Python, …). Pour l'exécution des tests en parallèle, il est possible de monter une SELENIUM GRID et de gérer le déroulé des tests. L'avantage est d'être libre de coder son propre moteur d'exécution et d'analyse, mais attention, c'est une charge de travail conséquente !

Les outils Desktop sur site

Les outils « propriétaires », gratuits ou pas, peuvent être très évolués et couvrir plusieurs aspects des tests (conception, exécution, reporting, suivi, …).

Il est important d'étudier les caractéristiques des différents outils en prenant soin de bien appréhender les risques décrits plus haut.

La principale difficulté reste le déploiement sur une infrastructure locale à maintenir en permanence avec des mises à jour logicielles et matérielles quasi-permanentes.

Les services en mode SaaS

Les outils en mode SaaS déportent la problématique d'exécution dans des Clouds privés. Ils peuvent se contenter d'exécuter des scripts sur des devices ou embarquer toute une logique de gestion de scénario (mode Record/Replay, scripting, mots-clés) et d'analyse de résultats évolués (screenshots et vidéos des tests en erreur, comparaison d'images pour détection des régressions graphiques, etc.).

L'intérêt principal est de déléguer toutes les problématiques de mise à disposition des ressources (derniers navigateurs, smartphones, tablettes) à un tier dont c'est le métier. Et bien sûr le paiement à l'usage.

Il est important de valider l'ouverture des solutions SaaS et en particulier la richesse des API qu'ils exposent.

Monde Mobile (smartphones/tablettes)

Principaux enjeux / difficultés de mise en œuvre

- Devices Réels vs Emulateurs
  Il faut opérer un choix sur la couverture des devices en fonction des cas de tests. En effet, les simulateurs sont purement logiciels et facilement scalables mais ne testent pas le Hardware, ni des couches logicielles constructeur et opérateur (surtout dans le monde Android) ;
- Comment aborder la segmentation (Hardware/Software)
  La combinaison du nombre de devices par le nombre de versions d'OS devient vite exponentielle. Il faut donc faire un choix pragmatique idéalement basé sur les statistiques du parc réel des utilisateurs et revoir ce parc régulièrement ;
- Technologie de pilotage des devices
  Vérifier que toutes les spécifications techniques pour les besoins des tests sont couvertes par l'outil (ex : URL profondes, envoi de SMS, …) ;
- Gestion des packages & environnement de test (app/ipa/apk)
  Industrialiser les processus de génération de packages pour l'automatisation.
  Envisager la création d'un environnement Backend de test dédié (exemple : service d'authentification) avec des packages « buildés » en mode DEBUG ;
- Il ne faut pas sous-estimer la gestion des problématiques hardware (mémoire/batteries/connectivité 4G …) ;
- Anticiper la gestion des tests complexes : Géolocalisation, connexions Bluetooth, … ;
- Intégration continue (tests automatisés via API).

Les outils open source : principalement la communauté APPIUM

APPIUM est l'équivalent de SELENIUM dans les tests mobiles. Il s'agit d'un framework logiciel de pilotage générique agnostique au device. Il pilote aussi bien des devices Android réels ou simulateurs que des devices iOS. L'automaticien est donc un développeur qui utilise son langage habituel (Java, C#, Python, …) et qui code la logique fonctionnelle en Java, C# ou autre. Les tests ainsi créés peuvent être exécutés localement ou dans des Cloud privés.

Les outils desktops

La principale difficulté reste le déploiement sur une infrastructure locale à maintenir en permanence sur des matériels en constante évolution (parc de téléphone / tablette).

Les services en mode SaaS

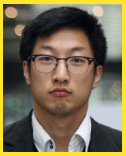
Les avantages décrits plus haut pour le monde Web restent identiques et sont d'autant plus flagrants que le parc matériel est beaucoup plus vaste et évolutif.

Lors du choix d'un outil d'automatisation, dans l'objectif d'industrialiser les processus de test, les deux axes principaux qui ont un impact direct sur le budget à consentir pour les tests sont :

- L'effort nécessaire pour maintenir la couverture de test (maintenance des scénarios au cours des itérations) : c'est un impératif, sinon l'échec à long terme est presque inévitable !
- L'effort nécessaire pour maintenir l'infrastructure d'exécution en permanence à jour en phase avec le parc réel des utilisateurs (exemple : mise à jour des navigateurs, mise à jour des smartphones, …)

Conclusion

L'automatisation des tests logiciels est un secteur en mouvement permanent et les acteurs en présence doivent être de plus en plus réactifs. L'industrialisation des processus de tests n'est plus une option pour qui veut concevoir aujourd'hui un système logiciel fiable et évolutif. Le test est également un métier spécifique ; au-delà de la question « quel outil choisir ? » on peut aussi se demander « est-ce que j'automatise mes tests en interne ou est-ce que je mandate un tiers de confiance dont c'est le métier ? ». Dans les deux cas, la réponse dépend de votre métier, de votre contexte et votre stratégie IT. Les solutions existent et si le choix est celui de l'externalisation, la proximité est clairement un facteur de succès. •



Vivien Fabing
Lead ALM, Infinite Square
https://blogs.infinitesquare.com/
@vivienfabing

INFINITE SQUARE

Selenium : les tests automatisés d'interface graphique d'application web de nos jours

À l'heure de **l'Agilité** *et du* **Déploiement continu,** *mettre en place des tests automatisés n'a jamais été aussi important. En complément de la mise en place de tests unitaires et de tests d'intégration, la mise en place de tests d'interface graphique est un vrai défi en soi, notamment du fait de leur faible résistance au changement (d'implémentation, d'interface graphique, etc.).*

De plus, suite à l'annonce récente de l'abandon du support des **Visual Studio Coded UI Test** dans les versions ultérieures à celle de 2019 (https://docs.microsoft.com/en-us/visualstudio/releases/2019/ release-notes-preview#test-tools), la question du choix du Framework de test peut apparaître.

En ce qui concerne le test d'interface graphique d'applications web, le Framework **Selenium** semble s'être imposé comme standard depuis de nombreuses années grâce à sa robustesse et son énorme communauté open source. C'est d'ailleurs le Framework recommandé officiellement par Microsoft depuis l'annonce du support des Coded UI Test.

Dans ce dossier, nous allons voir plus en détail les notions à avoir à l'esprit lorsque l'on se lance dans la création d'un plan de tests automatisés avec Selenium. Nous commencerons par aborder l'utilisation d'un enregistreur de tests appelé Katalon Studio, puis nous verrons l'architecture type d'un scénario de test Selenium. On s'arrêtera ensuite pour voir ce qui se cache derrière la notion de WebDriver, avant d'aborder la notion clé qui permet d'avoir des tests d'interface maintenables : le pattern Page Object. Nous finirons ce dossier par un exemple de mise en place de tests multi-navigateurs via BrowserStack.

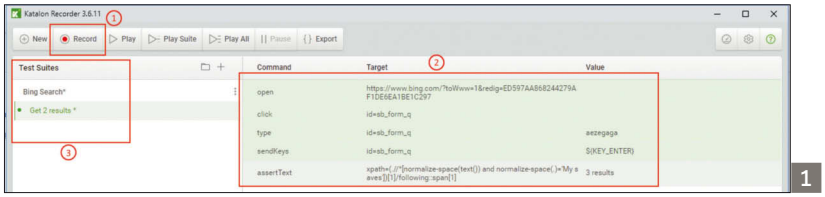
Démarrer en douceur et gagner en productivité avec Katalon Recorder

Écrire des tests Selenium en C# n'est pas très compliqué avec un peu de pratique, mais il peut être intéressant de s'armer d'un outil d'enregistrement, notamment lorsque l'on débute sur le sujet. **Katalon Recorder** se présente sous la forme d'une extension gratuite Chrome ou Firefox. Il permet d'enregistrer automatiquement des actions, des assertions, etc., et de générer le code Selenium en C# correspondant à ces enregistrements.

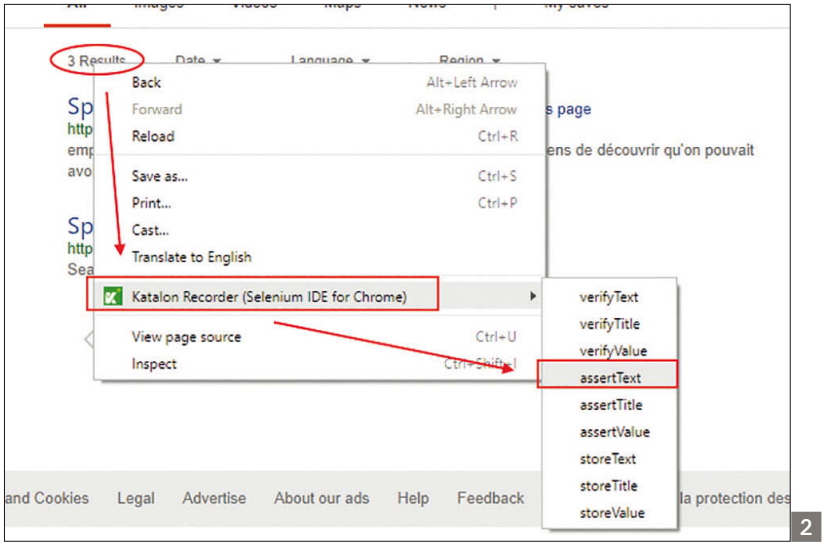
Enregistrement d'un scénario de test avec Katalon Recorder

Une fois l'extension ajoutée au navigateur, Katalon Recorder s'ouvre dans une fenêtre modale et permet de démarrer l'enregistrement d'un test via le bouton **Record** (1). Aussitôt, le navigateur se focalise sur un onglet et chaque action effectuée à l'intérieur de cet onglet est capturée par l'enregistreur.

L'enregistreur essaie de toujours s'appuyer sur un **id** lorsque celui-ci est disponible (cela a le mérite d'offrir en général de meilleures performances et une plus grande robustesse au changement d'interface graphique), mais est capable néanmoins de générer des



1



2

requêtes **XPath** afin de situer un élément de la page web (2). 1

À noter également que Katalon Recorder propose une organisation simple de ses scénarios de tests via des suites de tests, qui peuvent correspondre à une fonctionnalité ou un module donné, et permet ainsi de regrouper tous les cas de tests associés, et pouvoir plus facilement les exécuter (3).

Vérifier / Effectuer une assertion sur un élément d'une page web avec Katalon Recorder

Un scénario de test ne serait pas complet sans y rajouter des assertions pour vérifier que le scénario s'est bien déroulé. L'enregistreur propose donc, via le menu affiché lors d'un clic droit sur l'élément à vérifier, d'ajouter une assertion sur le texte, la valeur, etc., d'un élément. 2

Génération du code Selenium en C# à partir du scénario enregistré

Une fois le scénario enregistré, le bouton **Export** disponible sur l'extension permet de générer le code Selenium en C#, basé sur le

Framework de test **MSTest** ou **NUnit**.
Si tout le code généré permet une base pour démarrer rapidement un projet de test Selenium en C#, il est généralement plutôt conseillé de s'inspirer du code généré, notamment celui contenu dans la méthode préfixée par [**TestMethod**], et de l'intégrer dans son projet de test, qui aura pris soin de respecter une architecture maintenable (*POM*, *Webdriver Factory*, etc.). 3
Plutôt que d'inspecter les éléments d'une page un à un, il peut être intéressant de se baser sur un enregistreur (notamment lorsque l'on manque d'expérience sur le sujet, et notamment les sélecteurs XPath) afin de gagner du temps sur l'écriture du Framework. Néanmoins, il est important de garder en tête que lorsque l'on souhaite se constituer un Framework de test d'interface suffisamment robuste (*i.e. qui ne pète pas de partout au moindre déplacement d'un élément graphique*), le respect de pattern éprouvés tels que le POM (Page Object Model) est indispensable.

Comment architecturer un scenario de test Selenium : les bases

Les grandes étapes pour écrire un test Selenium sont généralement assez similaires : ouverture du navigateur, interaction avec éléments des pages web, assertions (très important pour vérifier que le test s'est bien déroulé), puis fermeture du navigateur. Selenium.WebDriver offre de nombreuses possibilités, et possède quelques petites particularités qu'il peut être bon de connaître pour s'éviter des problèmes de maintenance ou de performance par la suite.

Étape 1 : Instanciation des navigateurs web

La première étape importante dans l'écriture d'un test Selenium est de définir le ou les navigateurs web à tester. Pour chaque navigateur cible, il existe un **WebDriver** équivalent qui va permettre de traduire le scénario de test en instructions compréhensibles par le navigateur. Pour démarrer rapidement et se faire la main avec Selenium, il est possible d'utiliser « directement » l'un des WebDrivers disponibles (i.e. FirefoxDriver, ChromeDriver, etc.). Cependant, il est plutôt recommandé de se baser sur l'interface générique **IWebDriver**, implémentée par tous les WebDrivers, qui permet de constituer un scénario de test techniquement agnostique au navigateur web testé.
L'utilisation de l'interface IWebDriver est un choix important, car l'interface expose des méthodes génériques pour interagir avec le navigateur web, qui sont parfois différentes des méthodes exposées par les WebDrivers de chaque navigateur. Exemple pour la récupération d'un élément via son id en utilisant le ChromeDriver (spécifique à Chrome) et d'un autre exemple utilisant l'interface IWebDriver :

ChromeDriver :

```
var driver = new ChromeDriver();
driver.Navigate().GoToUrl("https://bing.com");
driver.FindElementById("sb_form_go").Click();
```

IWebDriver :

```
IWebDriver driver = new ChromeDriver();
driver.Navigate().GoToUrl("https://bing.com");
driver.FindElement(By.Id("sb_form_go")).Click();
```

La différence de syntaxe est subtile, mais écrire le code de ses tests en utilisant la syntaxe du IWebDriver permettra de réutiliser ses scénarios de tests à l'identique pour n'importe quel navigateur web.
Ensuite, l'un des moyens les plus simples pour récupérer les WebDrivers des navigateurs cibles en .NET est d'utiliser les packages NuGet correspondants. 4
L'avantage de passer par NuGet est de pouvoir facilement mettre à jour le WebDriver en mettant à jour notre package NuGet. À la vitesse où les navigateurs sont mis à jour aujourd'hui, c'est bien entendu indispensable .

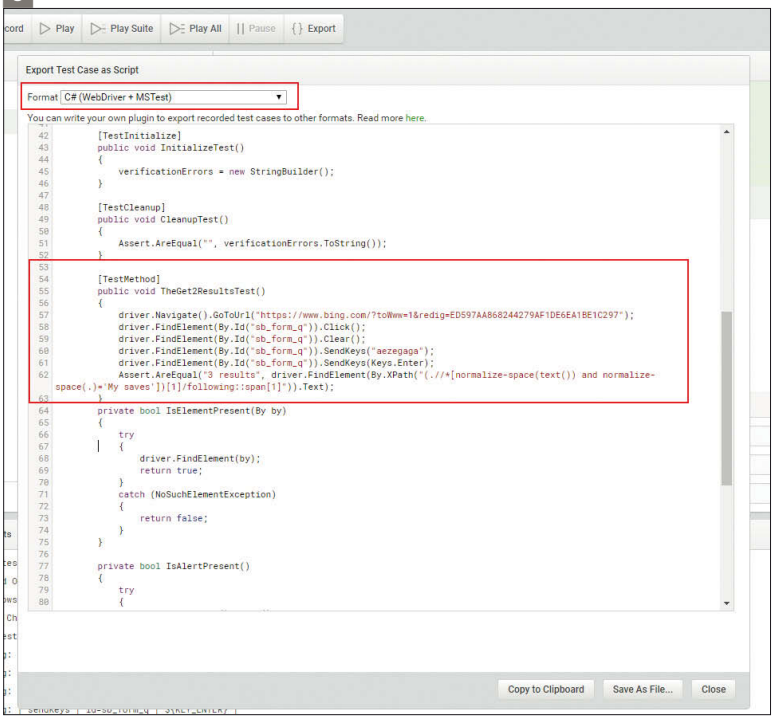
Étape 2 : Les actions, récupération de valeurs et assertions

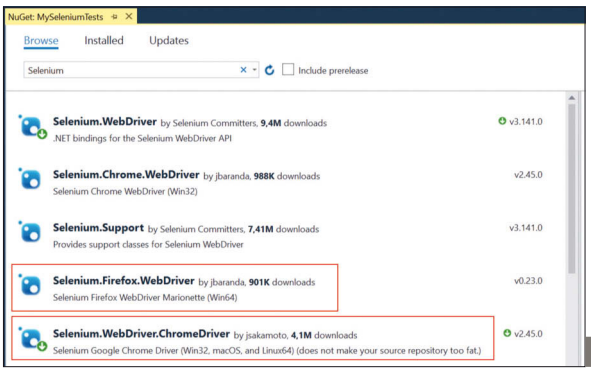
Une fois notre IWebDriver prêt, il est maintenant temps de voir les différentes interactions possibles avec nos navigateurs web via Selenium (Spoiler : il est possible de quasi tout faire).

La navigation avec Selenium 5

L'une des premières actions à effectuer est généralement la navigation vers une page web. Pour cela, on fait appel à la méthode **Navigate()** qui nous renvoie un **INavigation**, lui-même permettant de naviguer vers une url via la méthode **GoToUrl()**.
Par ailleurs, ce INavigation nous permet d'effectuer d'autres

3





4

actions telles que rafraichir la page (méthode **Refresh()**), revenir en arrière (méthode **Back()**) ou aller à la page suivante de l'historique (**Forward()**).

La sélection d'éléments avec Selenium

Ensuite vient la sélection d'un élément dans la page web pour pouvoir récupérer ses propriétés (le texte affiché, son état, etc.), ou même pouvoir agir dessus (saisir du texte, cliquer dessus, le survoler, etc.). 6

Depuis un IWebDriver, on utilise la méthode **FindElement()** pour sélectionner un élément ou la méthode **FindElements()** pour en sélectionner plusieurs, de type **IWebElement**. Ces méthodes prennent en argument un objet **By** qui s'initialise facilement via l'une des méthodes disponibles :

- **By.Id()** : permet de localiser un élément par son **id** (la valeur de l'attribut id de la balise html plus précisément).
- **By.Name()** : permet de localiser un ou plusieurs éléments par leur **name** (la valeur de l'attribut name de la balise html plus précisément).
- **By.ClassName()** : permet de localiser un ou plusieurs éléments par leur **class** (la valeur de l'attribut class de la balise html plus précisément).
- **By.TagName()** : permet de localiser un ou plusieurs éléments par leur **nom de balise** (exemple : `a` ou `div`).
- **By.LinkText() ou By.PartialLinkText()** : permet de localiser un ou plusieurs éléments suivant la valeur textuelle affichée par un lien html (soit la balise `a`). By.LinkText() recherchant une valeur exacte alors que By.PartialLinkText() recherche une valeur contenue dans le lien.
- **By.CssSelector()** : permet de localiser un ou plusieurs éléments par un sélecteur **CSS** (exemple : `#header div > p`).
- **By.XPath()** : permet de localiser un ou plusieurs éléments par un sélecteur **XPath** (exemple : `//div[@class='toto']//a`).

En termes de performance et de maintenabilité, la sélection des éléments par leur `id` (By.Id()) est à privilégier fortement. Cela permet de déplacer les éléments dans la page, de les imbriquer dans d'autres composants, tout en permettant à Selenium de toujours être capable de sélectionner les éléments par leur id.

Lorsqu'un élément ne possède pas d'id, on peut alors le localiser avec d'autres sélecteurs (By.Name(), By.ClassName(), By.TagName(), ByLinkText() ou By.PartialLinkText()), potentiellement plus difficiles à maintenir de par leur non-unicité et également légèrement moins performants.

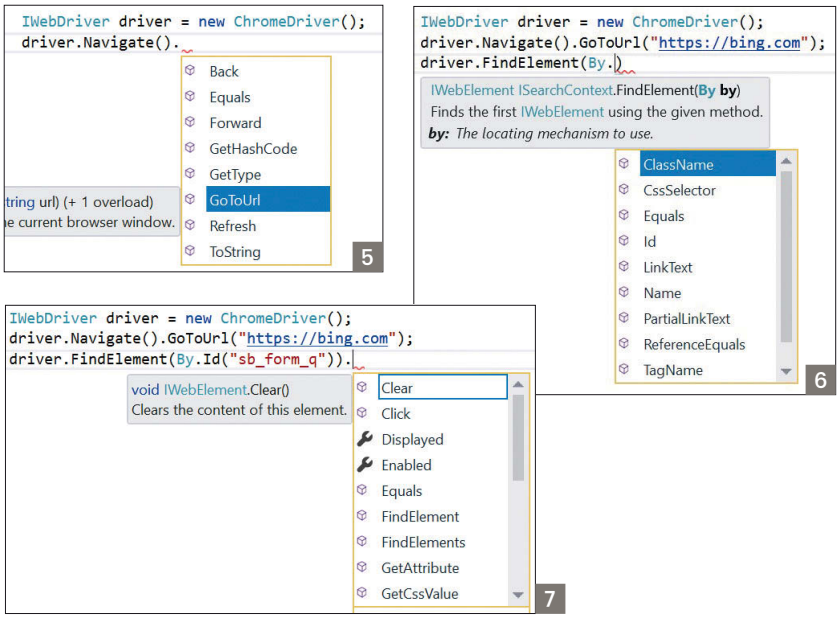
Dernières options disponibles pour localiser un élément : par sélecteur CSS (By.CssSelector()) ou par requête XPath (By.XPath()). Ces 2 sélecteurs sont très puissants, mais sont à manipuler avec précaution car, suivant la syntaxe de notre sélecteur CSS/requête XPath, les performances peuvent être drastiquement impactées.

Anecdotiquement, le sélecteur CSS est généralement plus rapide que la requête XPath.

Interagir avec les éléments via Selenium 7

Une fois le ou les éléments sélectionnés, et donc un IWebElement récupéré, Selenium fournit toute une batterie de propriétés et méthodes pour pouvoir interagir avec les éléments :

- **Cliquer** sur un élément avec la méthode **Click**() ;
- **Effectuer une saisie clavier** d'un texte sur un élément avec la méthode **SendKeys**(string text) ;



- **Effectuer des vérifications / assertions** en récupérant la valeur d'une propriété d'un élément telle que son text avec la propriété **Text**, un attribut avec la méthode **GetAttribute** (string attributeName) ou même son état avec les propriétés **Selected**, **Displayed** et **Enabled**, qui retournent la valeur `true` lorsque l'élément est, respectivement, sélectionné, affiché et modifiable.

Il reste encore bien d'autres méthodes et/ou propriétés disponibles pour interagir avec ces éléments :

- **Clear**() : méthode pour vider le contenu dans un élément (du texte par exemple) ;
- **Submit**() : méthode pour valider le formulaire parent du composant sélectionné ;
- **Location** : propriété qui retourne les coordonnées x-y du premier point en haut à gauche de l'élément ;
- **Size** : propriété qui retourne la taille largeur-hauteur de l'élément ;
- **TagName** : propriété qui retourne le nom de la balise html sélectionnée ;
- **GetCssValue**(string propertyName) : méthode qui retourne la valeur d'une propriété CSS ;
- **GetProperty**(string propertyName) : méthode qui retourne la valeur d'une propriété du DOM/JavaScript (exemple : checked, value, etc.).

Étape 3 : La fermeture des navigateurs web

Enfin, il est important de quitter correctement le navigateur web pour s'assurer qu'aucun processus fantôme ne subsiste et n'entraîne de fuite mémoire sur la plateforme de test.

Pour cela, deux choses importantes sont à garder en tête :

- S'assurer que l'opération de nettoyage soit effectuée dans tous les cas, et ce, même si le test fail, par exemple en plaçant le code de nettoyage dans une instruction de [TestCleanup] (voire [ClassCleanup]) dans le cas de MSTest.
- Il existe 3 méthodes pour quitter le navigateur web : **Close**(), **Quit**() et **Dispose**() :
  - Les 2 dernières (**Quit** et **Dispose**) sont des synonymes et sont à privilégier car elles vont nettoyer toutes les instances ouvertes durant l'exécution du test.

- La première (**Close**) ne va fermer que la fenêtre du navigateur qui a le focus. Cette méthode est plus encline à laisser des ressources non nettoyées en fin d'exécution de test, et est donc à réserver pour des usages bien précis et maîtrisés (test de reprise de contexte lors de la réouverture d'un navigateur, etc.). 8

Et ensuite ?

Nous avons vu à quel point il est simple de mettre en place un test Selenium en C#. C'est une bonne première étape et cela permet déjà de se lancer dans la création d'un plan de test automatisé. Nous verrons plus loin qu'il existe également quelques bonnes pratiques de pattern à mettre en place, qui s'avèrent très vite indispensables pour garantir une meilleure maintenabilité pour des plans de tests assez volumineux.

Deep Dive Selenium Driver : l'envers du décor

Après avoir vu comment démarrer sur Selenium, il est temps de clarifier un peu la magie mise en place derrière tout ça .

Je résume le processus pour démarrer via Visual Studio :

- Créer un nouveau projet de test **MSTest** ou **XUnit ;**
- Référencer le package **Selenium.WebDriver ;**
- (*Optionnellement référencer les packages NuGet des navigateurs cibles tels que Selenium.WebDriver.ChromeDriver, Selenium.Firefox.WebDriver, etc.*) ;
- Instancier un nouveau **RemoteWebDriver** (*ChromeDriver, FirefoxDriver*, etc.) ;
- Appeler les méthodes **Navigate**() pour naviguer vers une page, **FindElement**() pour interagir avec les contrôles, etc.
- Ne pas oublier d'appeler **Quit**() ou **Dispose**() pour fermer les navigateurs proprement.

Exemple de code Selenium : 9

Voyons plus en détail la partie WebDriver.

Architecture du fonctionnement de Selenium

Prenons par exemple le cas du ChromeDriver : celui-ci nécessite donc l'installation des 2 packages NuGet **Selenium.WebDriver** ainsi que **Selenium.WebDriver.ChromeDriver**. 11

Le fonctionnement est le suivant :

- Du code C# permettant de piloter les tests Selenium est écrit (1), tirant parti des classes génériques telles que *RemoteWebDriver* contenues dans la librairie **Webdriver.dll** (2) ;
- Les instructions sont ensuite traduites en requêtes *http POST*, suivant le récent standard W3C du WebDriver Protocol, standard suivi par les navigateurs les plus classiques (Chrome, Firefox, Edge, etc.), et sont envoyées au WebDriver **chromedriver.exe** (3) ;
- Le **chromedriver.exe,** quant à lui, est une web application self-hostée, qui expose des endpoints http en POST, et qui se charge de communiquer avec le navigateur cible : par exemple dans le cas de *chrome*, les instructions sont envoyées à **chrome.exe** (4) via les *WebSockets des DevTools de remote debugging* et pour *Firefox*, c'est le *Firefox remote protocol* qui est utilisé pour communiquer. 10

Pour illustrer ce fonctionnement, je propose une petite démonstration qui montre comment communiquer avec le chromedriver.exe directement en requêtes http POST via Postman.

Selenium4fun : piloter Selenium via Postman

Commençons par aller voir du côté du package NuGet Selenium.WebDriver.ChromeDriver. Celui-ci contient donc notre fameux **chromedriver.exe** : 11

Si on l'exécute, une fenêtre console apparaît, expliquant que le driver est disponible sur le port 9515: 12

Essayons maintenant de communiquer avec notre driver en lui envoyant une requête http POST via Postman :

Url cible : http://localhost:9515/session ;

Body : { "desiredCapabilities" : {} } . 13

L'envoi de notre requête http a plusieurs effets :

Tout d'abord, elle ouvre Chrome 14

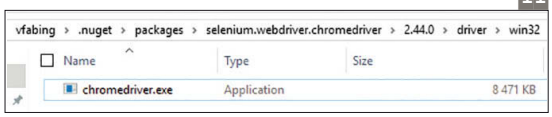
Mais surtout, elle nous renvoie tout un tas d'informations, notamment le sessionId qui peut être réutilisé dans les appels http POST ultérieurs pour piloter Chrome.

Exemple de navigation vers une url : 15

Nous avons vu le fonctionnement de l'architecture mise en place par Selenium, connaissance qui n'est évidemment pas obligatoire, mais toujours intéressante à savoir pour pouvoir être plus pertinent en cas d'erreur sur les webdrivers.

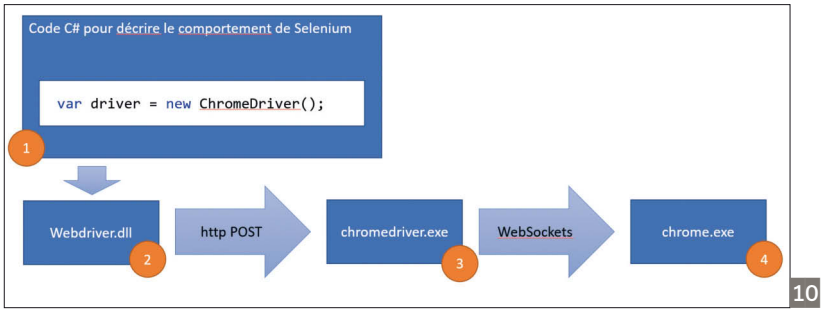
Il faut savoir que, comme les navigateurs évoluent très vite, ces dri-

```
private IWebDriver _driver;

[TestInitialize]
public void Initialize()
{
    _driver = new ChromeDriver();
}

[TestCleanup]
public void Cleanup()
{
    _driver.Quit();
}
```
8



12

```
// Arrange
var expected = "Results";
var driver = new ChromeDriver(".");

// Act
driver.Navigate().GoToUrl("https://bing.com");
driver.FindElement(By.Id("sb_form_q")).SendKeys("Hello World");
driver.FindElement(By.Id("sb_form_go")).Click();
var actual = driver.FindElement(By.ClassName("sb_count")).Text;

// Assert
Assert.Contains(expected, actual);
driver.Quit();
```
9



10

vers sont également amenés à monter de version très fréquemment (erreur simple de problème de communication entre le webdriver et le navigateur). Heureusement pour nous, les packages NuGet contenant ces drivers facilitent grandement cette maintenance, et là où il était nécessaire de télécharger les nouveaux webdrivers pour chaque nouvelle version, il suffit maintenant de mettre à jour le package de NuGet.

Comment organiser ses tests Selenium pour qu'ils restent maintenables ? Welcome Page Object Pattern

Créer un test d'interface graphique automatisé avec Selenium est vraiment très simple. Vous pouvez vous appuyer sur l'extension **Katalon Recorder** si vous êtes encore néophyte en programmation, ou attaquer directement avec **Selenium.WebDriver** si vous êtes plus à l'aise avec les lignes de code. Cependant dans tous les cas, les tests sont décrits via des lignes de code, et comme toute base de code grossissante, des problèmes de maintenabilité de la base de code vont faire leur apparition.

Comme toujours, ajouter de nouvelles couches d'abstractions est généralement une bonne pratique. Et avec les tests d'interface graphique, il existe un pattern très connu appelé le pattern **Page Object** ou encore **Page Object Model (POM)** décrivant la mise en place d'une couche d'abstraction avec les tests UI. L'utilisation de ce pattern permet, à terme, de corriger facilement plusieurs centaines de tests en échec, juste en changeant quelques lignes de code. Cela permet aussi de capitaliser sur les améliorations développées pour interagir avec l'UI à travers tous les tests.

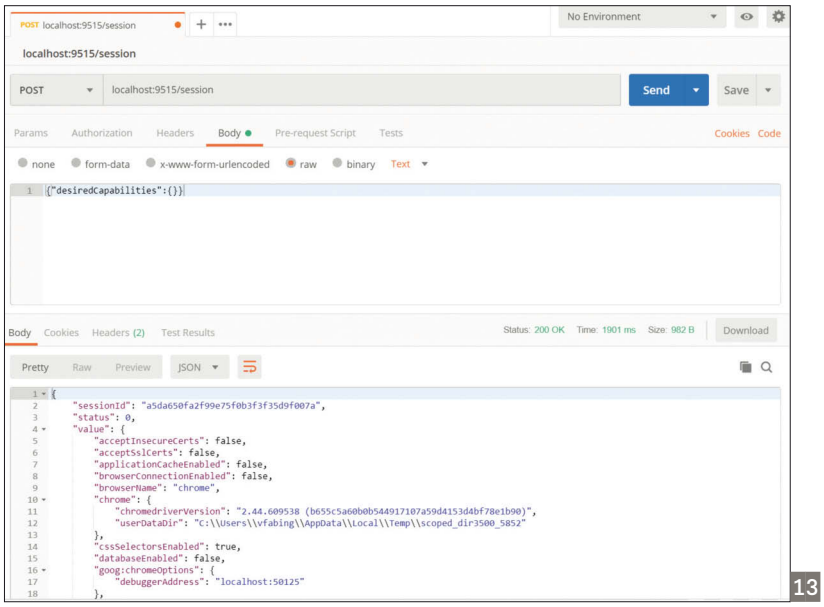
D'un point de vue conceptuel, dans votre scénario de test, à la place de manipuler un navigateur web en utilisant directement l'API Selenium, on va préférer passer par une couche intermédiaire appelée une **Page**. Celle-ci est responsable de l'appel des bonnes instructions Selenium afin de faire une action (cliquer sur un bouton, saisir du texte, etc.). Par la suite, dans votre scénario de test, vous n'aurez plus qu'à faire appel aux méthodes de manipulation exposées par cette **Page**. Et si un jour, par malchance, une méthode largement utilisée, telle qu'une méthode d'authentification, venait à être complexifiée. Ceci, par exemple, par l'ajout de la saisie d'un captcha. Il suffirait alors de rajouter cette nouvelle étape dans la méthode d'authentification de votre Page pour qu'automatiquement tous les tests utilisant cette méthode effectuent cette nouvelle étape et soient corrigés.

Note : *un projet d'exemple est disponible sur mon repository Github sample-selenium-dotnet (https://github.com/vfabing/sample-selenium-dotnet)*

Le pattern Page Object en pratique

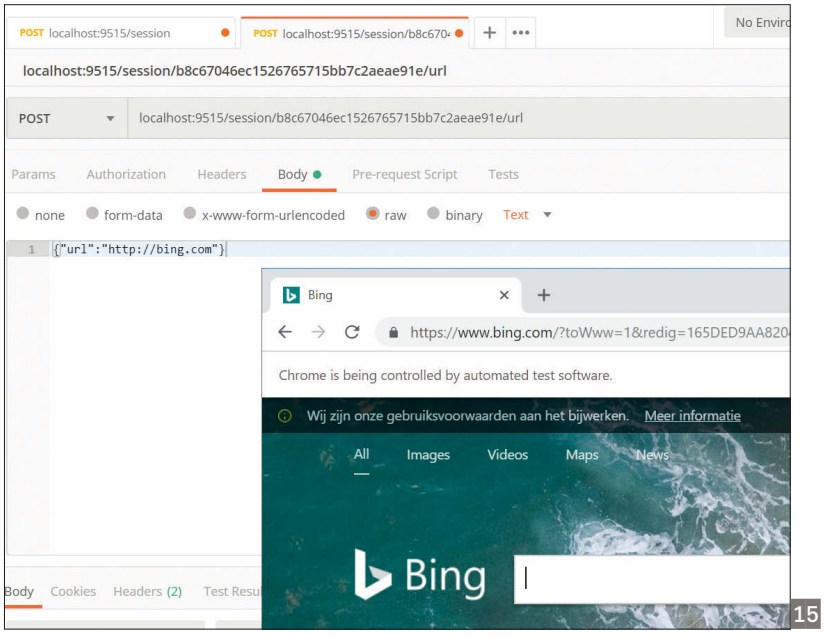
Disclaimer : il existe tout un framework de mise en place du pattern Page Object contenu dans les packages Selenium.Support. Cependant, ce framework n'est pas utilisé dans les exemples qui vont suivre, pour la bonne raison que le pattern est suffisamment simple pour être implémenté manuellement, ce qui permet une meilleure compréhension des principes, une meilleure maintenabilité, et une plus grande facilité à le déboguer.

La première étape à effectuer est la création d'une class de Page (**BingHomePage** par exemple), qui prendra en constructeur un **IWebDriver** :



13

14



15

```
public class BingHomePage
{
    private IWebDriver _driver;

    public BingHomePage(IWebDriver driver)
    {
        _driver = driver;
    }
}
```

Ensuite, comme nous ne voulons pas exposer les **IWebElement**, on les déclare en **private**. On expose ensuite une ou plusieurs

méthodes **public** pour pouvoir interagir indirectement avec des champs **private**.

Note : *dans l'exemple suivant, j'ai également utilisé une propriété* ***private*** *intermédiaire afin de ne charger le* ***IWebElement*** *qu'une seule fois.*

```
private IWebElement _searchTextBox;
private IWebElement SearchTextBox => _searchTextBox ??
_driver.FindElement(By.Id("sb_form_q"));

private IWebElement _searchButton;
private IWebElement SearchButton => _searchButton ??
_driver.FindElement(By.Id("sb_form_go"));

public void SearchFor(string text)
{
  SearchTextBox.SendKeys(text);
  SearchButton.Click();
}
```

Ce qui donne une fois tout rassemblé :

```
public class BingHomePage
{
  private IWebDriver _driver;

  public BingHomePage(IWebDriver driver)
  {
    _driver = driver;
  }

  private IWebElement _searchTextBox;
  private IWebElement SearchTextBox => _searchTextBox ??
_driver.FindElement(By.Id("sb_form_q"));

  private IWebElement _searchButton;
  private IWebElement SearchButton => _searchButton ??
_driver.FindElement(By.Id("sb_form_go"));

  public void SearchFor(string text)
  {
    SearchTextBox.SendKeys(text);
    SearchButton.Click();
  }
```

Plutôt concis je trouve. Cela permet de savoir où chercher en cas de problème avec la page d'accueil de Bing.

Utilisation des pages dans un scenario de test

Ensuite, si l'on retourne à notre scénario de test, on a maintenant besoin d'instancier les pages et de leur passer un **IWebDriver** en paramètre.

```
 TestMethod
public void BingSearch_ShouldReturnResults()
{
  var driver = new ChromeDriver();
  driver.Navigate().GoToUrl("https://bing.com");

  var bingHomePage = new BingHomePage(driver);
  bingHomePage.SearchFor("Hello World");

  var bingSearchResultsPage = new BingSearchResultsPage(driver);
  var numberOfResults = bingSearchResultsPage.GetNumberOfResults();

  Assert.IsTrue(numberOfResults > 0);
}
```

Plus aucune notion de Web Element n'est visible dans le scénario de test. Ceci signifie que l'on peut fixer le comportement de la recherche, ou de la récupération des résultats, simplement en modifiant la méthode SearchFor() ou GetNumberOfResults() des classes BingHomePage ou BingSearchResultsPage. Ensuite on voit tous ses tests utilisant ces méthodes automatiquement corrigés.

Au-delà du pattern Page Object

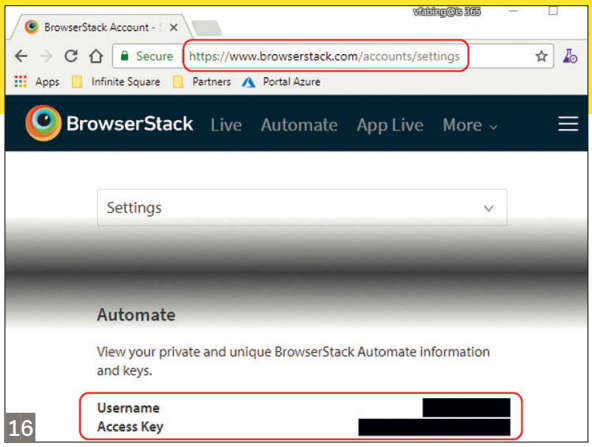
Comme nous avons pu le voir, implémenter le pattern de Page Object est très facile avec Selenium.NET. Et ce pattern pourtant très simple devrait vous épargner des heures de corrections de tests en échec . Bien entendu, on peut aller encore plus loin : on pourrait par exemple imaginer rajouter une couche d'abstraction supplémentaire qui serait responsable d'effectuer des actions / scénarios business, et qui serait en charge d'articuler ces actions en sollicitant les bonnes Pages nécessaires.
Un autre point d'amélioration important serait de supprimer les appels au mot clé new et d'utiliser des principes tels que l'injection de dépendance, ou le pattern Factory, afin de pouvoir gérer plus facilement les créations de ces pages.

Exécution multi-browser avec Selenium : Grid, BrowserStack, Sauce Labs

Dès lors que l'on parle d'automatiser des tests web, Selenium (*ou une surcouche basée dessus*) reste la solution la plus populaire à l'heure actuelle. Si elle permet bien d'automatiser facilement des comportements UI, sa mise en place peut se complexifier grandement lorsqu'il s'agit de garantir la compatibilité sur un grand nombre de navigateurs web (*que ce soit en vue « Desktop » ou « Mobile »*).
À moins de se constituer son propre parc de machines de test orchestré via Selenium Grid, une autre solution plus rapide à mettre en place consiste à s'appuyer sur des services Cloud tels que **BrowserStack** ou **Sauce Labs**.
Dans cet article, c'est la mise en place de BrowserStack qui est présentée.

Récupération de l'Access Key BrowserStack (prérequis)

Si vous ne possédez pas encore de compte BrowserStack, il est nécessaire de se créer un compte en Trial, qui offre 100 minutes d'exécution de tests automatisés multi-navigateurs.
Une fois votre compte créé, les informations importantes à récupérer sont les valeurs des paramètres **Username** et **Access Key** 16
Conservez précieusement ces informations à réutiliser dans les étapes suivantes.



16

Écrire le test Selenium et le configurer pour s'exécuter sur BrowserStack

Côté Visual Studio, la seule dépendance nécessaire est le **Selenium.WebDriver**, disponible sous forme de package NuGet 17

Le code de test nécessaire pour exécuter son test sur BrowserStack est lui aussi très minime. Il consiste à :

- Utiliser le **RemoteWebDriver**, qui permet d'exécuter les scripts de test sur un serveur Selenium distant, en pointant vers l'url http://hub-cloud.browserstack.com/wd/hub/ .
- S'authentifier en passant ses identifiants BrowserStack précédemment récupérés via la spécification des capacités du Driver :
  - « browserstack.user » correspond au **Username** BrowserStack ;
  - « browserstack.key » correspond à l'**AccessKey** BrowserStack ;

18

L'exécution et l'assertion du résultat du test sont effectuées de la même manière qu'un test Selenium classique (*à base de récupération d'éléments par leurs id, de saisies clavier, clic de souris et vérification du bon affichage d'éléments HTML*).

Note : *il est important de ne pas oublier d'appeler la méthode « Quit() » du WebDriver pour s'assurer que BrowserStack sache que la session de test est terminée (et éviter de consommer votre précieux crédit inutilement).*

Félicitations, vous venez d'automatiser votre premier test sur BrowserStack !

Visualiser les logs d'exécution sur BrowserStack

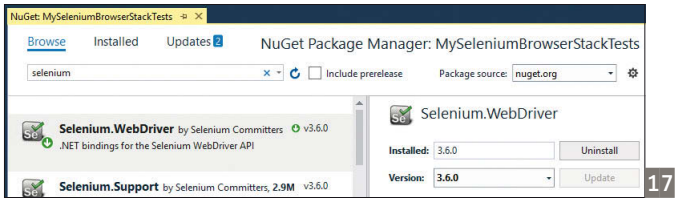
Pour vérifier les logs d'exécution, BrowserStack propose une interface web, accessible à l'url https://www.browserstack.com/automate (1).
On y trouve l'historique des exécutions (4), ainsi que divers logs (3) dont notamment l'enregistrement vidéo (2) 19

Pour exécuter ses tests sur un navigateur différent, la documentation de BrowserStack (https://www.browserstack.com/automate/c-sharp#setting-os-and-browser) permet de générer automatiquement les capacités à configurer sur le driver Selenium.
Exemple pour une exécution sur Chrome version 62, sur Windows 7 en résolution 1024 par 768 :

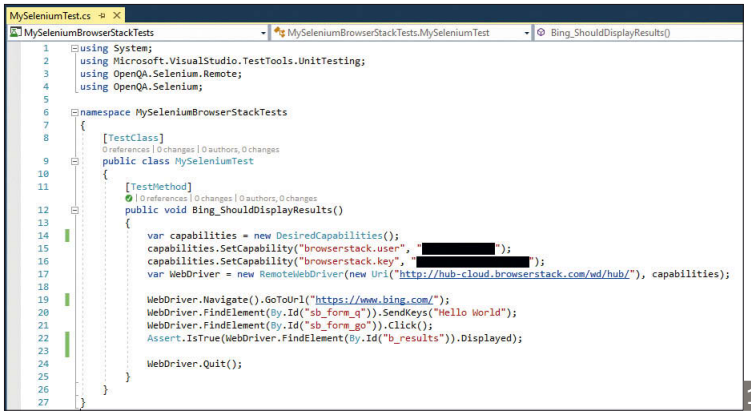
```
capabilities.SetCapability("browser", "Chrome");
capabilities.SetCapability("browser_version", "62.0");
capabilities.SetCapability("os", "Windows");
capabilities.SetCapability("os_version", "7");
capabilities.SetCapability("resolution", "1024x768");
```

Au-delà de la problématique multi-browser

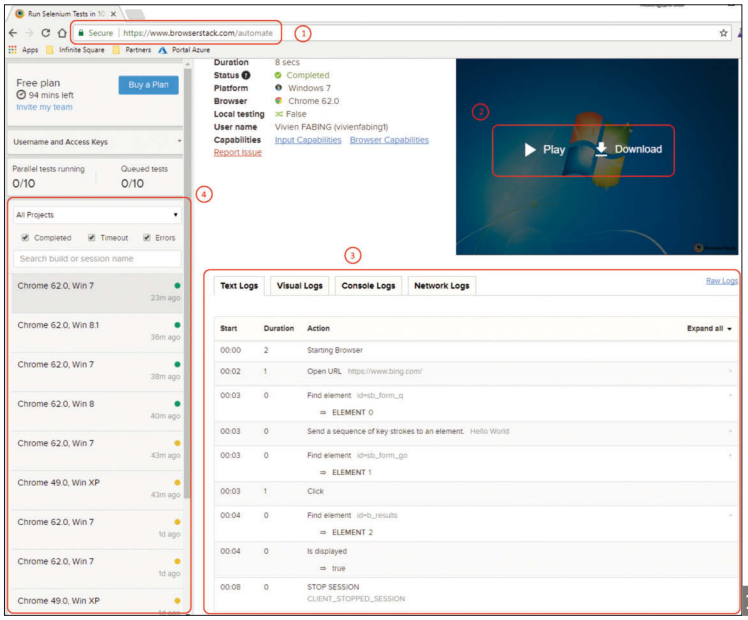
Nous l'avons vu, exécuter des tests d'interface web sur de multiples versions de navigateurs et s'assurer ainsi de la bonne compatibilité de ces derniers a un coût d'entrée techniquement faible.
Les réelles problématiques tournent plutôt sur l'environnement à mettre en place pour réaliser ces tests (un sous-ensemble de l'environnement, ou un environnement entier ?), ainsi que sur la mise en place de patterns améliorant la maintenabilité des tests tels que le pattern PageObject.



17



18

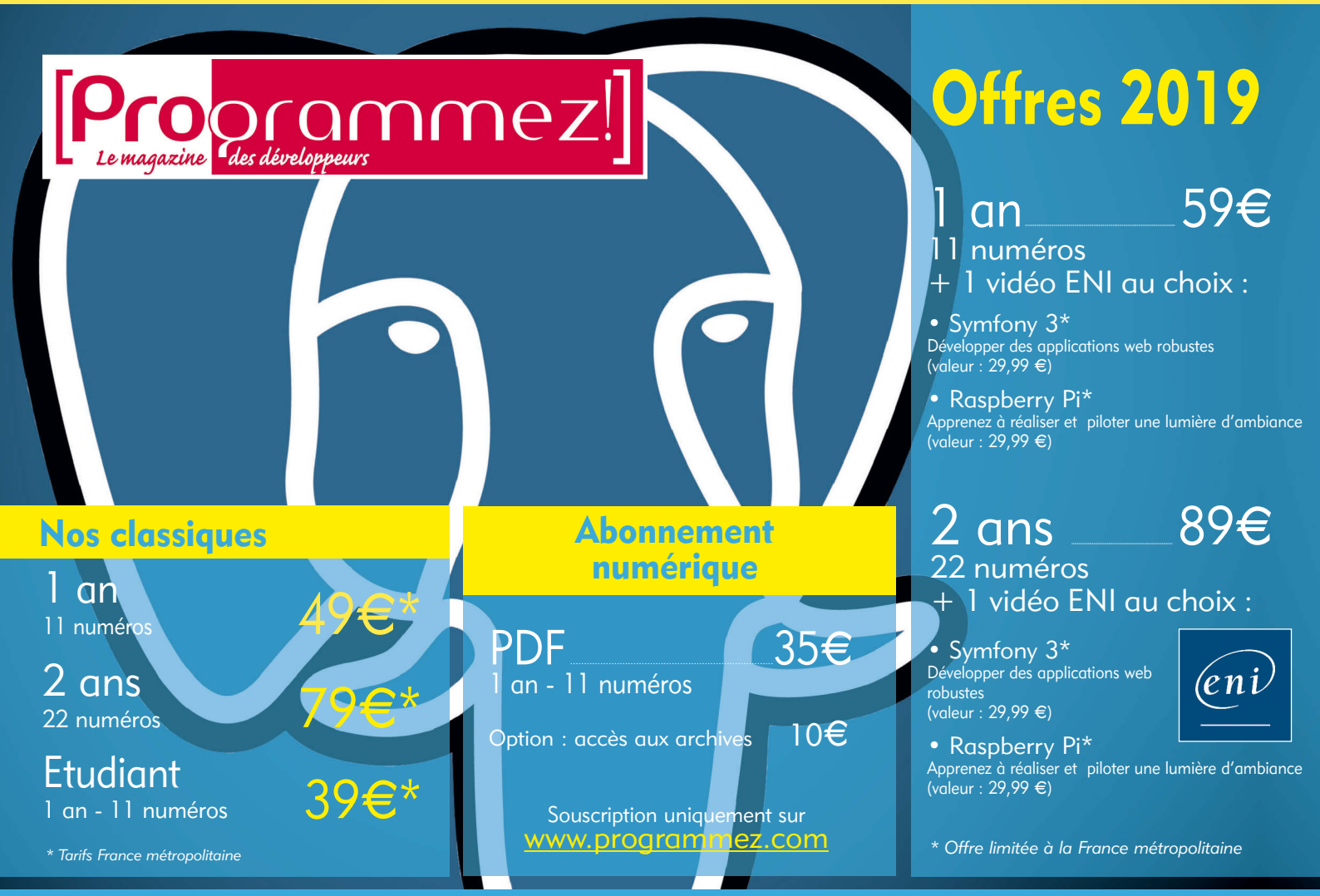


19

Pour aller plus loin

De nombreuses notions importantes concernant l'automatisation de tests avec **Selenium** ont pu être abordées durant ce dossier.
Pour aller plus loin, imaginons qu'au lieu d'avoir des Drivers pour piloter des navigateurs web, nous avions des Drivers pour piloter des applications Mobiles (iOS, Android) ou même Desktop (macOS, Windows) : c'est en partant de ce postulat de base que l'initiative **Appium** est née, et permet donc de piloter des applications desktop ou mobile, basées sur une architecture binding/driver similaire à Selenium . •

Abonnez-vous à Programmez! Abonnez-vous à Programmez! Abonne



Offres 2019

1 an ..... **59€**
11 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

2 ans ..... **89€**
22 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

** Offre limitée à la France métropolitaine*

Nos classiques

1 an — 11 numéros — 49€*

2 ans — 22 numéros — 79€*

Etudiant — 1 an - 11 numéros — 39€*

** Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

Option : accès aux archives 10€

Souscription uniquement sur
www.programmez.com

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ **Abonnement 1 an :** 49 €
☐ **Abonnement 2 ans :** 79 €
☐ **Abonnement 1 an Etudiant :** 39 €
Photocopie de la carte d'étudiant à joindre

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 228
Valable jusqu'au 3 mai 2019

☐ Mme ☐ M. Entreprise : ..... Fonction : .....

Prénom : ..... Nom : .....

Adresse : .....

Code postal : ..... Ville : .....

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : ..... @ .....

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

A DÉCOUVRIR D'URGENCE

NOUVEAU !

LE CADEAU GEEK IDÉAL

Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

☐ **Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007** :
9,99 € (+3 € de frais postaux) = 12,99 €

Commande à envoyer à :
Programmez!
57 rue de Gisors - 95300 Pontoise

PROG 228

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

commandez sur www.programmez.com

Christophe PICHAUD
Architecte Microsoft chez
'M Applications by Devoteam'
christophepichaud@hotmail.com
www.windowscpp.com

LMDB : la base NoSQL la plus rapide du monde

OpenLDAP-LMDB ou LMDB est une base de données embarquée transactionnelle de stockage clé-valeur disponible sous Linux. En 2009 le projet OpenLDAP est préoccupé par l'avenir de Berkeley DB passé dans le giron d'Oracle et il est décidé de faire un fork de Berkeley DB. Appelée MDB, la librairie devient LMDB. La librairie est utilisée massivement sous Linux.

niveau 200

Portage sous Windows

Avant de pouvoir l'utiliser sous forme de DLL Windows, il faut migrer le code Linux vers Windows. La librairie contient du code pour Windows mais le portage est incomplet. L'utilisation des noms de fichiers contient un / au lieu de \\ qui fait que cela ne marche que sous Linux… Une fois cette modification effectuée, il faut exporter les diverses fonctions pour obtenir une DLL.

LMDBWindowsDll

Le portage de la DLL sous Windows est disponible sur : https://github.com/ChristophePichaud/LMDBWindows/LMDBWindowsDll
En Debug, la DLL LMDBWindowsDllD64.dll fait 235 KB. En Release, elle en fait moins de 100 KB. C'est ultra-léger.

Utilisation de LMDB API

L'API est du C pur et dur. Il y a des structures à définir et à utiliser. Les fonctions sont préfixées par mdb_. L'utilisation du C est basique mais peut rebuter certaines personnes. Il faut passer des buffers, des longueurs, du C quoi ! Pour être élégant, il vaut mieux encapsuler l'API C en C++ et mettre à disposition des std::string.

```
class LMDBWRAPPER_API CLMDBWrapperEx
{
public:
    CLMDBWrapperEx() {}
    ~CLMDBWrapperEx();

private:
    MDB_env * env;
    MDB_dbi dbi;
    MDB_val key, data;
    MDB_txn *txn;
    MDB_cursor * cursor;

public:
    void Init(const std::wstring& db)
    void BeginTransaction();
    void CommitTransaction();
    void Set(const std::string& k, const std::string& v)
    bool Get(const std::string& k, std::string & value)
};
```

Cette version d'API est beaucoup plus simple à utiliser. On va déclarer un objet, faire appel à Init en y passant le nom de la base de données, et ensuite on peut faire Get/Set comme on veut…

```
std::string db = "mycache";

CLMDBWrapperEx we;
we.Init(db);
we.BeginTransaction();
std::string key = "key_v000";
std::string value = "value_v000";
we.Set(key, value);

std::string value2;
we.Get(key, value2);
we.CommitTransaction();
```

Comme vous pouvez le constater, l'ajout de données se fait par clé et valeur.

La base de données

Elle est constituée de deux fichiers :

- data.mdb
- lock.mdb

La méthode Init crée un répertoire du nom de la base dans c:\temp pour y stocker ces deux fichiers.

L'ouverture de la base

On vient de voir que Init crée un répertoire mais ce n'est pas tout :

```
void Init(const std::string& db)
{
    char sz[255];
    sprintf_s(sz, "%s\\%s", Constants::LMDBRootPath.c_str(), db.c_str());
    ::CreateDirectoryA(sz, NULL);

    mdb_env_create(&env);
    mdb_env_set_maxreaders(env, 1);
    mdb_env_set_mapsize(env, 10485760 * 100);
    mdb_env_open(env, sz, MDB_CREATE, 0);

    mdb_txn_begin(env, NULL, 0, &txn);
    mdb_dbi_open(txn, NULL, 0, &dbi);
    mdb_txn_commit(txn);
}
```

Init ouvre aussi une connexion à la base via mdb_dbi_open et positionne certains paramètres.

Gestion de la transaction

La transaction est créée via mdb_txn_begin et mdb_txn_commit :

```
void BeginTransaction()
{
    int err = 0;
    err = mdb_txn_begin(env, NULL, 0, &txn);
}

void CommitTransaction()
{
    int err = 0;
    err = mdb_txn_commit(txn);
}
```

Insertion de données

La création des données se fait via un appel à mdb_put :

```
void Set(const std::string& k, const std::string& v)
{
    int err = 0;

    key.mv_size = k.length() + 1;
    key.mv_data = (void *)k.c_str();
    data.mv_size = v.length() + 1;
    data.mv_data = (void *)v.c_str();
    err = mdb_put(txn, dbi, &key, &data, 0); // MDB_NOOVERWRITE);
    //printf("Set err:%d Key:%s Data:%s\n", err, key.mv_data, data.mv_data);
}
```

Récupération de données

La récupération des données se fait via un appel à mdb_get ;

```
bool Get(const std::string& k, std::string & value)
{
    int err = 0;

    key.mv_size = k.length() + 1;
    key.mv_data = (void *)k.c_str();

    err = mdb_get(txn, dbi, &key, &data);

    if (err != 0)
        return false;

    //printf("Get err:%d Key:%s Data:%s\n", err, key.mv_data, data.mv_data);
    value = (char *)(data.mv_data);

    return err == 0 ? true : false;
}
```

Evaluation des performances

Reprenons notre exemple et mettons-lui des indicateurs de temps :

```
    int count = 2;

    if (argc == 2)
    {
        count = atoi(argv[1]);
    }

    CLMDBWrapperEx wr1;
    wr1.Init(db);

    DWORD dwStart = 0;
    DWORD dwEnd = 0;

    dwStart = ::GetTickCount();
    wr1.BeginTransaction();
    for (int i = 0; i < count; i++)
    {
        std::string k = "key_v" + std::to_string(i);
        std::string v = "value_v" + std::to_string(i);
        wr1.Set(k, v);
    }
    wr1.CommitTransaction();
    dwEnd = ::GetTickCount();
    std::cout << "Elapsed Time for SET: "
<< dwEnd - dwStart << " ms" << std::endl;

    dwStart = ::GetTickCount();
    wr1.BeginTransaction();
    for (int i = 0; i < count; i++)
    {
        std::string k = "key_v" + std::to_string(i);
        std::string v;
        wr1.Get(k, v);
    }
    wr1.CommitTransaction();
    dwEnd = ::GetTickCount();
    std::cout << "Elapsed Time for GET: "
<< dwEnd - dwStart << " ms" << std::endl;
```

Le programme est simple. Il prend un nombre d'opérations à réaliser et ensuite il fait autant de GET et autant de SET sur un jeu de clé / valeur simple. Testons ce petit programme :

Opération	Nombre	Durée (ms)
get	1	0
set	1	0
get	10	0
set	10	0
get	100	15
set	100	0
get	1000	63
set	1000	15
get	10000	109
set	10000	63
get	100000	1141
set	100000	640

Modifions le programme pour insérer et récupérer des documents jSON :

```
std::string jsonDocument = R"("User": {
    "Account": "%s@agenda.com",
        "Password" : "agenda",
        "AdjustTime" : 1,
        "FiggoURL" : "https://wendelgroup.ilucca.net/",
        "FiggoToken" : "aaaaaaaa-fb8f-41a3-a08e-84b6521ec96a",
        "EnableFiggo" : false
    })";
```

```
dwStart = ::GetTickCount();
wr1.BeginTransaction();
for (int i = 0; i < count; i++)
{
    char sz[1024];
    std::string k = "key_j" + std::to_string(i);
    sprintf_s(sz, jsonDocument.c_str(),
std::to_string(i).c_str());
    std::string v = sz;
    wr1.Set(k, v);
}
wr1.CommitTransaction();
dwEnd = ::GetTickCount();
std::cout << "Elapsed Time for SET jSON: "
<< dwEnd - dwStart << " ms" << std::endl;

dwStart = ::GetTickCount();
wr1.BeginTransaction();
for (int i = 0; i < count; i++)
{
    std::string k = "key_v" + std::to_string(i);
    std::string v;
    wr1.Get(k, v);
}
wr1.CommitTransaction();
dwEnd = ::GetTickCount();
std::cout << "Elapsed Time for GET jSON: "
<< dwEnd - dwStart << " ms" << std::endl;
```

Voici le résultat :

Opération	Nombre	Durée (ms)
get jSON	1	0
set jSON	1	0
get jSON	10	0
set jSON	10	0
get jSON	100	0
set jSON	100	0
get jSON	1000	16
set jSON	1000	16
get jSON	10000	218
set jSON	10000	63
get jSON	100000	2375
set jSON	100000	672

Les performances sont exceptionnelles. La taille de la base de données est conséquente (53 MB) :

```
D:\Dev\LMDBWindows\LMDBWindows\x64\Debug>dir c:\temp\mycache
Directory of c:\temp\mycache
02/10/2019  05:11 AM    <DIR>          .
02/10/2019  05:11 AM    <DIR>          ..
02/10/2019  05:11 AM        53,145,600 data.mdb
02/10/2019  05:11 AM               128 lock.mdb
```

Application pratique

Vous souhaitez utiliser une base de données de cache dans votre site web sur Azure et vous hésitez entre DocumentDB ou MongoDB. Arrêtez de payer ce genre de services, essayez LMDB, c'est gratuit et beaucoup plus rapide. De plus LMDB possède aussi des fonctionnalités de curseur pour lire le contenu de la base…

Explorer la base

Voici le code qui permet de parcourir les données :

```
void BeginTransactionReadOnly()
{
    int err = 0;

    err = mdb_txn_begin(env, NULL, MDB_RDONLY, &txn);
}


void OpenCursor()
{
    int err;
    err = mdb_cursor_open(txn, dbi, &cursor);
}

void CloseCursor()
{
    mdb_cursor_close(cursor);
}


bool GetFromCursor(std::string & k, std::string & v)
{
    int err = 0;

    err = mdb_cursor_get(cursor, &key, &data, MDB_NEXT);

    if (err != 0)
        return false;

    //printf("Get err:%d Key:%s Data:%s\n",
err, key.mv_data, data.mv_data);

    k = (char *)(key.mv_data);
    v = (char *)(data.mv_data);

    return err == 0 ? true : false;
}

void GetAllData()
{
    int err;
    BeginTransactionReadOnly();

    OpenCursor();

    std::string k, value;
    while (GetFromCursor(k, value))
    {
        printf("key: '%s', value: '%s'\n",
k.c_str(), value.c_str());
```

```
    }

    CloseCursor();

    AbortTransaction();
  }
```

Le code de GetAllData() n'est pas très compliqué. On déclare un curseur et on itère sur la méthode GetFromCursor. Voici le programme client :

```
CLMDBWrapperEx wr1;
wr1.Init(db);
wr1.GetAllData();
```

Voici la sortie console, on y remarque les documents jSON : 1
Si on modifie le programme client pour avoir des indications de temps :

```
Elapsed Time for GetAllData: 32 ms
```

Il faut moins de 100ms pour charger 200.000 items soit 53 MB de data en RAM.
C'est du C/C++ ! C'est performant. Ultra-performant.

Interop en C# .NET

Il existe des bindings pour le langage C#. C'est le rôle de LMDB.NET. Un portage C# qui fait des appels Interop à la DLL qui va bien. Voici le diagramme de classes simplifié : 2
Voici comment ouvrir une base :

```
string dir = "c:\\temp\\mycache";
LMDBEnvironment _env = new LMDBEnvironment(dir);
_env.MaxDatabases = 2;
_env.MapSize = 10485760 * 100;
_env.Open();
```

Voici comment faire un Put :

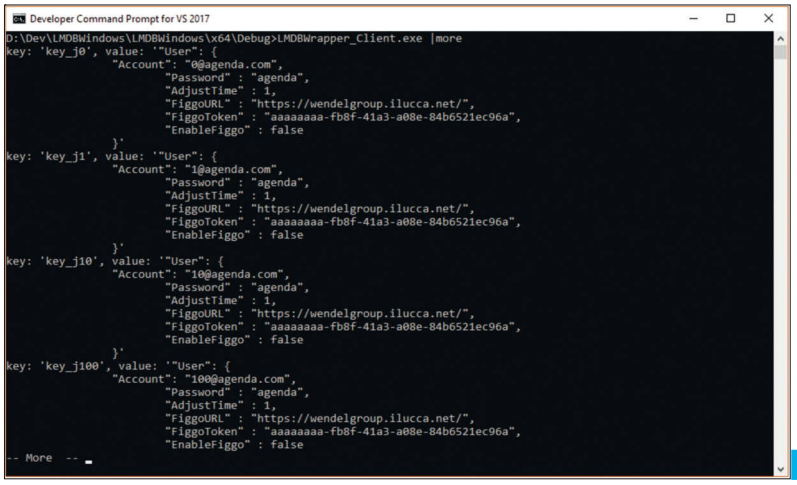
```
var tx = _env.BeginTransaction();
var db = tx.OpenDatabase(null,
new DatabaseConfiguration { Flags = DatabaseOpenFlags.Create });
tx.Put(db, "hello", "world");
tx.Commit();
```

Voici comment faire un Get :

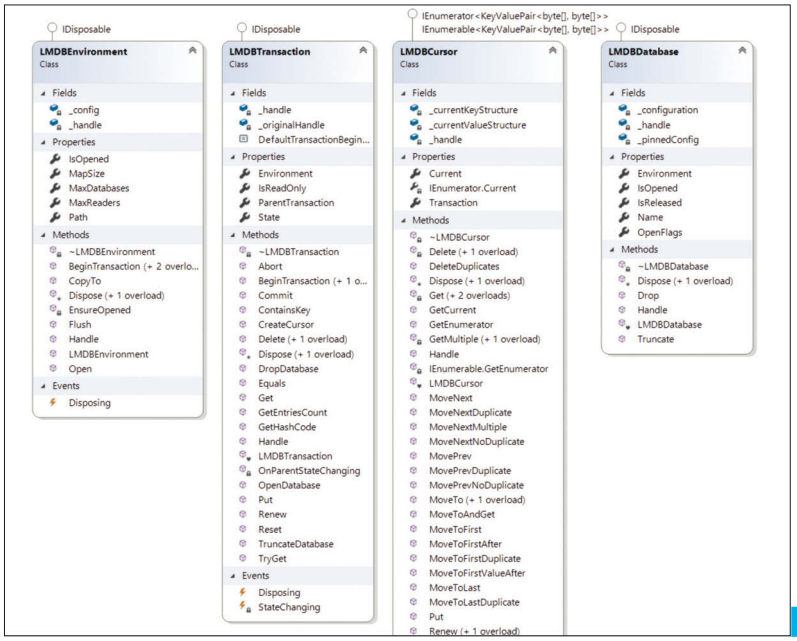
```
tx = _env.BeginTransaction(TransactionBeginFlags.ReadOnly);
db = tx.OpenDatabase(null);
var result = tx.Get(db, "hello");
tx.Commit();
db.Dispose();
```

Voici le code de récupération des informations jSON via le cursor :

```
  LMDBTransaction tx3 = _env.BeginTransaction(TransactionBeginFlags.NoSync);
  LMDBDatabase db3 = tx3.OpenDatabase(null,
new DatabaseConfiguration { Flags = DatabaseOpenFlags.Create });
  var cur = tx3.CreateCursor(db3);
  while (cur.MoveNext())
  {
    var fKey = Encoding.UTF8.GetString(cur.Current.Key);
```



1



2

```
  var fValue = Encoding.UTF8.GetString(cur.Current.Value);
  string str3 = String.Format("key:{0} => value:{1}", fKey, fValue);
  Logger.LogInfo(str3);
}
tx3.Commit();
db3.Dispose();

_env.Dispose();
```

Conclusion

Que vous souhaitiez utiliser SQLite ou un fichier de config, utiliser une base Azure comme Document DB ou MongoDB, il faut maintenant prendre en compte cette nouvelle librairie qu'est LMDB. C'est gratuit, rapide et facile à mettre en œuvre. Elle explose les performances de toutes les bases NoSQL. Vous pouvez l'utiliser dans un container Docker avec la mise à disposition des données, soit volatiles dans le container, soit persistantes dans des volumes partagés. LMDB et LMDB.NET, c'est le meilleur des deux mondes, le C++ pour la rapidité et le C# pour la facilité d'utilisation. Lancez-vous ! •

Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.frauteur

La promesse des promises en JavaScript

Partie 1

L'implémentation de la spécification ECMAScript 2015 sur laquelle JavaScript est fondé a conduit à introduire un nouvel objet standard en JavaScript : Promise. Cet objet permet de faciliter et de sécuriser l'écriture d'un appel à une fonction asynchrone. Son apparition souligne l'importance prise par l'asynchronisme en JavaScript.

niveau 200

L'objet `Promise` est bien d'un objet standard, c'est-à-dire propre au langage comme String et non au navigateur, comme `XMLHttpRequest`. Il permet de faciliter et de sécuriser l'écriture d'un appel à une fonction asynchrone, c'est-à-dire à une fonction qui rend la main sans avoir encore retourné son résultat, s'engageant à signaler au programme qui l'a appelée quand ce résultat sera disponible, par exemple en appelant une fonction que le programme lui a fournie – une callback.

Tout développeur JavaScript a très probablement déjà utilisé ce type de fonction, notamment pour charger des fichiers via l'objet `XMLHttpRequest`, ou pour déclencher une action après expiration d'un timer programmé via `window.setTimeout ()`.

Avec l'objet `Promise`, l'asynchronisme a en quelque sorte été dégagé des objets qui viennent d'être cités pour accéder au rang d'aspect fonctionnel fondamental de JavaScript, au même titre que l'héritage basé sur les prototypes ou les closures. Partant, tout développeur JavaScript doit maîtriser l'asynchronisme.

Pour autant, qui met le nez dans la spécification ne peut être que rapidement rebuté par l'opacité de cette dernière. Fort heureusement, il existe de nombreuses autres sources auxquelles se référer. Apportons ici une modeste contribution avec une présentation détaillée du système des promises destinée à qui découvre le sujet.

Avertissement

Le code figurant dans cet article emploie intensivement des fonctions fléchées. Pour rappel :

```
// Accolades si plusieurs instructions
var f = (i) => {
        i + 1;
        return (i);
    };
// Appel classique par la suite
f (42);

// Pas d'accolades si une instruction qui constitue la valeur retournée
var f = (i) => i ++;

// Pas de parenthèses si un seul paramètre
var f = i => i ++;

// IIFE (retourne 42 immédiatement)
(i => { i ++; return (i); }) (41)

// Idem
(i => i ++) (41)
```

Une fonction fléchée n'a pas de `this` : toute référence à `this` dans son corps se traduit par une recherche d'une variable nommée `this` dans son contexte local, à défaut dans un de ses contextes englobants :

```
function Banner (message) {
    this.message = message;
    this.displayNormal = function () {
        // Dans la fonction de l'IIFE, this est l'objet window !
        (function f () { console.log (this.message); }) ();
    }
    this.displayArrow = function () {
        // Dans la fonction de l'IIFE, this est l'objet Banner
        (() => console.log (this.message)) ();
    }
}
// Affiche undefined car window n'a pas de propriété .message
new Banner ("Normal function").displayNormal ();
// Affiche "Arrow function"
new Banner ("Arrow function").displayArrow ();
```

Au début, je n'étais pas grand fan des fonctions fléchées : une nouvelle syntaxe pour les fonctions, qui va venir tout compliquer ? Toutefois, l'essayer, c'est l'adopter. A l'usage, l'écriture des fonctions est grandement facilitée, même si la lecture peut s'avérer un peu plus délicate, du moins le temps de s'habituer. Par ailleurs, on constate que – comme le dit Douglas Crockford en prétendant qu'il ne faut pas tout utiliser dans JavaScript, mais seulement ses good parts, dont `this` n'est pas – qu'on peut se passer de `this` dans une fonction de base.

Le code emploie aussi intensivement les closures. Pour rappel, une closure est un ensemble de variables qui restent accessibles dans une fonction alors que ces variables ne figurent pas dans le contexte local ni dans le contexte global de cette dernière (ce ne sont pas des arguments, ni des variables locales, ni des variables globales) :

```
function f (something) {
    return (function () {
        console.log (something);
    });
};
var g = f ("Saved in closure!");
// Affiche "Saved in closure!"
g ();
```

Au commencement : les fonctions asynchrones

Comme l'écrit Kyle Simpson au tout début de son excellent ouvrage *You dont' know JS: Async & Performance* :

One of the most important and yet often misunderstood parts of programming in a language like JavaScript is how to express and manipulate program behavior spread out over a period of time.
Cela tient au fait que pour un développeur, il n'apparaît pas d'emblée intéressant de savoir comment un programme JavaScript est exécuté, c'est-à-dire comment le runtime JS – le moteur JavaScript – fonctionne. Pourtant, il est indispensable d'en savoir plus sur le sujet avant d'aborder les promises.
Le runtime JS est un programme qui exécute du code à la demande. Il est appelé par l'event loop – boucle infinie de gestion des évènements – du navigateur, en premier lieu pour exécuter le programme contenu dans une balise `<script>` rencontrée durant l'interprétation du contenu d'une page Web, mais encore pour exécuter une callback à l'expiration d'un timer créé par `window.setTimeout ()`, ou pour exécuter un gestionnaire d'évènement ajouté par `addEventListener ()` lorsque l'utilisateur entreprend l'action correspondante.
Appelons « task » le code que le runtime JS exécute à un instant donné.
Le principe de « run-to-completion » impose le runtime JS n'est jamais interrompu durant l'exécution d'une task.
Ajoutons à cela qu'à de très rares exceptions, un appel de fonction est toujours synchrone, ce qui signifie que la fonction appelée est exécutée immédiatement, avant la fin de la task. Ce sont des fonctions des API qui permettent des appels asynchrones, comme `window.setTimeout ()` ou `XMLHttpRequest.send ()` : ces appels entraînent plus tard la création des évènements adéquats pour exécuter une callback.
En JavaScript, jusqu'à ECMAScript 2017, il n'existait pas de possibilité de créer une fonction asynchrone en dehors de ces appels à des fonctions des API. Ou alors, il fallait détourner de telles fonctions pour n'utiliser que leur capacité à faire ajouter un évènement dans l'event loop :

```
function makeAsyncCallTo (f) {
    // Revient à window.setTimeout (f, 0)
    window.setTimeout (f);
}
var flag = false;
makeAsyncCallTo (() => console.log (`1st async call: ${flag}`));
makeAsyncCallTo (() => console.log (`2nd async call: ${flag}`));
flag = true;
1st async call: true
2nd async call: true
```

Le fait que la valeur affichée soit `true` alors que flag a été passé à true APRES les appels aux fonctions montre bien que ces dernières ont été exécutées après la fin de la task, que rien n'est venu interrompre en application du principe de run-to-completion, même pas l'expiration d'un délai implicitement fixé à `0`.
A l'occasion de l'introduction de l'objet Promise dans ECMAScript 2015, une nouvelle notion est apparue : celle de job queue. Ici, il faut clarifier le vocabulaire. Ce que les auteurs de la spécification ECMAScript 2015 appellent « job » en faisant abstraction du contexte dans lequel fonctionne le runtime JS, les développeurs Web l'appellent « microtask » en tenant compte du contexte du navigateur Web où ce runtime JS fonctionne.
Une microtask est une task qui doit être exécutée aussitôt que possible par l'event loop, c'est-à-dire avant la task suivante s'il s'en trouve. La possibilité pour une task de créer une microtask, c'est ce qui fait dire à Kyle Simpson :
It's kinda like saying, "oh, here's this other thing I need to do later, but make sure it happens right away before anything else can happen." The event loop queue is like an amusement park ride: once you finish the ride, you have to go to the back of the line to ride again. But the Job queue is like finishing the ride, cutting in line, and getting right back on.
Cela ne signifie pas que les microtasks s'insèrent dans la file des tasks. En fait, elles s'insèrent dans une file distincte. Ce qui fait de plus dire à Kyle Simpson que :
Jobs are kind of like the spirit of the setTimeout(..0) hack, but implemented in such a way as to have a much more well-defined and guaranteed ordering: later, but as soon as possible.
Dans un navigateur, les règles sont donc que :
- une task n'est exécutée qu'après une autre (principe de « run-to-completion ») ;
- une microtask créée par une task est exécutée après cette task, avant la task suivante ;
- les microtasks sont exécutées les unes après les autres.

Au-delà, la manière dont le navigateur peut ou non intercaler des tasks pour assurer des fonctions essentielles est hors de propos ici. Ces règles impliquent que toutes les microtasks créées par une tasks sont exécutées avant la task suivante.
Pour terminer sur cette présentation de l'asynchronisme en JavaScript, rajoutons un point qui fera sens plus tard, car il fait référence au système des promises.
Pour l'heure, les microtasks sont réservées au système des promises. Ainsi, `window.setTimeout ()` avec un délai nul ne permet que de rajouter une task, pas une microtask.
L'exécution des fulfillers d'une promise est programmée sous la forme de microtasks. Par exemple… :

```
console.log ("Start");
window.setTimeout (() => console.log ("Timeout"));
var p = new Promise ((resolve, reject) => resolve ());
p.then (result => console.log ("Fulfiller"));
console.log ("End");
```

…produit :

```
Start
End
Fulfiller
Timeout
```

La promesse des promises

Soit une API exposant plusieurs fonctions asynchrones, qui seront pour l'occasion simulées par des timers :
- `asyncGetSomeDate ()` doit retourner un tableau de valeurs ;
- `asyncComputeSomething ()` doit en calculer la somme.

Et dans le programme principal, une fonction synchrone `syncDisplaySomeResult ()` affiche la somme.

Par défaut, il est possible d'écrire cela :

```
// API
function asyncGetSomeData (n, callback) {
    console.log ("Getting some data...");
    window.setTimeout (() => {
        var i, numbers;
        numbers = new Array ();
        for (i = 0; i != n; i ++)
            numbers.push (i);
        callback (numbers);
    }, 2000);
}

function asyncComputeSomething (numbers, callback) {
    console.log ("Computing something...");
    window.setTimeout (() => {
        var i, s;
        s = 0;
        for (i = 0; i != numbers.length; i ++)
            s += numbers[i];
        callback (s);
    }, 2000);
}

function syncDisplaySomeResult (sum) {
    console.log (`And the result is: ${sum}`);
}

// Utilisation de l'API
asyncGetSomeData (10, (numbers) => {
    asyncComputeSomething (numbers, (sum) => {
        syncDisplaySomeResult (sum);
    })
});
```

Le système des promises permet d'écrire cela :

```
// API
function asyncGetSomeData (n) {
    return (new Promise ((resolve, reject) => {
        console.log ("Getting some data...");
        window.setTimeout (() => {
            var i, numbers;
            numbers = new Array ();
            for (i = 0; i != n; i ++)
                numbers.push (i);
            resolve (numbers);
        }, 2000);
    }));
}

function asyncComputeSomething (numbers) {
    return (new Promise ((resolve, reject) => {
        console.log ("Computing something...");
        window.setTimeout (() => {
            var i, s;
            s = 0;
            for (i = 0; i != numbers.length; i ++)
                s += numbers[i];
            resolve (s);
        }, 2000);
    }));
}

function syncDisplaySomeResult (sum) {
    console.log (`And the result is: ${sum}`);
}

// Utilisation de l'API
asyncGetSomeData (10).then (asyncComputeSomething).then (syncDisplaySomeResult);
```

Comme il est possible de le constater, en faisant disparaître toute mention à des callbacks le système des promises permet de simplifier non seulement l'écriture du code utilisant l'API, mais aussi celle du code de l'API elle-même, quoique ce soit dans une moindre mesure. La promesse des promises, c'est donc d'échapper au « callback hell » en permettant d'écrire une composition d'appels synchrones ou asynchrones sous une forme très proche de son écriture formelle qui serait, dans le cas précédent :

```
syncDisplaySomeResult (asyncComputeSomething (asyncGetSomeData (10)))
```

En fait, il y a plus à dire sur les apports du système des promises sur le système des callbacks. Quelque chose de plus essentiel même, qui permet de mieux comprendre que le système des promises soit appelé ainsi.

Comme Kyle Simpson l'explique dans le chapitre *Callbacks* de son ouvrage, appeler une fonction asynchrone d'un tiers qui devra en retour appeler une callback, c'est implicitement attendre que la callback soit appelée une fois et une seule, dans certaines circonstances. Bref, c'est passer un contrat implicite avec ce tiers à l'occasion d'une inversion de contrôle – on passe le contrôle de l'exécution à un tiers, en attendant qu'il le rende.

Or la découverte des termes de contrat peut s'effectuer aux dépens. Kyle Simpson prend l'exemple d'un site marchand qui appellerait une fonction d'un service de paiement, lequel devrait en retour appeler une callback du site marchand quand le paiement a été effectué. Si les développeurs du site marchand s'amusent à faire un test appelant plusieurs fois la callback avant le paiement, et que cela n'a pas été anticipé par les développeurs du site : problème! Ce qui fait écrire à l'auteur :

You're probably slowly starting to realize that you're going to have to invent an awful lot of ad hoc logic in each and every single callback that's passed to a utility you're not positive you can trust. Now you realize a bit more completely just how hellish « callback hell » is.

Le système des promises apporte une solution. Une promise ce n'est pas seulement, comme on le verra, un objet qui signalera lorsqu'il disposera du résultat retourné par une fonction appelée de manière asynchrone. C'est un objet qui le signalera au bon moment, et qui le ne signalera qu'une fois.

Pour reprendre l'exemple précédent, dans une fonction telle que `asyncGetSomeData ()` , la promise retournée ne changera pas d'état tant que `resolve ()` n'aura pas été appelé. Et si par la suite la promi-

se est de nouveau utilisée, le code qui appelle `resolve()` ne sera pas exécuté de nouveau, car la promise ayant été résolue, elle conserve le résultat et y donne accès immédiatement. Ainsi… :

```
var p = asyncGetSomeData(10);
p.then(asyncComputeSomething).then(syncDisplaySomeResult);
p.then(asyncComputeSomething).then(syncDisplaySomeResult);
```

…va générer la sortie suivante :

```
Getting some data...
Computing something...
Computing something...
And the result is: 45
And the result is: 45
```

« Getting some data… » n'est bien affiché qu'une fois.

Comprendre les promises

Avant toute chose, il convient de préciser que le fonctionnement interne du système des promises est très complexe, et qu'il est parfaitement possible de l'utiliser sans maîtriser ce fonctionnement, pour la bonne raison que c'est un système qui vise à masquer la complexité. Comme on l'a vu, il ne vise qu'à simplifier l'écriture de compositions d'appels synchrones ou asynchrones, en permettant dans ce dernier cas d'échapper au « callback hell ».
Maintenant, qu'est-ce qu'une promise ? La définition donnée dans la spécification ECMAScript 2015 :
A Promise is an object that is used as a placeholder for the eventual results of a deferred (and possibly asynchronous) computation.
Comme toujours, rien n'est clair dans la spécification, et ce n'est que la première phrase ! En fait, pour comprendre le système des promises, le mieux est de procéder par reverse-engineering, c'est-à-dire de partir de l'usage qu'on en fait, en cherchant éventuellement à le réécrire par polyfill, comme le propose Axel Rauschmayer. Ce n'est qu'enfin qu'il convient de revenir sur la définition, car une fois que le système des promises est assimilé, elle se révèle effectivement particulièrement pertinente. D'ailleurs, quand il en vient à expliquer le fonctionnement interne des promises, Axel Rauschmayer écrit :
In this section, we will approach Promises from a different angle: Instead of learning how to use the API, we will look at a simple implementation of it. This different angle helped me greatly with making sense of Promises.
Avant même de considérer qu'une promise présente un intérêt pour gérer les appels asynchrones, il faut partir de la manière dont elle fonctionne dans le cas d'une fonction `f()` de base, c'est-à-dire synchrone.
Une promise est un objet créé par new sur une fonction désignée comme l'executor :

```
var p = new Promise(f);
```

La création de la promise sur `f()` entraîne immédiatement l'exécution de `f()`, qui reçoit en paramètre deux fonctions : `resolve()` et `reject()`. Chacune de ses fonctions prend un unique paramètre. Lors de son exécution, `f()` décide de les appeler pour témoigner au système – qui gère la promise – de sa réussite ou de son échec. A cette occasion `f()` fournit un résultat via `resolve()` ou une raison via `reject()` – on verra que ce résultat peut être rien, une valeur ou une promise :

```
function f(resolve, reject) {
    if ...
        resolve(result);
    else
        reject(reason);
}
```

Le code de `resolve()` et de `reject()` est inaccessible : ce sont des fonctions créées par le système.
La promise dispose notamment de méthodes : `.then()` et `.catch()`. Lorsqu'elle est appelée, `resolve()` appelle tous les fulfillers ajoutés par `.then()`. Pour sa part, lorsqu'elle est appelée, `reject()` appelle tous les rejecters ajoutés par `.then()` ou par `.catch()`, car `.catch(rejecter)` revient à `.then(undefined, rejecter)`. Si un fulfiller est appelé, il reçoit le résultat en paramètre ; et si un rejecter est appelé, il reçoit la raison en paramètre :

```
function onFulfilled(result) {}
function onRejected(reason) {}
p.then(onFulfilled, onRejected);
/*
// Ecriture équivalente :
p.then(onFulfilled);
p.catch(onRejected);
*/
```

Noter que dans la littérature, la signature de `.then()` est souvent notée `.then() (resolve, reject)`. Mais comme Kyle Simpson le souligne, cela introduit une confusion entre la méthode `Promise.resolve()` et le fulfiller. Il suggère de noter plutôt `.then(fulfilled, rejected)`. Retenons le point, mais notons plutôt ici `.then(fullfiller, rejecter)`.
Il est important de noter que l'appel aux fulfillers et au rejecters est asynchrone et non synchrone. Nous y reviendrons.
Il est possible d'ajouter plusieurs fulfillers et plusieurs rejecters. Ils sont appelés dans l'ordre dans lequel ils ont été ajoutés :

```
p = new Promise(f);
p.then(onFulfilled1, onrejecter1);
p.then(onFulfilled2);
p.catch(onReject2);
p.then(onFulfilled3, onReject3);
```

De combien de temps dispose-t-on à partir de la création de la promise pour ajouter des fulfillers ou des rejecters ? Cela n'a aucune importance, car par définition, un fulfiller doit être appelé après que `f()` a rapporté avoir réussi, et un rejecter après que `f()` a rapporté avoir échoué.
Noter que c'est ici qu'on comprend l'intérêt d'un promise, car le comportement qui vient d'être décrit se produit que `f()` soit synchrone ou asynchrone. Pour le démontrer on transforme généralement `f()` en fonction asynchrone en lui faisant appeler `resolve()` ou `reject()` après expiration d'un timer programmé par `window.setTimeout()`, qui rend la main immédiatement :

```
function f(resolve, reject) {
```

```
    window.setTimeout (function () {
        if ...
            resolve (result);
        else
            reject (reason);
    },
    4000);
}
```

Pour rajouter un intérêt supplémentaire, une API comportant des fonctions asynchrones les exposera sous la forme de fonctions qui retournent des promises. Autrement dit, ce n'est pas une fonction telle que `f()` qui ne sera pas exposée, mais une fonction retournant une promise créée sur `f()`, `f()` assurant le service à proprement parler, sur ce modèle :

```
function asyncF (someParameters) {
    return (new Promise ((resolve, reject) => {
        // Ici, du code exécuté de manière asynchrone
        // comme un appel à window.setTimeout ()
    }));
}
```

Par exemple, une fonction d'une API pour charger un fichier :

```
function asyncLoadTXT (url) {
    return (new Promise ((resolve, reject) => {
        var request;
        request = new XMLHttpRequest ();
        request.overrideMimeType ("text/plain");
        request.onreadystatechange = () => {
            if (request.readyState != 4)
                return;
            if (request.status != 200)
                reject (`HTTP error ${request.status} : ${request.statusText}`);
            else
                resolve (request.responseText);
        }
        request.open ("GET", url);
        request.send ();
    }));
}

// De la sorte, le client de l'API peut utiliser la fonction ainsi
asyncLoadTXT ("helloWorld.txt").then (result => console.log (result), reason => console.log (reason));
```

Mais comment le système peut-il savoir que `f()` a réussi ou échoué ? Tout bêtement car `f()` appelle `resolve()` ou `reject()` selon le cas, comme cela vient d'être dit. Le système ne devine rien. Ces fonctions lui permettent de conserver la trace de la réussite ou de l'échec de `f()`, c'est-à-dire tenir à jour l'état de la promise, avant d'appeler – de manière asynchrone – en conséquence les fulfillers ou les rejecters déjà ajoutés à ce stade.

C'est qu'une promise peut avoir trois états. A sa création, elle est dans l'état pending. Si `f()` appelle `resolve()`, elle passe dans l'état fulfilled. Et si `f()` appelle `reject()`, elle passe dans l'état rejected. Noter que malheureusement, l'objet Promise ne comprend pas de méthode ni de propriété pour connaître l'état d'une promise – toutefois, dans Firefox, un `console.log()` permet de visualiser cet état.

C'est pourquoi, lorsque `f()` appelle `resolve()` – même raisonnement pour `reject()` -, `resolve()` appelle – de manière asynchrone – tous les fulfillers ajoutés à ce stade. Si un fulfiller est ajouté par la suite, `f()` ayant signalé qu'elle a réussi, le fulfiller est immédiatement appelé – toujours de manière asynchrone. Il est possible de le démontrer en faisant de `f()` une fonction asynchrone :

```
function onFulfilled1 () { console.log (`onFulfilled1 () after: ${Date.now () - date} ms`); }
function onFulfilled2 () { console.log (`onFulfilled2 () after: ${Date.now () - date} ms`); }
function f (resolve, reject) {
    window.setTimeout (() => {
        resolve ();
    }, 2000);
}
// t = 0
var date = Date.now ();
var p = new Promise (f);
// onFulfilled1 () appelée au bout de t = 2 000 ms une fois que la promesse est fulfilled
p.then (onFulfilled1);
window.setTimeout (() => {
        // onFulfilled2 () appelée au bout de t = 4 000 ms car la promesse est déjà fulfilled quand
 ce fulfiller est ajouté
        p.then (onFulfilled2);
    },
    4000);
```

Des appels consécutifs à `resolve()` ou `reject()` ne produisent rien : une fois que `f()` a signalé avoir réussi ou échoué, ce résultat est définitif pour le système, et l'état de la promise ne peut donc être modifié :

```
function f (resolve, reject) {
    resolve ();
    // Aucun effet
    reject ();
}
var p = new Promise (f);
p.then (() => { console.log ("Success!") });
// Jamais exécutée
p.catch (() => { console.log ("Failure!") });
```

Il est maintenant temps de préciser ce point essentiel : un fulfiller ou un rejecter est appelé de manière asynchrone. Cela signifie qu'il est appelé après la fin de l'exécution du programme en cours.

Pour le démontrer, il faut faire preuve d'un peu de subtilité. En effet, le problème est que l'executor est exécuté à la création de la promise. Or un fulfiller ou un rejecter ne peut être ajouté qu'après la création de la promise. Dans ces conditions, comment montrer que lorsque l'executor appelle `resolve()`, `resolve()` n'exécute pas le fulfiller immédiatement, mais programme son exécution ?

Pour cela, il faut introduire une temporisation dans l'executor : du fait d'un timeout, ce dernier n'appellera `resolve()` qu'après avoir laissé le temps d'ajouter le fulfiller. Le fulfiller affiche la valeur d'un boolén `isFulfilled`, par défaut à false, mais passé à `true` après l'appel à `resolve()`. Si jamais `resolve()` exécutait le fulfiller de manière synchrone, ce dernier afficherait `false`. Or il affiche `true` :

```
var isFulfilled = false;
function f (resolve, reject) {
    window.setTimeout (function () {
            // resolve () appelé alors que isFulfilled est à false
            resolve ();
            // isFulfilled passé à true APRES l'appel à resolve ()
            isFulfilled = true;
        },
        // Donne le temps de rajouter un fulfiller tant que la promise est pending
        4000);
}
var p = new Promise (f);
// Pour afficher l'état de la promise, qui apparaît pending
console.log (p);
// A expiration du délai, le fulfiller affiche true !
p.then (result => console.log (isFulfilled));
```

Et si le fulfiller est ajouté alors que la promise est fulfilled ? Il est plus simple de montrer que le fulfiller est, ici encore, appelé de manière asynchrone :

```
var isFulfilled = false;
function f (resolve, reject) {
    resolve ();
}
var p = new Promise (f);
// Pour afficher l'état de la promise, qui apparaît fulfilled
console.log (p);
// A la fin de ce programme, le fulfiller affichera true !
p.then (result => console.log (isFulfilled));
// isFulfilled passé à true APRES l'appel à resolve ()
isFulfilled = true;
```

Bref, dans un programme ou un sous-programme qui crée une promise et qui y ajoute des fulfillers et des rejecters, il est certain que tous ces fulfillers et rejecters ne seront appelés qu'une fois l'exécution du programme terminée d'abord, et qu'une fois que la promise aura été résolue ou rejetée ensuite.
C'est ce qui rend complexe la réalisation d'un polyfill, car comment en JavaScript est-il possible d'appeler une fonction de manière asynchrone si elle n'est pas prévue pour cela ? Rien ne le permet par défaut dans le langage. Il faudrait détourner des fonctions asynchrones des API, ce qui semble affreux. Pourtant, c'est bien ce qu'on observe dans les polyfills, qui utilisent `windows.setTimeout ()` pour parvenir à leurs fins.

Un autre moyen pour créer une promise

Il est possible de créer une promise autrement qu'avec le constructeur `Promise ()`, en utilisant plutôt la méthode `.resolve ()` de l'objet `Promise` . Cette méthode permet de créer une promise non plus sur une fonction, mais sur une valeur, ou sur une promise, ou sur un thenable comme il en sera question plus loin :

valeur	Promise fulfilled dont le résultat est la valeur
promise	La promise elle-même, inchangée

Noter que dans le cas d'une valeur, les deux écritures suivantes sont donc équivalentes :

```
p = new Promise ((resolve, reject) => resolve (666));
p = Promise.resolve (666);
```

Et dans le cas d'un thenable, cet objet est donc supposé disposer d'une méthode `.then ()` prenant un fulfiller et un rejecter en paramètre. L'objet est alors transformé en promise :

```
var o = {
    then: (onFulfilled, onRejected) => onFulfilled (666)
};
var p = Promise.resolve (o);
// Affiche 666
p.then (result => console.log (result));
// Affiche true
console.log (p instanceof Promise);
```

Catch me if you catch can

Pour rappel, un rejecter peut être ajouté à une promise en étant fourni en second paramètre à `.then ()` … :

```
p.then (null, reason => console.log (reason));
```

… ou en unique paramètre à `.catch ()` :

```
p.catch (reason => console.log (reason));
```

Les rejecters d'une promise sont appelés sur appel de `reject ()` dans l'executor… :

```
var p = new Promise ((resolve, reject) => reject ("Error"));
p.catch (reason => console.log (reason));
```

…mais aussi quand une exception est soulevée dans ce dernier, la raison correspondant à l'exception :

```
var p = new Promise ((resolve, reject) => { throw "Error" });
p.catch (reason => console.log (reason));
```

Les rejecters ne sont pas appelés si une exception est soulevée dans un fulfiller. En effet, si un fulfiller est appelé, c'est que la promise est fulfilled. Et un rejecter ne peut être appelé que la promise est rejected. Or une fois fixé, l'état d'une promise ne peut plus changer : une exception soulevée dans un resolver ne peut donc le faire passer de fulfilled à rejected. Ici, le rejecter n'est donc pas appelé :

```
var p = Promise.resolve (666);
p.then (result => { throw "Error" }, reason => console.log (reason));
```

Pour détecter l'exception tout de même, l'usage serait de rajouter un rejecter sur la promesse retournée par `.then ()` :

```
var p = Promise.resolve (666);
p.then (result => { throw "Error" }).catch (reason => console.log (reason));
```

Toutefois, dans son ouvrage, Kyle Simpson oppose à cela le fait que le rejecter pourrait lui-même générer une erreur, etc. Il faudrait donc élaborer un système de gestion des erreurs plus sophistiqué. Pour en savoir plus, se reporter aux solutions qu'il présente dans son ouvrage. •

La suite de cet article dans le prochain numéro

Thierry BOLLET

Powershell DSC, gratuit, robuste et totalement intégrée.

niveau 100

Le langage de script Powershell n'est plus à présenter. Lancé en 2006, et proposé sous le nom de code Monad pour les systèmes d'exploitation Microsoft, Powershell est solidement ancré dans le paysage Windows. Je vous propose une plongée au cœur de Powershell DSC.

L'administration en lignes de commande est devenue peu à peu une norme, même dans l'environnement Microsoft. Toujours en évolution, le produit propose maintenant des versions Core utilisables sous macOS, Linux (pour les distributions les plus populaires) et en version expérimentale sur les processeurs ARM pour Windows. Ce langage s'appuie sur le .NET Framework. Parallèlement à ces évolutions, une offre complémentaire est venue enrichir ce langage de script, elle se nomme Powershell DSC. La version 4 de Powershell propose effectivement une extension au langage nommé DSC (Desired State Configuration ou maintien de la configuration dans l'état désiré), ainsi qu'un moteur de configuration intégré au système d'exploitation.
L'intégration est totale, ce n'est pas une fonctionnalité ; le moteur LCM (Local Configuration Manager) est par défaut en attente de fichier de configuration. Il ne se désactive pas, c'est un élément de base du système. Utilisable sur les OS poste de travail et sur les OS serveur, c'est sur la plateforme serveur que le produit propose le plus d'avantages.
Très fortement amélioré par l'arrivée de Powershell 5.1 avec l'augmentation de la sécurité et l'ajout des audits pré-configuration, DSC se révèle être aujourd'hui un outil indispensable pour qui souhaite maintenir son parc à jour de configuration et corriger "à la volée" les éventuelles configurations qui ne se trouvent plus dans l'état désiré.
Quelques exemples simples :
- Modification du mode de démarrage d'un service et état du service.
- Présence ou absence d'une fonctionnalité Windows.
- Contrôle et modification d'une valeur de clef dans le registre.
- Modification du contenu des groupes et comptes locaux pour un serveur, etc.

Ce ne sont pas moins de 23 ressources disponibles de base avec le système qui permettront de traiter un grand nombre de situations. On affichera ces ressources à l'aide de la commande Get-DscResource sur une machine Windows Server (version 2016 dans cet exemple). 1

```
PS C:\Users\administrateur.DSC> Get-DscResource

ImplementedAs   Name                    ModuleName                      Version
-------------   ----                    ----------                      -------
Binary          File
Binary          SignatureValidation
PowerShell      Archive                 PSDesiredStateConfiguration     1.1
PowerShell      Environment             PSDesiredStateConfiguration     1.1
PowerShell      Group                   PSDesiredStateConfiguration     1.1
Composite       GroupSet                PSDesiredStateConfiguration     1.1
Binary          Log                     PSDesiredStateConfiguration     1.1
PowerShell      Package                 PSDesiredStateConfiguration     1.1
Composite       ProcessSet              PSDesiredStateConfiguration     1.1
PowerShell      Registry                PSDesiredStateConfiguration     1.1
PowerShell      Script                  PSDesiredStateConfiguration     1.1
PowerShell      Service                 PSDesiredStateConfiguration     1.1
Composite       ServiceSet              PSDesiredStateConfiguration     1.1
PowerShell      User                    PSDesiredStateConfiguration     1.1
PowerShell      WaitForAll              PSDesiredStateConfiguration     1.1
PowerShell      WaitForAny              PSDesiredStateConfiguration     1.1
PowerShell      WaitForSome             PSDesiredStateConfiguration     1.1
PowerShell      WindowsFeature          PSDesiredStateConfiguration     1.1
Composite       WindowsFeatureSet       PSDesiredStateConfiguration     1.1
PowerShell      WindowsOptionalFeature  PSDesiredStateConfiguration     1.1
Composite       WindowsOptionalFeatureSet PSDesiredStateConfiguration   1.1
PowerShell      WindowsPackageCab       PSDesiredStateConfiguration     1.1
PowerShell      WindowsProcess          PSDesiredStateConfiguration     1.1
```

1

Fonctionnement du fichier de configuration

Dans tous les scénarii, DSC s'appuie sur un fichier déclaratif (Quoi, quel résultat attendu) et prend en charge de manière transparente les mises à l'état désiré. L'administrateur ne se préoccupe pas de savoir comment réaliser les actions, il va écrire une baseline des points qu'il souhaite garantir, ils seront corrigés par le moteur si l'état n'est plus respecté. Une valeur est modifiée intentionnellement ou par maladresse ? Le moteur local LCM se chargera de la correction.
Ce maintien en condition s'appuie sur deux piliers :

- Le fichier de configuration compilé de type Mof (Managed Object Format).
- Le moteur LCM qui effectue les actions de contrôles et de corrections si nécessaire.

Un exemple avec l'écriture d'une première configuration, une clef de registre doit être présente pour que le serveur soit dans l'état désiré.

```
set-location C:\temp
Configuration Clef1 {
Import-DscResource –ModuleName PSDesiredStateConfiguration
Node 'DSC2' { # Le node, nom de la machine distante.
Registry RegistryExemple #Registry est le nom de la ressource utilisée.
{
Ensure = "Present"
Key = "HKEY_LOCAL_MACHINE\SOFTWARE\Clef1"
ValueName = "Le nom de ma clef"
ValueData = "La valeur de ma clef"}}}
```

Ici, pas d'action de scripting, mais une déclaration d'état. La ressource utilisée est la ressource Registry. Cette ressource prend en charge toute la mécanique de gestion d'une clef de registre. Elle « masque » les difficultés techniques. Le chemin et la valeur de la clef sont indiqués, la cible (machine) de cette configuration également *(Node 'DSC2')*. Ce fichier est écrit sous Powershell puis compilé par la commande *Clef1* qui porte le nom de la configuration (ligne 2 du script). Le fichier généré est un fichier de type Mof qui porte le nom du nœud cible. Dans sa forme la plus simple (méthode Push), la configuration est envoyée à la machine distante par la commande *Start-DscConfiguration "C:\temp\Clef1" -Force -Wait –Verbose*
Cette commande traite le contenu du répertoire courant « C:\temp\Clef1 ». Le nom du fichier Mof porte le nom de la cible (DSC2.Mof). Il est envoyé pour traitement au moteur LCM de la machine DSC2. Le paramètre *–Verbose* est intéressant puisqu'il permet d'afficher sur la machine source la réponse du moteur de configuration de la machine distante. Ici, une vue partielle de cette réponse.

```
COMMENTAIRES : [DSC2]:                [[Registry]RegistryExemple] La clé de Registre
« HKLM:\SOFTWARE\Clef1 » n'existe pas
COMMENTAIRES : [DSC2]: LCM: [ Fin   Test   ] [[Registry]RegistryExemple]  en 0.2340 secondes.
COMMENTAIRES : [DSC2]: LCM: [ Début  Définir ] [[Registry]RegistryExemple]
COMMENTAIRES : [DSC2]:                [[Registry]RegistryExemple] (DÉFINITION) Créer la clé
de Registre « HKLM:\SOFTWARE\Clef1 »
```

```
COMMENTAIRES : [DSC2]:          [[Registry]RegistryExemple] (DÉFINITION) Définir la valeur
de clé de Registre « HKLM:\SOFTWARE\Clef1\Le nom de ma clef » à « La valeur de ma clef » de
type « String »
```

La ressource DSC prend connaissance de la demande de conformité, elle teste l'état de conformité et effectue les mises à jour si nécessaire. Cette commande peut être relancée, DSC est idempotent. Le moteur de configuration de la machine distant a traité le fichier, mais il ne l'appliquera que pour une durée de 10 jours, conformément au paramétrage par défaut lors de l'installation d'une machine. Cette application est partielle, puisque le réglage appliquera le fichier à réception puis consignera les modifications de configuration sans les réappliquer. Le mode de configuration (paramètre *ConfigurationMode* du moteur) étant *ApplyAndMonitor*. Cette solution de Push des configurations n'est donc qu'une étape.

Fonctionnement du moteur

Par défaut, le moteur de configuration LCM est en mode Push. Il attend de recevoir un fichier Mof de configuration comme expliqué précédemment. Les informations du moteur sont affichées par la commande *Get-DscLocalConfigurationManager*. 2

Pour un fonctionnement automatique et durable de la solution, cette configuration est à modifier. La modification s'effectue par la commande *Set-DscLocalConfigurationManager*. Cette commande attend un fichier de configuration du moteur. Cette partie de configuration est assez complète, le moteur est doté de 7 blocs de configuration différents.

- Settings pour l'ensemble des paramètres généraux.
- ConfigurationRepositoryWeb pour le paramétrage du serveur Web.
- ConfigurationRepositoryShare contient les informations de paramétrage pour le client qui utilise un serveur SMB.
- ResourceRepositoryWeb contient les informations de paramétrage pour le client qui utilise les fichiers de modules de ressources pour serveur web.
- ResourceRepositoryShare contient les informations pour le client qui utilise les fichiers de modules de ressources pour serveur SMB.
- ReportServerWeb contient les informations pour le serveur de rapport.
- PartialConfiguration pour le paramétrage des configurations partielles.

Un paramétrage très complet est possible, il répondra à toutes les situations. Tous les blocs ne sont pas obligatoires. Dans l'exemple présenté plus bas dans ce sujet, seul les blocs Settings et ConfigurationRepositoryShare seront utilisés. Une version de base ne nécessite qu'un seul bloc, le bloc Settings.

Fonctionnalité de la méthode Pull

Le mode Pull est un mode de fonctionnement qui permet aux machines de venir « tirer » leur configuration automatiquement sur un serveur central appelé serveur collecteur. Ce serveur met à disposition une configuration pour chaque machine. Toutes les 15 minutes, le moteur de configuration LCM d'une machine cliente vérifie si le fichier de configuration qu'il stocke localement est bien la dernière version disponible sur son serveur de configuration. Il récupère la dernière version puis applique les modifications. S'il possède déjà la bonne version du fichier, il maintient sa configuration actuelle. Le maintien de la configuration est durable dans le temps avec la méthode Pull. Si le serveur collecteur n'est pas disponible, le maintien de la configuration est assuré sur le modèle de la méthode Push. C'est-à-dire que le serveur considère que son fichier de configuration est valide 10 jours. Une fois le serveur collecteur de nouveau en ligne, le mécanisme de vérification et d'application reprend normalement.

Dans l'exemple proposé, un serveur collecteur de type SMB est créé. La maquette est composée d'un serveur collecteur DSC1 et d'une machine cliente DSC2. Le serveur DSC1 a une partition D:\. Une arborescence DSC est créée. Les commandes sont lancées depuis la machine DSC1.

```
New-item -Name DscSmbShare -Path D:\ -ItemType Directory
New-SmbShare -Path 'D:\DscSmbShare' -Name DscSmbShare
New-item -Name Configurations -Path D:\DscSmbShare -ItemType Directory
New-item -Name WorkFolder -Path D:\DscSmbShare -ItemType Directory
Grant-SmbShareAccess -Name DscSmbShare -AccountName 'DSC\DSC2$' -AccessRight
 Read -Confirm:$false
```

Un script de configuration simple est préparé, il vérifie la conformité d'une clef de registre. En mode Pull, le node ne porte plus le nom de la machine, mais un Guid stocké dans la variable $GUID.

```
$GUID=New-Guid
Set-location "D:\DscSmbShare\WorkFolder"##Evite que la configuration soit compilée
 dans le répertoire courant.
configuration TestRegistre {
Import-DscResource –ModuleName PSDesiredStateConfiguration
    Node $GUID {
      Registry RegistryExample #Registry est le nom de la ressource utilisée.
      {
      Ensure     = "Present" ## L'état désiré, la clef doit être présente
      Key        = "HKEY_LOCAL_MACHINE\SOFTWARE\SMBCreate"
      ValueName  = "TestValue"
      ValueData  = "TestData"} } }
```

La commande *TestRegistre* lance la compilation. Le PS1 devient Mof. Pour garantir à la machine cliente la validité de son fichier Mof, un fichier de somme de contrôle est créé.

```
Move-Item  "D:\DscSmbShare\WorkFolder\TestRegistre" "D:\DscSmbShare\Configurations\"
New-DSCCheckSum -Path "D:\DscSmbShare\Configurations\TestRegistre\"
```

Cette opération est obligatoire. Le répertoire « TestRegistre » doit contenir un fichier Mof accompagné de son fichier mof.checksum. Ces 2 fichiers portent le GUID de la configuration. Pour terminer, le moteur de configuration de la machine DSC2 cliente du collecteur est mis à jour avec les informations de type Pull. Ci-dessous, la création du fichier meta.mof, fichier de configuration du moteur LCM. Le nom du serveur collecteur et le partage qui stocke le fichier Mof (*ConfigurationRepositoryShare*), le GUID (*ConfigurationID*) de son fichier de configuration. Le *refresh mode Pull* et le mode de configuration

```
PS C:\Users\administrateur.DSC> Get-DscLocalConfigurationManager

ActionAfterReboot              : ContinueConfiguration
AgentId                        : BF7BB458-17F9-11E9-8845-080027AFCF30
AllowModuleOverWrite           : False
CertificateID                  :
ConfigurationDownloadManagers  : {}
ConfigurationID                :
ConfigurationMode              : ApplyAndMonitor
ConfigurationModeFrequencyMins : 15
Credential                     :
DebugMode                      : {NONE}
DownloadManagerCustomData      :
DownloadManagerName            :
LCMCompatibleVersions          : {1.0, 2.0}
LCMState                       : Idle
LCMStateDetail                 :
LCMVersion                     : 2.0
StatusRetentionTimeInDays      : 10
SignatureValidationPolicy      : NONE
SignatureValidations           : {}
PartialConfigurations          :
RebootNodeIfNeeded             : False
RefreshFrequencyMins           : 30
RefreshMode                    : PUSH
ReportManagers                 : {}
ResourceModuleManagers         : {}
PSComputerName                 :
```

2

ApplyAnAutocorrect. Cette valeur *ApplyAnAutocorrect* n'est pas la valeur par défaut. Attention, la valeur par défaut *ApplyAndMonitor* applique une seule fois la configuration puis notifie les modifications de conformité dans le gestionnaire d'évènements dédié DSC. Elle ne réaligne pas la machine.

```
[DSCLocalConfigurationManager()]
configuration SmbParamNoeuds
   { Node DSC2
     {Settings
       {
          RefreshMode = 'Pull' ## Modification du mode de configuration
          RefreshFrequencyMins = 30 # Intervalle de vérification de la conformité.
          RebootNodeIfNeeded = $true # Autorise le redémarrage si nécessaire.
          ConfigurationID  = $GUID ## Injection de la valeur $GUID qui identifie le fichier
de configuration. Unique pour chaque machine cliente du serveur collecteur.
          ConfigurationMode = 'ApplyAndAutoCorrect' ## Applique puis force la
correction au fil de l'eau. C'est l'un des 3 états disponibles pour ce paramètre.
              AllowModuleOverwrite = $true # Ecrase un module local en cas de mise à
jour garantissant systématiquement l'utilisation du module le plus récent.
       }
        ConfigurationRepositoryShare SmbConfigShare
    {SourcePath='\\DSC1\DscSmbShare\Configurations\TestRegistre\'}}} ## Partage
dédié sur le collecteur, c'est l'emplacement centralisé ou sont stockées les configurations.
SmbParamNoeuds
```

Le moteur est mis à jour par la commande :

```
Set-DscLocalConfigurationManager -Path "D:\DscSmbShare\WorkFolder\SmbParam
Noeuds\" -CimSession  DSC2 –force.
```

Il reste à afficher la configuration du moteur DSC2 par la commande :

```
Get-DscLocalConfigurationManager -CimSession DSC2.
```

La machine DSC2 est correctement paramétrée. Le mode ***Pull*** est présent, la configuration dédiée portera le ***GUID 63c07d13-33fe-42e6-8b59-ba8b7a5f851f***, et la valeur ***ConfigurationDownload Managers*** est l'information de partage pour le stockage de la configuration sur la machine collecteur.

Fonctionnalité avancées.

Cette présentation ne peut se terminer sans présenter les fonctionnalités avancées de DSC.

3

```
PS D:\DscSmbShare\WorkFolder> Get-DscLocalConfigurationManager -CimSession DSC2

ActionAfterReboot              : ContinueConfiguration
AgentId                        : 9A8E6ACC-C6B1-11E7-855E-0800273ECC2A
AllowModuleOverWrite           : True
CertificateID                  :
ConfigurationDownloadManagers  : {[ConfigurationRepositoryShare]SmbConfigShare}
ConfigurationID                : 63c07d13-33fe-42e6-8b59-ba8b7a5f851f
ConfigurationMode              : ApplyAndAutoCorrect
ConfigurationModeFrequencyMins : 15
Credential                     :
DebugMode                      : {NONE}
DownloadManagerCustomData      :
DownloadManagerName            :
LCMCompatibleVersions          : {1.0, 2.0}
LCMState                       : Idle
LCMStateDetail                 :
LCMVersion                     : 2.0
StatusRetentionTimeInDays      : 10
SignatureValidationPolicy      : NONE
SignatureValidations           : {}
MaximumDownloadSizeMB          : 500
PartialConfigurations          :
RebootNodeIfNeeded             : True
RefreshFrequencyMins           : 30
RefreshMode                    : Pull
ReportManagers                 : {}
ResourceModuleManagers         : {}
PSComputerName                 : DSC2
PSComputerName                 : DSC2
```

Configuration partielle : ce mode de configuration permet de déployer des fragments de fichiers mof qui sont placés en attente sur la machine cliente. Ce mode permet d'impliquer différentes équipes et de décentraliser l'écriture des fichiers. Une fois toutes les configurations partielles distribuées, elles sont publiées puis compilées en un seul fichier par la commande *Publish-DscConfiguration*. Le LCM est informé de cette publication et considère que ce fichier Mof est son nouveau fichier de référence.

Certificats : l'ajout de certificats pour augmenter le niveau de sécurité des fichiers Mof. Si la configuration nécessite d'utiliser un compte de haut niveau, les informations ne sont pas stockées en clair mais DSC utilise un certificat de chiffrement de documents pour masquer les informations d'identifications.

Ressources libres et personnalisées : les 23 ressources disponibles par défaut ne sont pas toujours suffisantes. Il existe des centaines de ressources partagées par la communauté pour étendre la fonctionnalité. Ces ressources sont partagées et facilement accessibles. Si un besoin n'est pas couvert, il est possible de créer ses propres ressources et de les ajouter aux ressources existantes. On parle ici de ressources personnalisées.

Ressources composites : ces ressources utilisent une ressource classique, mais étendent l'utilisation et permettent un traitement par lots. Par exemple, la ressource *Service* adressera un seul service par bloc de configuration. La ressource composite *ServiceSet*, extension de la ressource *Service* autorise le traitement par lot, c'est-à-dire X services dans le même bloc de configuration.

Ressource Kit : Microsoft propose, améliore et maintient depuis Décembre 2013 un ressource kit dédié DSC. Les possibilités sont énormes, depuis la configuration et le paramétrage d'un serveur Exchange en passant par la gestion du stockage ou la promotion des contrôleurs de domaines.

Fichier de nœuds : le fichier de configuration peut s'adresser à une seule machine ou à plusieurs. Le fichier de nœud permet d'isoler certains paramètres. Lors de la compilation, un fichier de configuration unique générera des fichiers Mof différents pour chaque type de machine.

Conclusion.

DSC est en évolution constante et étend petit à petit son domaine d'utilisation. Outil de conformité, il devient un outil de paramétrage et d'installation. La communauté est active, avec plusieurs centaines de modules libres. Son intégration est totale et disponible pour le Cloud Microsoft Azure. Il est notamment proposé au travers de la brique Azure Automation qui met à disposition un serveur collecteur sous la forme de service. Ce service permet de gérer les machines Cloud, mais aussi des machines On Premise.

Les dernières nouveautés Azure proposent également en preview de rédiger ses configurations depuis une interface graphique disponible sur le menu des machines virtuelles avec le module Desired State Configuration Management.

Linux peut également bénéficier de la fonctionnalisé au prix de l'installation de quelques paquets et du module Nx dédié qui propose 11 ressources de gestion.

Un Datacenter (Cloud, On Prem) sans DSC est-il encore possible ? Alors qu'une solution gratuite, intégrée et robuste existe ? Certainement pas …

Un ensemble de ressource est disponible à cette adresse : https://www.powershellgallery.com/

Valentin Koukponou
Consultant et formateur à Zenika Paris

zenika
<animés par la passion>

Parlons Deep Learning !

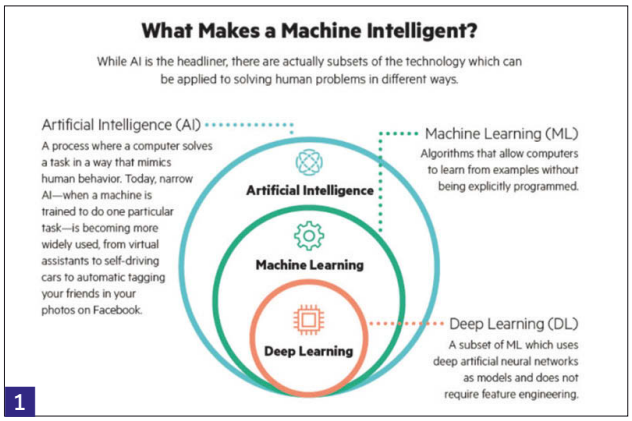
On n'a jamais autant entendu parler de l'intelligence artificielle (IA) que ces 10 dernières années. L'IA washing est très répandue, au point qu'il est devenu facile de confondre IA, Deep Learning et réseaux de neurones.

niveau 200

Dans cet article, nous allons éclaircir ces trois termes. Ensuite, nous allons implémenter le premier réseau de neurones de l'histoire à savoir le **perceptron**, en utilisant **TensorFlow**. Nous entraînerons notre perceptron à simuler les problèmes logiques, le OU par exemple. Pour finir, nous allons exposer nos modèles de réseaux de neurones via une application **Rest** développée en **Node.JS** et interfacée par un client en **React.JS** pour l'affichage. Le choix de ces technos est motivé par une volonté de décloisonner les univers, en collaborant avec des devs front. Cette dernière partie est écrite en collaboration avec mon collègue Joan Anagbla.

IA – Deep Learning – Réseaux de neurones

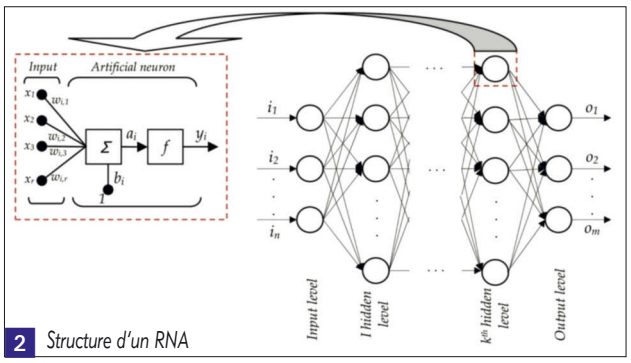
L'intelligence artificielle est un ensemble de techniques permettant de simuler l'intelligence humaine. Elle est née dans les années 50. 1



1

Il y a plusieurs branches dans l'I.A. (liste non exhaustive) :

- Représentation des connaissances ;
- Résolution de problèmes généraux ;
- Robotique ;
- Vision automatique (Computer Vision aka CV) ;
- Traitement de langage naturel (NLP). Ces branches utilisent le plus souvent des techniques d'apprentissage automatique (Machine Learning) ou d'apprentissage profond (Deep Learning). Un système de Deep Learning utilise exclusivement les réseaux de neurones artificiels pour apprendre. Quand on parle de réseaux de neurones artificiels (RNA), on pense à : 2



2 *Structure d'un RNA*

Un RNA est composé de :

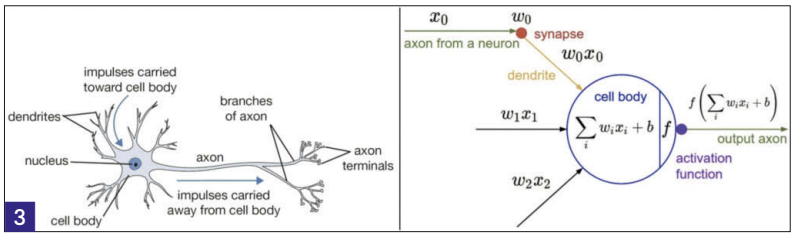
- Une couche d'entrée ;
- Une couche de sortie ;
- Une ou plusieurs couches intermédiaires, dites couches cachées. Plus il y a de couches cachées, plus l'apprentissage est profond (deep).
- Et chaque couche est composée d'un ou plusieurs neurones.

Neurones artificiels vs Neurones biologiques

Comme vous pouvez vous en douter, les RNA sont inspirés des réseaux de neurones biologiques (RNB).

RNB	RNA	Rôle
Dendrites	Input	Point d'entrée de l'information
Soma	Noeud	Processing de l'information
Axones	Output	Point de sortie de l'information
Synapses	Interconnexion des poids	Connexion aux autres neurones

Zoom sur le fonctionnement d'un neurone 3



3 *Neurone biologique (gauche) et son modèle mathématique (droite)*

- x0,x1,x2,…xn sont les entrées du neurone. Ce sont soit les données brutes (pour la couche d'entrée), soit les valeurs intermédiaires issues d'une couche cachée : couches comprises entre la couche d'entrée et la couche de sortie.
- w0,w1,w2,…wn sont les poids de chaque entrée (biais compris)
- x0 est le biais du modèle. C'est une constante permettant de décrire un comportement initial du modèle. Il vaut 1.
- f est la fonction d'activation.

La sortie d'un neurone est donnée par :

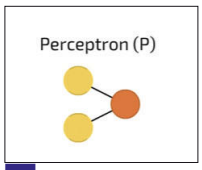
ai=f(∑i=0nwixi)

En choisissant la fonction sigmoid comme fonction d'activation, on construit facilement une couche via **TensorFlow**:

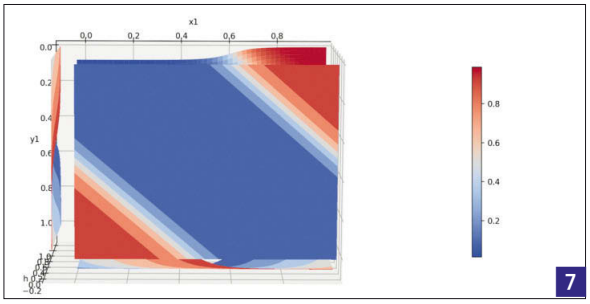
```
1 def CreateOneLayer(X, n_neurones):
2 W = tf.get_variable("W", [X.get_shape()[1].value, n_neurones], tf.float32)
3 b = tf.get_variable("b", [n_neurones], tf.float32)
4 return tf.nn.sigmoid(tf.add(tf.matmul(X, W), b))
```

```
2018-07-09 17:34:27.855145: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow
 binary was not compiled to use: AVX2 FMA

Dealing with a <X...> model ...

Step : 0     - Lost = 0.2740226686000824
Step : 100   - Lost = 0.16782978177070618
Step : 200   - Lost = 0.16672033071517944
Step : 300   - Lost = 0.0008497289381921291
Step : 400   - Lost = 0.0004332107200752944
Step : 500   - Lost = 0.0002788209822028754
Model trained. Session saved in ./models/xor.ckpt

Training done ...


Dealing with a <X...> model ...

 x1| x2| x1 XOR x2
 ---+---+----------
 0 | 0 | 0.015
 0 | 1 | 0.985
 1 | 0 | 0.985
 1 | 1 | 0.021
```

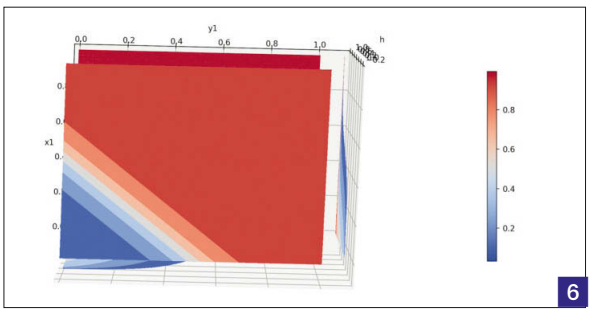
5



4



7



6

Pour rappel, TensorFlow utilise des Tensors : des vecteurs multidimensionnels. La fonction CreateOneLayer crée ainsi une couche de neurones et non un neurone(cellule).

L'apprentissage d'un RNA :

Avant l'étape d'entraînement effective, on définit l'opérateur train_op qui permet de minimiser l'erreur des moindres carrés loss_op.

```
1 loss_op = tf.reduce_mean(tf.square(tf.subtract(predictions, labels)))
2 train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss_op)
```

Entraîner le modèle consiste à trouver les poids wi,i=1…n qui minimise la fonction de coût.

```
1  def train(X_train, y_train, n_iterations, _model_) :
2  saver = tf.train.Saver()
3  for step in range(n_iterations + 1):
4      _, cost = sess.run([training_op, loss_op], feed_dict={X: X_train,
5  y: y_train})
6      if step % 100 == 0:
7          print("Step : {: <5} - Lost = {: <23}".format(step, cost))
8  save_path = saver.save(sess, "./models/"+_model_+".ckpt")
```

Après la phase d'apprentissage du système, on exporte le modèle dans le répertoire save_path, notamment ./models/or.ckpt pour le modèle du OU logique. Toutefois, pour changer de modèle, il suffit d'assigner à la variable _model_ une des quatre valeurs possibles : or, xor, and et xnor, et d'adapter ley_train.

Nous avons vu le fonctionnement d'un neurone et comment le paramétrer. Passons maintenant à l'entraînement du **Perceptron**.

Le perceptron de Rosenblatt

Connu comme le premier système artificiel capable d'apprendre par l'expérience, le Perceptron est composé de deux neurones sur sa couche d'entrée et d'un neurone pour la couche de sortie. 4

Ses inputs et outputs sont des valeurs binaires. Ce qui fait de ce système un solveur d'opérations logiques, notamment le ET et le OU logique.

On implémente avec TensorFlow le Perceptron :

```
1  X = tf.placeholder(tf.float32, [None, 2], name="X-input")
2  y = tf.placeholder(tf.float32, [None, 1], name="y-output")
3
4  def model(_model_):
5    if 'x' not in _model_ : # Si _model_ == xnor ou xor
6      with tf.variable_scope(_model_+"layer1", reuse = tf.AUTO_REUSE):
7        model = CreateOneLayer(X, 1)
8    [...]
```

Puis après 500 itérations d'entraînement, on sauvegarde le modèle. Les données d'entraînement pour le OU logique sont :

```
1 X_train = np.array([[0, 0],
2              [0, 1],
3              [1, 0],
4              [1, 1]])
5
6 y_train = np.array([[0],
7              [1],
8              [1],
9              [1]])
```

La logique booléenne nous permet de savoir que :

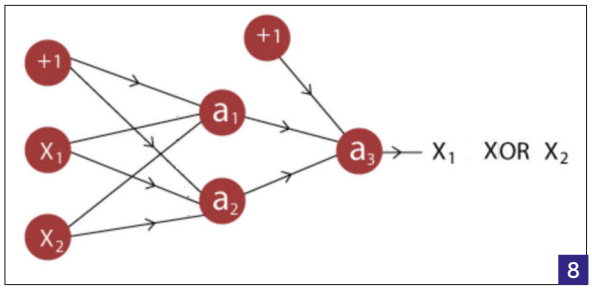
x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

En restaurant le modèle précédemment sauvegardé, les prédictions issues du modèle entrainé sont : 5

Quand on représente en 3D les outputs du modèle, on obtient un graphe sur lequel on peut séparer clairement les 0 des 1. En d'autres termes, le problème du OU logique est **linéairement séparable**. 6

Celui du OU exclusif ne l'est pas. Sur le plan de base (x1,y1): on ne peut pas séparer la zone *bleue* de la zone *rouge* avec une droite. 7

Le **perceptron** dans sa version simple est donc incapable de résoudre le problème du XOR. D'où la nécessité d'ajouter des



8

couches cachées (intermédiaires).
La décomposition du *XOR*, nous donne :

X1⊕X2====X1.X2⁻⁻⁻⁻⁻⁻⁻+X1⁻⁻⁻⁻⁻⁻⁻.X2(X1+X2).(X1⁻⁻⁻⁻⁻⁻⁻+X2⁻⁻⁻⁻⁻⁻⁻)(X1+X2)(X1.X2⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻)(X1+X2).(X1⁻⁻⁻⁻⁻⁻⁻+X2⁻⁻⁻⁻⁻⁻⁻)

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

(Notons que '+' signifie OU et '.', ET)

Le modèle du *XOR* est constitué de :
- une couche d'entrée à deux neurones ;
- une couche cachée à deux neurones ;
- une couche de sortie à un neurone ;
- chaque couche dispose d'un neurone pour le biais (bias en anglais). 8

```
1  def model(_model_):
2     [...]
3     else :
4        with tf.variable_scope(_model_+"layer1", reuse = tf.AUTO_REUSE):
5           z0 = CreateOneLayer(X, 2)
6        with tf.variable_scope(_model_+"layer2", reuse = tf.AUTO_REUSE):
7           model = CreateOneLayer(z0, 1)
8     return model
```

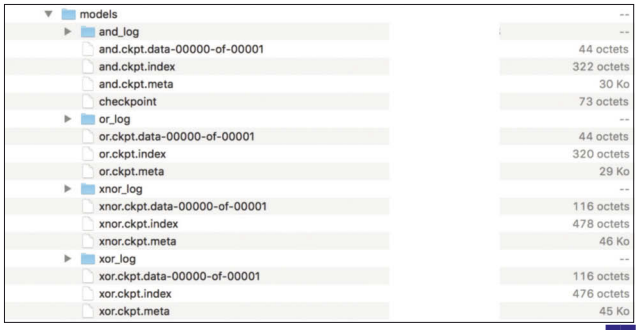
Une fois ce modèle entraîné au bout de 500 itérations : 9

Mise en production des modèles

TensorFlow vient avec la suite TensorServing qui permet de mettre en production des modèles construits avec TensorFlow. Ce sujet fera l'objet d'un prochain article.
Le back-end de notre application est principalement composé des 4 modèles sauvegardés, ainsi que de la fonction qui permet d'effectuer la prédiction 10

```
1  def predict_single(x1, x2, _model_) :
2     z0 = model()
3     saver = tf.train.Saver()
4     saver.restore(sess, "./models/"+_model_+".ckpt")
5     X_in = np.array([x1, x2]).reshape([-1, 2])
6     pred = sess.run(z0, feed_dict={X: X_in})
7     result = np.int(np.round(np.reshape(pred, 1)[0]))
8     return result
```

```
2018-07-09 16:01:43.867747: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow
 binary was not compiled to use: AVX2 FMA
Step : 0    - Lost = 0.28761202096939087
Step : 100  - Lost = 0.010955806680972576
Step : 200  - Lost = 0.003535101655870676
Step : 300  - Lost = 0.0018453459488227963
Step : 400  - Lost = 0.0011619940463295424
Step : 500  - Lost = 0.0008098477846942842
Model trained. Session saved in ./models/or.ckpt

Training done ...

 x1| x2| x1 OR x2
---+---+----------
 0 | 0 | 0.043
 0 | 1 | 0.974
 1 | 0 | 0.973
 1 | 1 | 1.000
```
9



10

Pour pouvoir exploiter notre modèle, nous allons maintenant le déployer sur un serveur et afficher le résultat des prédictions à l'aide d'une page web.

Exposition du modèle

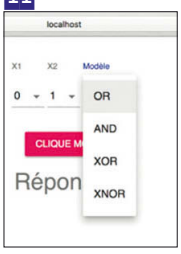
La création d'un serveur Node.JS se fait à l'aide de commandes Linux. Dans un terminal, il faut initier un nouveau projet Node.JS (npm init) et renseigner les informations demandées (nom du projet, etc.). On copie le répertoire des modèles à l'intérieur du projet initié. On utilise ensuite PythonShell pour lancer l'exécution de la prédiction (script Python) depuis un script JavaScript:

```
1  PythonShell.run("predict.py", options, function (err, result) {
2     if (err) {
3        res.send(err); //Retourne une erreur en cas d'erreur
4     }
5     else {
6        res.send(result); //Expose le résultat grâce à la fonction callback de express (res.send)
7     }
8  });
```

Enfin, on expose le résultat sur le web grâce à une Api Rest (créée avec Express).
Affichage du résultat 11

Pour créer la page web chargée de contacter le serveur et d'afficher le résultat des prédictions, nous avons fait le choix de React.JS. React est un framework créé par Facebook permettant de développer des applications web. En plus de React nous avons utilisé une bibliothèque de composants Material-UI. Pour que l'application web puisse contacter le serveur nous avons eu recours à la bibliothèque axios.
Félicitations… Vous venez de mettre en production votre premier modèle de Deep Learning.
Tous les codes de cet article sont disponibles sur mon https://github.com/JuniorKoukponou/Deep-Learning-Back-To-Basics.

11



Jean-Pierre Gervasoni

L'interopérabilité .net

Selon le Larousse, « l'interopérabilité » c'est la capacité de matériels, de logiciels ou de protocoles différents à fonctionner ensemble et à partager des informations. Dans le cas de l'interopérabilité .net, cela s'applique au partage d'informations entre du code managé et non-managé.

niveau 200

Comme définition simple de *code managé*, on peut dire que c'est un code dont l'exécution est gérée par un runtime. Dans le cas de dotnet, il s'agit du *Common Language Runtime* ou CLR pour les intimes. Quand on compile un code managé, on génère un bytecode (code binaire non exécutable) appelé *Common Intermediate Language* ou CIL. Un des rôles de la CLR est de prendre ce bytecode et de générer à la volée du code binaire exécutable par un processeur, étape appelée *just-in-time compiling*. La CLR fournit en plus des services tels que la gestion automatique de la mémoire, des vérifications de sécurité quand deux applications interagissent.

Par opposition, dans un code natif, le développeur gère les aspects de mémoires, de sécurité, de consommation de ressources. Quant au programme obtenu, c'est du binaire que le système d'exploitation peut charger et faire exécuter. La communication entre ces deux mondes peut se faire via différentes techniques :

- Utilisation d'une couche C++/CLI ;
- Platform Invocation Services ou P/Invoke ;
- COM interop ;
- Les canaux nommés (ou named pipes). Moyen simple de communication inter-process en local ou remote, supportant une communication asynchrone et full duplex.
- La gestion de l'échange de données entre des codes managés et natifs est prise en charge par les services de marshaling de la CLR.

Il est possible de cloner le dépôt Git suivant afin d'avoir accès au code de cet article : https://github.com/nitrogene/DotNetInterops.git

Apartments et objets COM

Un apartment est un environnement dans lequel vivent des objets COM. Ce n'est ni un thread, ni un process, et cela gère les communications entre clients et objets COM. Son système de threading est fixé à sa création, et ne peut être changé. En C++, l'initialisation de COM par un thread se fait avec la fonction CoInitializeEx :

```
HRESULT CoInitializeEx(
 LPVOID pvReserved,
 DWORD  dwCoInit
);
```

Le premier paramètre doit toujours être égal à 0. Le deuxième paramètre peut être soit COINIT_APARTMENTTHREADED, ce qui créé un STA ou Single Thread Apartment, soit COINIT_MULTITHREADED, ce qui créé un MTA ou Multiple Thread Apartment. CoInitializeEx doit absolument être appelée au moins une fois par thread. Des appels successifs renvoient soit S_FALSE, ce qui indique que COM est déjà initialisé, soit RPC_E_CHANGED _MODE, ce qui signifie qu'on tente de changer le type d'apartment, ce qui n'est pas permis.

Le thread peut ensuite charger une ou des librairies COM en faisant appel à CoCreateInstance. A noter que chaque appel CoInitializeEx renvoyant S_OK doit être contrebalancé par un appel à CoUninitialize, une fois que le thread a fini d'utiliser COM.

Single thread apartment

Un même process peut avoir zéro ou plusieurs STA, un par thread ayant fait appel CoInitializeEx. Seul ce thread peut accéder directement aux objets COM résidant dans cet apartment. C'est le modèle de threading le plus simple car COM se charge de quasi tout le travail de synchronisation d'accès par d'autres threads aux objets COM de cet apartment. Un serveur COM in-proc est associé à une clé de registre indiquant notamment le chemin de la dll ou de l'exe et le modèle de threading :

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID
 {CLSID}
  InprocServer32
   (Default) = path
   ThreadingModel = value
```

Si ThreadingModel n'est pas spécifié, le serveur COM est supposé non thread-safe et une seule instance de ce serveur est donc créée dans le STA principal, celui du premier thread ayant appelé CoInitializeEx. Seul ce thread pourra accéder directement au serveur COM, les autres devront passer par du marshaling entre leur apartment et le principal. Ce qui peut être source de lenteurs à l'exécution.

Dans la mesure du possible, il vaut mieux mettre ThreadingModel à Apartment, ce qui indique que le serveur COM est thread safe, et par conséquent on pourra avoir autant d'instances de ce serveur que de threads ayant fait un CoCreateInstance.

Multiple thread apartment

Un process peut avoir zéro ou un seul MTA. Tous les threads d'un process faisant appel à CoInitializeEx partagent le même MTA, et peuvent accéder directement aux objets COM qui résident dans cet apartment. La synchronisation d'accès est à la charge du développeur.

Dans le cas d'un serveur in-proc codé pour supporter le mode MTA, il faut mettre ThreadingModel à Free. Quel que soit le type d'apartment du thread client, le serveur COM sera instancié dans le MTA du process parent.

Modèle client-serveur mixte

Comme on l'a vu, le threading model d'un client et d'un serveur peuvent être différents. L'apartment dans lequel est créé le serveur COM, ainsi que les communications inter-objets sont résumés dans le tableau suivant :

Threading model client	Threading model serveur	Instanciation du serveur dans :	Mode de communication entre objets
STA	MTA	Le MTA du process, créé au besoin par COM	Marshaling
MTA	STA	Un nouveau STA créé par COM	Marshaling
STA	Both	Le STA du client	Accès direct
MTA	Both	Le MTA du process, créé au besoin par COM	Accès direct

Les différents types d'interopérabilité

Via C++ avec extensions managées ou C++/CLI

Le C++/CLI (C++ modified for *Common Language Infrastructure*) est une extension du langage C++ standardisée par l'ECMA en 2005 en tant que ECMA-372. Son but est de servir de glue entre du code managé et du code C++ natif. Pour générer une assembly utilisable depuis par exemple un projet C#, il faut créer un nouveau projet de type librairie dynamique et activer le support de la CLR. Voici quelques éléments spécifiques au C++/CLI.

Pour pointer sur des objets managés, on utilise des handles, la mémoire allouée étant gérée par le garbage collector :

```
System::String^ pStr = gcnew System::String("Stack");
```

Il est possible de définir des classes, interfaces, structures et énumérations managées :

```
// This class will be seen by CLR world
public ref class NativeLibraryWrapper:DisplayMessageInterface // :IDisposable, automatically
 added by compiler
{
private:
    // Pointer to native library
    NativeLibrary* p_NativeLibrary;

protected:
    // This is a finalizer, called when the garbage collector decides to delete this object
    !NativeLibraryWrapper();        // converted to virtual void Finalize() by compiler;

public:
    NativeLibraryWrapper();

    // The destructor is called when the user delete the object
    ~NativeLibraryWrapper();          // converted to IDispposable.Dispose() by compiler;
};
public interface class DisplayMessageInterface
{
    void DisplayMessage(System::String^ from, System::String^ message, [System::Runtime::
InteropServices::Out] System::String^% answer);
};
```

```
public value struct MyStruct
{
    System::String^ from;
    System::String^ message;
};

public enum struct MyEnum
{
    UN, DOS, TRES
};
```

Attention, une ref class ne peut pas hériter d'une classe native ! Les variables managées peuvent être allouées sur la pile ou le tas.

```
MyEnum^ pMyEnum = gcnew MyEnum(); // Réside dans le tas
MyEnum  myENum;                   // Réside dans la pile
```

A noter que les types valeur comme les enum struct et value struct sont passés par valeur, ils peuvent donc être créés directement sur la pile.

Il y a trois moyens de caster un handle en un autre type :

- **Safe cast** : si la conversion est impossible, une exception de type InvalidCastException est levée, c'est l'équivalent en C# d'un type cast.

```
B^ pB = gcnew B();
C^ pC = gcnew C();
A^ pA = (A^)pB;
A^ pA = safe_cast<A^>(pB);
```

- **Dynamic cast** : si la conversion échoue, nullptr est retourné. C'est l'équivalent en C# du mot clé as.

```
C^ pC2 = dynamic_cast<C^>(pA);
```

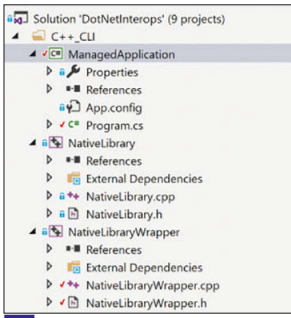
- **Static cast** : aucune vérification n'est faite, si la conversion échoue, on aboutira à un comportement indéfini. A noter que ce genre de cast ne peut pas être utilisé si l'option de compilateur CLR:safe est activée.

```
D^ pD = static_cast<D^>(pA);
```

Pour de plus amples informations sur ce sujet, vous pouvez vous référer au standard ECM-372 à l'adresse suivante https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-372.pdf. 1

Le C++/CLI est donc l'outil de choix pour rendre accessibles des librairies natives au monde .net. Il suffit d'activer le support de la CLR dans un projet de type librairie dynamique. Supposons que l'on dispose d'une librairie statique native, dans laquelle on souhaite accéder depuis du code managé à la classe suivante (définie dans le projet NativeLibrary) :

```
#pragma once
#include <string>
```



1 *Wrapper C++/CLI*

```
class NativeLibrary
{
public:
void DisplayMessage(const std::wstring& from, const std::wstring& message, std::wstring&
 answer);
};
```

L'idée est de créer une classe managée, dont le constructeur sera en charge de créer une instance de la classe native ci-dessus :

```
// This class will be seen by CLR world
public ref class NativeLibraryWrapper:DisplayMessageInterface // :IDisposable, automatically
 added by compiler
{
private:
    // Pointer to native library
    NativeLibrary* p_NativeLibrary;

protected:
    // This is a finalizer, called when the garbage collector decides to delete this object
    !NativeLibraryWrapper();          // converted to virtual void Finalize() by compiler;

public:
    NativeLibraryWrapper();

    // The destructor is called when the user delete the object
    ~NativeLibraryWrapper();          // converted to IDisposable.Dispose() by compiler;

    static int MessageBox_(System::IntPtr hWnd, System::String^ text,
        System::String^ caption, unsigned int type);

    virtual void DisplayMessage(System::String^ from, System::String^ message, [System::
Runtime::InteropServices::Out] System::String^% answer);
};
```

A noter qu'une ref class ne peut pas contenir de smart pointer, d'où l'utilisation d'un pointeur nu sur la classe NativeLibrary. L'implémentation de DisplayMessage est la suivante :

```
#pragma unmanaged
#include "../NativeLibrary/NativeLibrary.h"
#pragma managed

#include "NativeLibraryWrapper.h"
#include <msclr/marshal_cppstd.h>

NativeLibraryWrapper::NativeLibraryWrapper()
{
    this->p_NativeLibrary = new NativeLibrary();
}

NativeLibraryWrapper::~NativeLibraryWrapper()
{
    this->!NativeLibraryWrapper();
}

NativeLibraryWrapper::!NativeLibraryWrapper()
{
    delete this->p_NativeLibrary;
}

void NativeLibraryWrapper::DisplayMessage(System::String^ from, System::String^ message,
 [System::Runtime::InteropServices::Out] System::String^% answer)
{
    auto nFrom = msclr::interop::marshal_as<std::wstring>(from);
    auto nMessage = msclr::interop::marshal_as<std::wstring>(message);
    std::wstring nAnswer;

    this->p_NativeLibrary->DisplayMessage(nFrom, nMessage, nAnswer);

    answer = msclr::interop::marshal_as<System::String^>(nAnswer);
}
```

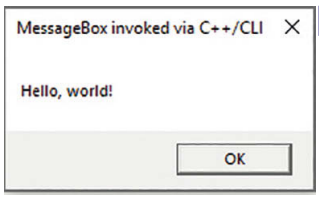
L'utilisation des directives pragma (un)managed permet d'inclure une portion de code qui doit être considérée comme du C++ non managé. Le marshaling des paramètres est assuré par la fonction marshal_as. D'autres exemples de conversions sont donnés dans le tableau suivant :

À partir de type	En type	Marshaler (méthode)	Fichier Include
System::String ^	const char *	marshal_context	Marshal.h
const char *	System::String ^	marshal_as	Marshal.h
Char *	System::String ^	marshal_as	Marshal.h
System::String ^	wchar_t const*	marshal_context	Marshal.h
wchar_t const *	System::String ^	marshal_as	Marshal.h
wchar_t *	System::String ^	marshal_as	Marshal.h
System::IntPtr	HANDLE	marshal_as	marshal_windows.h
HANDLE	System::IntPtr	marshal_as	marshal_windows.h
System::String ^	BSTR	marshal_context	marshal_windows.h
BSTR	System::String ^	marshal_as	Marshal.h
System::String ^	bstr_t	marshal_as	marshal_windows.h
bstr_t	System::String ^	marshal_as	marshal_windows.h
System::String ^	std::String	marshal_as	marshal_cppstd.h
std::String	System::String ^	marshal_as	marshal_cppstd.h
System::String ^	std::wstring	marshal_as	marshal_cppstd.h
std::wstring	System::String ^	marshal_as	marshal_cppstd.h
System::String ^	CStringT<char >	marshal_as	marshal_atl.h
CStringT<char >	System::String ^	marshal_as	marshal_atl.h
System::String ^	CStringT < wchar_t >	marshal_as	marshal_atl.h
CStringT < wchar_t >	System::String ^	marshal_as	marshal_atl.h
System::String ^	CComBSTR	marshal_as	marshal_atl.h
CComBSTR	System::String ^	marshal_as	marshal_atl.h

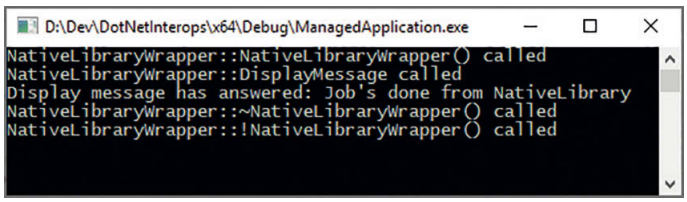
Tableau 1 Utilisation de marshal_as

Une fois cette assembly incluse aux références d'un projet C#, ce dernier pourra appeler directement la fonction DisplayMessage :

```
namespace ManagedApplication
{
  class Program
  {
    static void Main(string[] args)
    {
```



2 *Exemple de sortie 1*



3 *Exemple de sortie-2*

```
        using (var wrapper = new NativeLibraryWrapper())
        {
wrapper.DisplayMessage("MessageBox invoked via C++/CLI", "Hello, world!", out string answer);
            System.Console.WriteLine($"Display message has answered: {answer}");
    }

}
    }
}
```

Voici un exemple de sortie : 2
Très logiquement, le constructeur, le destructeur et le finaliseur sont appelés dans cet ordre : 3
A noter comment le wrapper est vu par la CLR. La classe est bien déclarée comme implémentant l'interface IDisposable, et le troisième paramètre de DisplayMessage est bien out string answer :

```
public class NativeLibraryWrapper : IDisposable
{
  public NativeLibraryWrapper();

  ~NativeLibraryWrapper();

public virtual void DisplayMessage(string from, string message, out string answer);
  public sealed override void Dispose();
  [HandleProcessCorruptedStateExceptions]
  protected virtual void Dispose(bool A_0);
}
```

Via Platform Invoke ou P/Invoke

P/Invoke ou Platform Invocation Services permet à du code managé d'appeler des fonctions dans des dll natives. Voici un exemple simple. Imaginons que l'on souhaite appeler la fonction MessageBox définie dans la dll user32 depuis un programme console C#. Il faut donc examiner la signature de cette fonction :

```
int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType);
```

Et s'appuyer sur une table de correspondance de paramètres pour faire le lien entre les paramètres natifs et managés. A noter que le site http://pinvoke.net fournit un ensemble important de signatures de fonctions invocables par PInvoke. On y trouvera par exemple la signature suivante pour la fonction glBegin, définie dans la dll glut32 :

```
[DllImport("opengl32.dll")]
public static extern void glBegin(uint mode);
```

Et aussi l'enum associée :

```
[Flags()]
  public enum BeginMode : uint
  {
    GL_POINTS = 0x0000,
    GL_LINES = 0x0001,
    GL_LINE_LOOP = 0x0002,
    GL_LINE_STRIP = 0x0003,
    GL_TRIANGLES = 0x0004,
    GL_TRIANGLE_STRIP = 0x0005,
    GL_TRIANGLE_FAN = 0x0006,
    GL_QUADS = 0x0007,
    GL_QUAD_STRIP = 0x0008,
    GL_POLYGON = 0x0009,
  }
```

P/Invoke et DllImport

La signature de MessageBox en C# est la suivante :

```
public static class PInvoke
{
  [DllImport("user32.dll", CharSet = CharSet.Auto)]
  public static extern IntPtr MessageBox(int hWnd, String text,
  String caption, uint type);
}
```

Une fois muni de cette signature, son utilisation est triviale en C# :

```
class Program
{
  static void Main(string[] args)
  {
    var ret = (ushort)WinWrapper.PInvoke.MessageBox(
      0,
      "Hello World!",
      "MessageBox invoked via PInvoke",
      0);

    Console.WriteLine($"MessageBox has exited with {ret} value");
  }
}
```

It Just Works (IJW) :

Le code suivant appelle aussi MessageBox, mais en utilisant IJW dans un projet C++/CLI :

```
int NativeLibraryWrapper::MessageBox_(System::IntPtr hWnd, System::String^ text,
    System::String^ caption, unsigned int type)
{
```

```
    auto textPtr = System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(text);
    auto captionPtr = System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(caption);
    auto nativeHWnd = (HWND)hWnd.ToPointer();

    auto ret=MessageBox(nativeHWnd, (LPCSTR)textPtr.ToPointer(), (LPCSTR)captionPtr.To
Pointer(), type);

    System::Runtime::InteropServices::Marshal::FreeHGlobal(textPtr);
    System::Runtime::InteropServices::Marshal::FreeHGlobal(captionPtr);

    return ret;
}
```

Malgré une complexité de code accrue, il y a clairement des avantages liés à IJW :

- Plus besoin de DllImport, il suffit d'inclure le header requis, et de se lier avec la bonne librairie ;
- C'est un mécanisme plus rapide, car la gestion des paramètres est gérée par le développeur ;
- Si le même jeu de données managées est utilisé par plusieurs appels à des fonctions natives, le marshaling est effectué une seule fois.

IJW est recommandé si votre code fait surtout appel à des types de données ou à des APIs non managés. PInvoke est plus pertinent dans le cas contraire.

Via COM intérops

Les modèles objets COM et .NET ont des différences fondamentales :

- Un client d'objets COM doit gérer de manière explicite la durée de vie de ces objets, alors qu'en .NET cet aspect est géré par la CLR ;
- Un client d'objets COM tente d'accéder à un service via utilisation d'un CoCreateInstance pour avoir accès à un pointeur sur une interface :

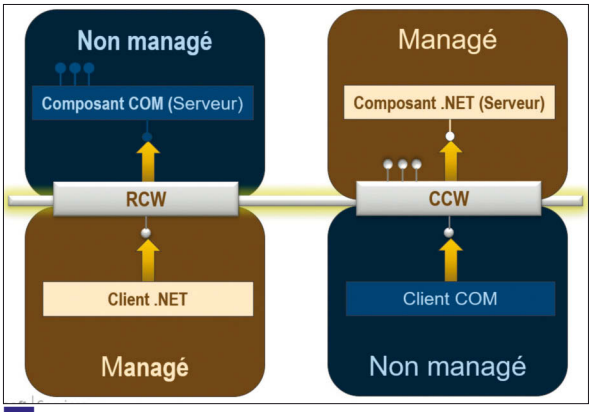
```
IFileOpenDialog *pFileOpen;

// Create the FileOpenDialog object.
hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_ALL,
        IID_IFileOpenDialog, reinterpret_cast<void**>(&pFileOpen));
```

Alors qu'en .NET on utilise la réflexion :

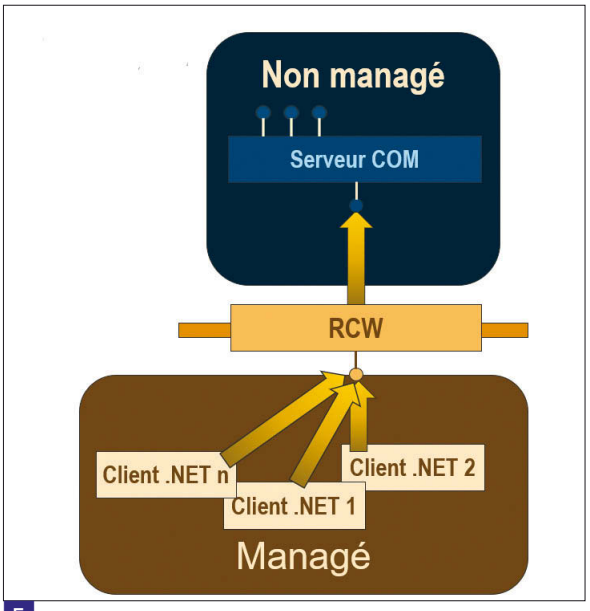
```
    var type = Type.GetTypeFromCLSID(new Guid(CLSIDGuid.FileOpenDialog), true);
var fileOpen = (IFileDialog)Activator.CreateInstance(type);
```

- Les objets .NET sont gérés par la CLR, et leur emplacement mémoire n'est pas défini – il peut varier pour des raisons de performance. Un client d'objets COM, une fois qu'il a obtenu un pointeur valide sur interface suppose que l'objet COM ne change pas de zone mémoire.

La CLR permet toutefois de gérer de manière transparente ces différences, via l'utilisation de classes wrapper RCW et CCW : 4



4 *RCW vs CCW*



5 *Un RCW maintenu par serveur COM et par processus*

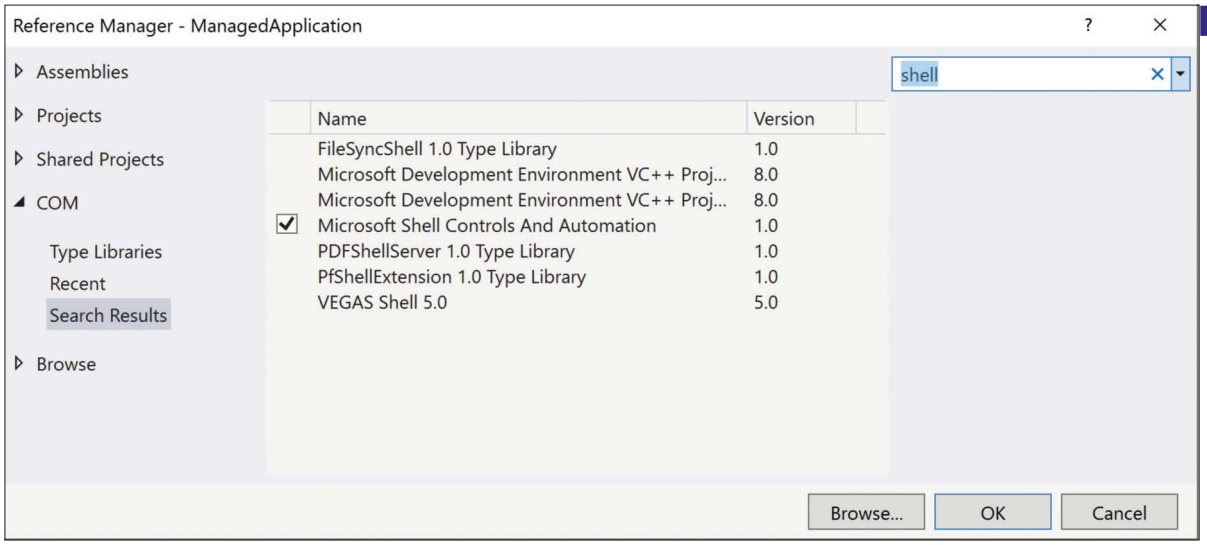
RCW

Quand un client managé accède à une méthode d'un objet COM, un RCW ou *Runtime Callable Wrapper* est créé de même qu'une instance du composant COM. La CLR crée et maintient un seul RCW par process et par serveur COM. De manière transparente, la CLR instancie un objet COM, et le RCW associé, qui sera détruit par le garbage collector dès que plus aucun client ne l'utilisera. Le marshaling des données est automatiquement géré. 5

Pour pouvoir appeler un objet COM non managé en C#, et donc créer de manière transparente un RCW dans Visual Studio, il suffit de cliquer sur Références, Ajouter une référence. Sélectionnez ensuite sur la gauche l'onglet COM – à noter qu'il est impossible d'ajouter par ce moyen une référence à un objet COM managé. 6

Par exemple, pour ouvrir par ce moyen un explorateur Windows, on peut inclure *Microsoft Shell controls and Automation* :

```
// This attribute is only required for Shell32 example
[STAThread]
static void Main(string[] args)
{
```



6 *Ajout d'une référence COM*

```
    // Open a shell explorer via using Microsoft Shell controls and Automation com object
    var shellClass = new Shell32.Shell();
    shellClass.Open("");
  }
```

CCW

Quand un client COM accède à une méthode d'un objet COM, un CCW ou COM *Callable Wrapper* est créé. La CLR maintient un seul CCW par objet COM et par process. Ces proxys sont invisibles aux autres classes du framework dotnet. Le serveur COM est créé dans la zone mémoire gérée par la CLR, son emplacement exact étant donc susceptible de changer au gré des besoins du garbage collector. Le CCW quant à lui, est instancié dans une zone mémoire non gérée par le garbage collector, ce qui permet aux clients COM de le référencer directement, de manière transparente.

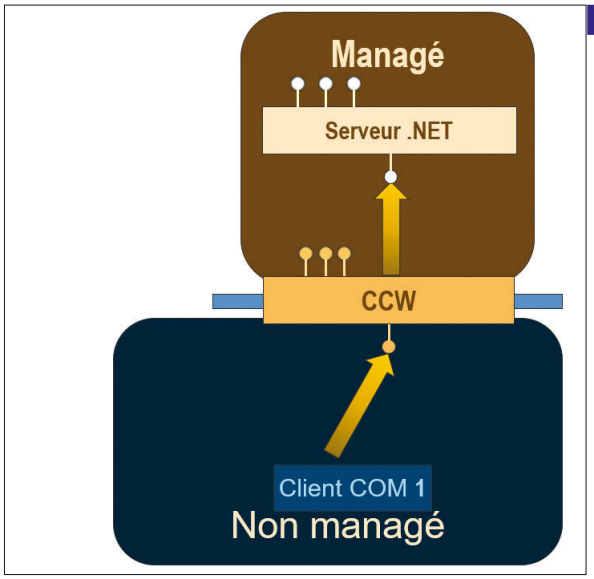
De même que pour le RCW, dès que le nombre de clients COM tombe à 0, le proxy libère sa référence sur l'objet managé qui sera donc collecté par le garbage collector dès que possible. 7

Par exemple, quand un code natif fait appel à un objet COM développé en C#, un CCW est créé. Dans le dépôt Github associé à cet article on trouvera un exemple d'un tel objet COM :

```
[Guid("eb46cc30-3039-446a-9908-d43120168c61")]
    [ClassInterface(ClassInterfaceType.None)]
    [ComSourceInterfaces(typeof(ISaySomething))]
    [ComVisible(true)]
    public class SaySomething : ISaySomething
    {
            public void Greet(string name)
            {
            MessageBox.Show("Hello", name, MessageBoxButtons.OK);
            }
    }
```

Et du client natif associé – le fichier tlb est généré par RegAsm.exe CustomCOMObject.dll /tlb :

```
#import "CustomCOMObject.tlb" named_guids
```



7 *Un CCW maintenu par serveur COM et Process*

```
int main(int argc, char** argv)
{
    CoInitialize(NULL);
    try
    {
        CustomCOMObject::ISaySomethingPtr pSaySomething;
        auto hRes =
pSaySomething.CreateInstance(__uuidof(CustomCOMObject::SaySomething));
pSaySomething->Greet("My name is Bond, Jean-Pierre Bond!");
    }
    return 0;
}
```

Conclusion

L'interopérabilité .Net est un vaste sujet, dont il est impossible de faire le tour dans un article. Le lecteur curieux pourra assouvir sa soif de connaissances avec les livres suivants : « COM and .NET Interoperability » d'Andrew Troelsen et « The Visual C++ Language for .NET » de Gordon Hogenson. •

Sébastien Angelloz-Nicoud,
Techleader JEE Osaxis

JDBI : se faciliter l'utilisation de JDBC

Partie 2

Lorsqu'il s'agit de récupérer des données depuis une base relationnelle et de manipuler les objets correspondants, il est souvent fait mention des ORM (Object Relational Mapping). L'utilisation d'un ORM n'est pas toujours la solution retenue ou la plus adaptée. Dans l'univers Java, la solution la plus directe est l'interface JDBC. Cependant, l'utilisation de cette API de bas-niveau peut s'avérer fastidieuse.

niveau 100

RowMapper

Les « RowMapper » sont de simples classes implémentant l'interface « RowMapper » et ayant une seule méthode « map ». La méthode « map » transforme une ligne de résultats en un objet du type défini par et pour ce mapper.
Exemple avec l'objet « Produit » :
Code complet sur programmez.com

Pour que l'association soit effective, il faut déclarer lors de la configuration de l'instance JDBI que le « ProduitRowMapper » permet de transformer une ligne de résultats en un objet « Produit » :

```
// Inscription du mapper à l'instance JDBI
this.jdbi.registerRowMapper(Produit.class, new ProduitRowMapper());
```

La récupération du produit « soda » peut être réécrite de la manière suivante :

```
// JDBI fait le lien entre le mapping demandé et le mapper associé au type.
Produit soda = this.jdbi.withHandle(handle ->
  handle.createQuery("SELECT ean13, nom, description FROM produit WHERE ean13 = :ean13")
    .bind("ean13", "1234567891011")
    .mapTo(Produit.class)
    .findOnly()
);
```

ColumnMapper

Les « ColumnMapper » sont similaires aux « RowMapper » et permettent de transformer une colonne depuis le type de l'enregistrement de la base de données vers un type Java.
Cependant leur intérêt est plus limité car ils ne peuvent s'appliquer qu'aux requêtes contenant une seule colonne dans leurs résultats. Dans le cas d'une requête ayant plusieurs colonnes, le mapper ne s'appliquera qu'à la première colonne. Les cas d'usages peuvent concerner la conversion de données monétaires ou encore la transformation de valeurs d'énumération stockées sous forme de chaîne de caractères ou numérique.

Récupérer un graphe d'objets

Les jointures sont des opérations basiques en SQL. Mais lorsque qu'il s'agit de transformer un modèle relationnel en graphe d'objets, les problèmes commencent.
JDBI permet de récupérer tout un graphe d'objets à partir d'une seule requête, grâce à un mécanisme de « reduce » et d'un accumulateur. Pour récupérer toutes les listes de courses, avec pour chaque liste, la collection des éléments la composant et les produits associés, on utilise la requête suivante :

```
SELECT lc.id AS lc_id
    , lc.commentaire AS lc_commentaire
    , lc.etat AS lc_etat
    , lci.quantite AS lci_quantite
    , p.ean13 AS p_ean13
    , p.nom AS p_nom
    , p.description AS p_description
FROM liste_course AS lc
INNER JOIN liste_course_item AS lci ON lci.liste_course_id = lc.id
INNER JOIN produit AS p ON p.ean13 = lci.produit_ean13
```

Grâce à une jointure sur la relation « 1..n » entre l'objet « ListeCourse » et l'objet « Produit », la requête va « exploser » toutes les composantes de chaque liste de course avec autant de lignes pour chaque « ListeCourse » que d'éléments la constituant. Grâce à l'utilisation de la méthode « reduceRows » sur le résultat de la requête, il est possible de créer un graphe d'objets complet.

```
List<ListeCourse> listeCourseList = this.jdbi.withHandle(handle ->
  handle.createQuery(listeCourseCompleteSql)
    .registerRowMapper(BeanMapper.factory(ListeCourse.class, "lc_"))
    .registerRowMapper(BeanMapper.factory(ListeCourseItem.class, "lci_")) /* 1 */
    .registerRowMapper(BeanMapper.factory(Produit.class, "p_"))
    .reduceRows(new HashMap<String, ListeCourse>(), (accumulator, rowView) -> { /* 2 */
      ListeCourse listeCourse = accumulator.computeIfAbsent(
        rowView.getColumn("lc_id", String.class),
        id -> rowView.getRow(ListeCourse.class)
      ); /* 3 */

      ListeCourseItem listeCourseItem = rowView.getRow(ListeCourseItem.class); /* 4 */
      listeCourseItem.setProduit(rowView.getRow(Produit.class)); /* 5 */
      listeCourseItem.setListeCourse(listeCourse); /* 6 */

      listeCourse.getItems().add(listeCourseItem);

      return accumulator; /* 7 */
    }).values().stream().collect(Collectors.toList()) /* 8 */
);
```

Voici le détail des différentes étapes du traitement :

- On indique à l'API l'enregistrement d'un mapper générique avec préfixe. Ce préfixe sert à déterminer l'appartenance d'un champ du résultat de la requête à un objet à créer. Ici, tous les champs préfixés par « lc_ » appartiennent à un objet « ListeCourse »
- Cette instruction définit le traitement de « réduction » du résultat de la requête. L'accumulateur est initialisé avec une « Map » stockant les différentes listes de courses avec pour clef leur identifiant.
- La liste de courses correspondante est extraite de la ligne du résultat si inconnue de l'accumulateur, ou récupérée de la map si connue.
- L'item de la liste de courses est extrait de la ligne de résultat.
- Le produit est également extrait de la ligne de résultat puis est assigné à l'item correspondant.
- L'item est ajouté à la liste de courses courante.
- L'accumulateur complété est retourné pour être fourni au parcours des lignes suivantes.
- En fin de traitement, les listes de courses sont récupérées sous la forme d'une collection (ici une liste).

Interface DAO

JDBI possède un plugin nommé « SQLObject » afin de définir les opérations élémentaires des DAO. Les DAO prennent la forme d'interfaces spécifiant des méthodes publiques ainsi qu'une opération SQL associée pour chaque méthode. L'exemple suivant montre comment définir et utiliser une interface DAO :

```
//Exemple de classe DAO manipulable par JDBI
package fr.osaxis.jdbi.dao;
import java.util.List;
import java.util.Optional;

import org.jdbi.v3.core.transaction.TransactionIsolationLevel;
import org.jdbi.v3.sqlobject.config.RegisterRowMapper;
import org.jdbi.v3.sqlobject.customizer.Bind;
import org.jdbi.v3.sqlobject.statement.SqlUpdate;
import org.jdbi.v3.sqlobject.transaction.Transaction;

import fr.osaxis.jdbi.mapper.ProduitRowMapper;
import fr.osaxis.jdbi.modele.Produit;

public interface ProduitDao {

  @SqlQuery("SELECT ean13, nom, description FROM produit WHERE ean13 = :ean13")
  @RegisterRowMapper(ProduitRowMapper.class)
  Optional<Produit> get(@Bind("ean13") String ean13);

  @SqlQuery("SELECT ean13, nom, description FROM produit")
  @RegisterRowMapper(ProduitRowMapper.class)
  List<Produit> list();

  @SqlUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13, :nom, :description)")
  void insert(@Bind("ean13") String ean13, @Bind("nom") String nom,
      @Bind("description") String description);

  @SqlUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13, :nom, :description)")
  void insert(@BindBean Produit produit);

}
```

On retrouve dans la classe précédente les annotations suivantes :

- @SqlQuery et @SqlUpdate : de manière similaire aux méthodes « createQuery » et « createUpdate » de l'objet « Handle », ces deux annotations permettent de déclarer les instructions SQL et leurs types d'opérations (lecture, écriture).
- @RegisterRowMapper : cette annotation permet d'associer le « mapper » nécessaire à la transformation des lignes du « ResultSet » SQL en objets Java.
- @Bind : cette annotation permet de faire le lien entre le paramètre de la méthode et le paramètre de la requête SQL correspondante. Il est possible de se passer de cette annotation si le code source est compilé avec le paramètre « -parameters ». Cette option de compilation a pour effet de conserver le nom des paramètres remplacés habituellement par « arg0 », « arg1 », …, « argn » à la compilation.
- @BindBean : l'annotation permet de faire le lien entre les attributs d'un objet et les paramètres de la requête.

L'utilisation des DAO est possible après l'avoir lié à un objet « Handle » :

```
// Utilisation du DAO Produit pour récupérer un produit par sa clef primaire
Optional<Produit> soda = this.jdbi.withHandle(handle -> {
  ProduitDao produitDao = handle.attach(ProduitDao.class);
  return produitDao.get("1234567891011");
});
```

Externaliser les requêtes SQL

Que l'on utilise les méthodes de l'objet « Handle » ou les interfaces DAO, les requêtes SQL sont écrites directement dans le code.
Il est possible d'externaliser dans des fichiers de ressources toutes les requêtes SQL. Il est même envisageable de modifier les requêtes « à chaud » car JDBI embarque un système de cache invalidant les requêtes un certain temps après le dernier accès.
Pour récupérer simplement une requête grâce à ce mécanisme, on utilise la classe « ClasspathSqlLocator » :

```
//Récupération d'une requête depuis un dossier de ressources.
String requete = ClasspathSqlLocator.findSqlOnClasspath("fr.osaxis.jdbi.modele.Produit.get");
/* Ou de manière similaire */
String requete = ClasspathSqlLocator.findSqlOnClasspath(Produit.class, "get");
```

La requête de l'exemple ci-dessus est récupérée du fichier « get.sql » se situant dans le dossier de ressources « fr/osaxis/jdbi/modele/Produit/ ». Ce dossier doit être accessible depuis le classpath de l'application.
Pour une utilisation avec les interfaces DAO, il suffit d'annoter l'interface avec @UseClasspathSqlLocator :

```
@UseClasspathSqlLocator
public interface ProduitDao {

  /* L'annotation n'a plus besoin de la définition de la requête SQL.
```

```
 */
 @SqlQuery
 /* Elle est récupérée depuis le fichier de ressources
  * fr/osaxis/jdbi/dao/ProduitDao/get.sql
  */
 @RegisterRowMapper(ProduitRowMapper.class)
 Optional<Produit> get(@Bind("ean13") String ean13);

}
```

Transactions

Les méthodes « useHandle » et « withHandle » n'ont pas de notion de transaction (hormis celles implicites, gérées par le SGBD).
Ces deux méthodes permettent simplement d'ouvrir une connexion à la base (ou d'en récupérer une depuis le pool de connexions), et de l'utiliser et de la relâcher (fermeture ou remise à disponibilité du pool).
Pour les besoins transactionnels, l'objet « Handle » propose deux méthodes similaires à « useHandle » et « withHandle » qui sont « useTransaction » et « inTransaction ».

```
// Création d'une liste de courses contenant un nouveau produit, dans le cadre d'une transaction.
String listeCoursId = UUID.randomUUID().toString();

this.jdbi.useTransaction(handle -> {
  handle.createUpdate("INSERT INTO liste_course (id, commentaire, etat) VALUES (?, ?, ?)")
    .bind(0, listeCoursId)
    .bind(1, "Attention, supermarché ferme à 19h30.")
    .bind(2, ListeCourseEtatEnum.OUVERTE.name())
    .execute();

  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13,
:nom, :description)")
    .bind("ean13", "1234567891011")
    .bind("nom", "Soda")
    .bind("description", "Soda rafraîchissant en bouteille")
    .execute();

  handle.createUpdate("INSERT INTO liste_course_item (liste_course_id, produit_ean13,
quantite) VALUES
          (:listeCourseId, :ean13, :quantite)")
    .bind("listeCourseId", listeCoursId)
    .bind("ean13", "1234567891011")
    .bind("quantite", BigDecimal.ONE)
    .execute();
});
```

Ainsi, une erreur lors de l'exécution du bloc annule la transaction et aucune donnée n'est insérée.
Dans le cas de l'utilisation des classes DAO gérées par l'extension SqlObject, une annotation est disponible pour indiquer que l'opération doit s'effectuer dans une transaction :

```
// La méthode « insert » du DAO « Produit » doit s'exécuter dans un contexte transactionnel.
public interface ProduitDao {

 /* ... */

 @Transaction
 @SqlUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13, :nom,
:description)")
 void insert(@Bind("ean13") String ean13, @Bind("nom") String nom,
     @Bind("description") String description);

 /* ... */

}
```

Quelle que soit la méthode utilisée, il est possible d'indiquer le niveau d'isolation de la transaction. Il s'agit d'un paramètre supplémentaire aux méthodes « inTransaction » et « useTransaction » et de l'unique attribut de l'annotation @Transaction.

Autres fonctionnalités

JDBI possède d'autres fonctionnalités comme son intégration avec des librairies tierces. Sous forme de plugins, il faut déclarer leur utilisation lors de la création de l'instance JDBI :

```
// Déclaration de l'utilisation d'un plugin spécifique aux bases de données H2
this.jdbi.installPlugin(new H2DatabasePlugin());
```

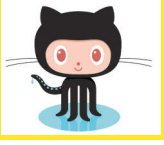
Il existe des plugins spécifiques à quelques moteurs de bases de données (H2, Oracle, PostgreSQL par exemple) afin de profiter de fonctionnalités propres à ces SGBD.
Autre exemple : un plugin Guava est disponible pour l'utilisation directe des collections et structures de données propres à cette librairie.

Le mot de la fin

JDBI est une librairie très complète qui permet de s'abstraire d'opérations fastidieuses par rapport à l'utilisation directe de l'interface JDBC.
Les différentes possibilités de récupération et transformation des données issues de requêtes pour en créer des objets Java en font une boîte à outils très pratique pour ceux qui ne souhaitent pas mettre en place une solution plus lourde comme un ORM.
Son API de bas niveau assure également un contrôle complet sur les paramètres JDBC ainsi qu'un contrôle fin des éléments du requêtage SQL.

Ressources

Site officiel JDBI et documentation : http://jdbi.org
Code source du projet JDBI : https://github.com/jdbi/jdbi



Aurélie Vache
Cloud Dev(Ops) chez Continental
Duchess France & DevFest Toulouse Leader
Toulouse Data Science core-team & Marraine Elles bougent
@aurelievache

Terraform Advanced : jouons avec les data external de Terraform

niveau 100

Avec Terraform, vous pouvez utiliser une data source, afin de récupérer une ressource existante, dans une nouvelle ressource que vous souhaitez créer. Ce concept de data source est puissant et facile à comprendre et à manipuler, mais il est important de savoir que vous ne pouvez pas récupérer tous les types de data source à votre guise. En effet, plusieurs types de data source ne peuvent pas être récupérés avec un filtre, mais plusieurs autres le peuvent.
Par exemple, vous pouvez récupérer un VPC existant par nom de tag :

```
# VPC
data "aws_vpc" "vpc-programmez" {
 filter {
   name = "tag:Name"
   values = ["mon-vpc-pour-programmez"]
 }
}
```

C'est cool, utile, mais malheureusement, ce filtrage n'est possible qu'avec certaines ressources, pas toutes.
Partons du principe que dans notre cas de figure nous devons récupérer un (network) load balancer existant, chez AWS, et que nous utiliserons cette information pour créer une AWS Route 53.

Pour cela, il suffit de :
1. Créer un script qui récupère le network load balancer ;
2. Utiliser le résultat du script dans une data source Terraform, grâce à un external provider ;
3. Utiliser cette donnée spéciale dans la ressource que l'on veut créer.

1. Le Script

mon_module/scripts/get_elb.sh :

```
#!/bin/bash
set -e
kubectl get svc mygateway -n istio-system -o json | jq .status.loadBalancer.ingress[0]
```

Ce script retourne un objet JSON avec une gateway Kubernetes (Istio) (qui est un AWS Load Balancer) :

```
$ ./mon_module/scripts/get_elb.sh
{
"hostname": "a123456789gsfgfsgfsg12134gsgsg78-123456789dfdsf45545fdsf.elb.eu-central-1.amazonaws.com"
}
```

2. La Data Source avec un Terraform External Provider

mon_module/data.tf :

```
data"external""elb"{
   program= ["bash", "${path.module}/scripts/get_elb.sh"]
}
```

3. La définition de la ressource souhaitée

mon_module/aws_r53.tf:

```
resource "aws_route53_record""mymodule_CNAME" {
  zone_id="${data.aws_route53_zone.my_zone.zone_id}"
  name="${var.domain_name}"
  type="CNAME"
  records= ["${data.external.elb.result.hostname}"]
  ttl="${var.route53_ttl}"
}
```

Si ce n'est pas déjà fait, initialisez le workspace Terraform pour ce projet :

```
$ terraform init
```

Et pour finir, appliquer vos ressources Terraform définies afin de créer les ressources.

```
$ terraform apply target=mon_module
```

Tips : vous pouvez vérifier les providers Terraform utilisés dans votre projet et vos modules.

```
$ terraform providers
.
├──── provider.aws ~> 1.24.0
└──── module.mon_module
├──── provider.aws (inherited)
└──── provider.external
```

Félicitations, vous avez créé votre première external data source avec Terraform !

A bientôt pour de nouveaux épisodes.

Jean-Bernard Boichat
physicien retraité, auteur chez Eyrolles
jb@boichat.ch | http://www.boichat.ch/wpjrsp/

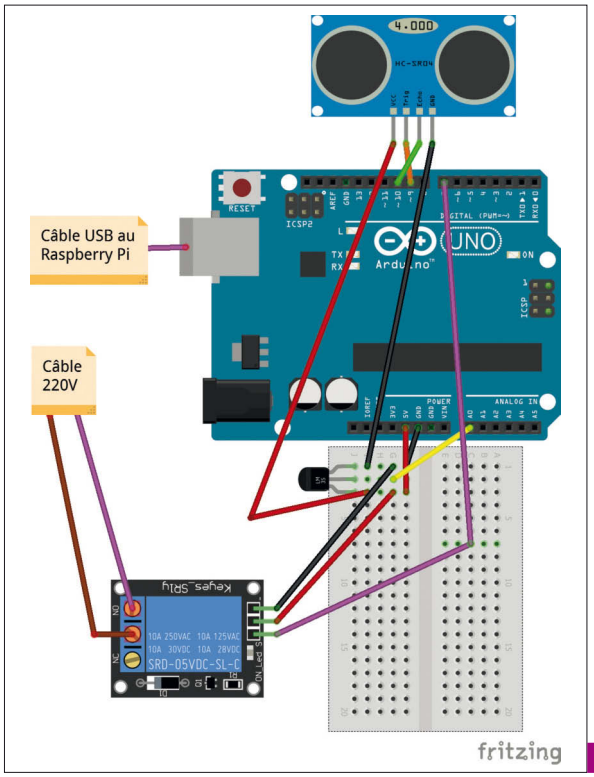
Communication Arduino Raspberry Pi 3 via USB

niveau 200

Lorsque nous bricolons à la fois avec un Arduino et un Raspberry Pi et que nous voulons les faire communiquer, il nous est impossible de faire simple. Il nous faut tout d'abord maîtriser l'IDE de l'Arduino, se débrouiller avec sa programmation en langage C, et ensuite faire l'équivalent sur le Raspberry Pi. Pour un informaticien désirant approfondir ses connaissances, c'est le bonheur absolu, pour un investissement relativement modéré. J'ai essayé ici d'être le plus concis possible dans mon exposé et de ne pas trop entrer dans les détails pour que l'article finisse par remplir un cahier complet de Programmez !

J'ai utilisé l'IDE de l'Arduino uniquement dans sa version PC et non dans sa variante Web avec stockage sur le Cloud. Il est évidemment possible de l'installer sur Ubuntu ou macOS.
Il y a diverses possibilités de communiquer entre un Arduino et un Pi et j'ai choisi ici celle du câble USB. D'autres solutions tout aussi ludiques seraient possibles avec une interface WiFi (Web ou socket), voire au travers d'une interface série RX/TX sur les broches des Pi et Arduino. Une alternative ESP8266 bon marché pour remplacer l'Arduino, programmable avec le même IDE de l'Arduino, est aussi envisageable.

La solution du câble USB est principalement intéressante par sa simplicité. Nous aurons aussi besoin de ce même câble pour communiquer avec le PC lors du développement du logiciel de l'Arduino. Le logiciel du Raspberry Pi sera développé tout d'abord en langage interprété **Python** pour vérifier l'interface : nous sortirons le câble USB du PC pour le connecter au Pi. Ensuite nous passerons en **Java**, avec du code développé sur le PC, avec **Eclipse**, afin de maîtriser le protocole de communication entre l'Arduino et le Pi. Déposer un capteur sur une platine d'expérimentation connectée à l'Arduino et les diverses étapes conduisant au programme Java sur le Pi sont les mêmes techniques et méthodes que décrites dans mon livre, **Programmer en Java pour le Raspberry Pi 3**, sorti chez Eyrolles en janvier 2019. Pour le Raspberry Pi, les capteurs ou autres composants sont normalement connectés à son port GPIO. Mais, dans certains cas, comme le LM35, qui est un capteur de température analogique, il ne pourra pas être utilisé directement sans une carte supplémentaire analogique. Une interface de ce type est directement disponible sur l'Arduino, sur la partie des broches analogiques. Un bricoleur aura peut-être déjà d'autres circuits disponibles, avec du logiciel pour récupérer des mesures particulières ou d'autres résultats. Il lui suffira de piquer les quelques instructions ci-dessous pour les ajouter à du code existant. Ce type d'interface, avec le câble USB, est tout à fait envisageable lorsque les deux Arduino et Pi sont proches. Nous irons chercher dans cet article-ci la distance d'un objet avec un capteur à ultrasons.



1

Schéma Fritzing 1

Le schéma Fritzing qui suit contient trois composants :
- un LM35, un capteur analogique de température ;

et deux composants digitaux :
- un relais ;
- et un capteur à ultrasons.

Un exemple de sketch Arduino suivra avec le capteur à ultrasons qui mesurera la distance à un objet en déplacement pour enclencher le relais suivant la distance. Pour le LM35, la mesure de température ambiante, un autre sketch pourrait être développé par le lecteur, voire par moi-même dans un article à venir. Je l'ai aussi laissé pour montrer que la platine de démonstration est nécessaire lorsque l'alimentation et/ou la terre sont utilisées par plusieurs composants. Le relais peut être branché sur un appareil connecté au réseau électrique domestique (220-230 Volt). Lorsqu'il faudra connecter le relais sur un câble ou une prise 220 Volt, il faudra se

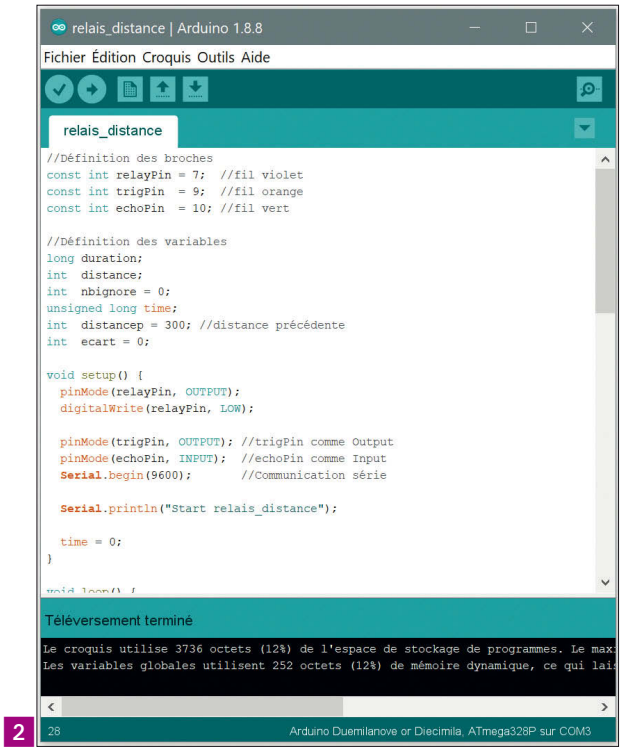
méfier lors du montage en le manipulant correctement et évidemment non connecté. Des exemples de code Python et Java pour ces composants pour le Raspberry Pi se trouvent dans mon livre, mais pas pour le LM35 que j'ai remplacé par un (ou plusieurs) Dallas DS18B20A sur le port GPIO du Pi avec le protocole 1-Wire.
Dans ce diagramme Fritzing (http://fritzing.org/home/) il faudra bien suivre les couleurs pour retrouver les broches. Pour la description de cet article, j'avais construit le circuit avant de faire le diagramme. Il n'est donc pas optimisé pour le côté visuel. Sans le capteur de température, il serait plus dépouillé.
Nous retrouverons la description des broches utilisées dans le sketch Arduino (croquis en français). Le câble USB sera branché au PC lors de la préparation du croquis et pour l'alimenter. Le relais, côté réseau électrique, pourra rester non connecté pendant le développement : des clics sonores significatifs lors des enclenchements et déclenchements suffiront à nous indiquer que cela fonctionne.

Capteur à ultrasons et relais

Pour l'édition, la compilation et le téléversement de croquis, nous devons utiliser l'IDE de l'Arduino, https://www.arduino.cc/en/main/software. J'ai travaillé ici avec la version 1.8.8. Il faudra s'assurer que le *Type de carte* sous *Outils* est correct. Si le type d'Arduino n'est pas visible, il faudra utiliser le Gestionnaire de carte sous la même rubrique du menu. Nous pourrons alors le rechercher et le télécharger. Dans mon cas, j'ai utilisé mon tout vieux Arduino Duemilanove. Le port, COM3 pour moi, apparaîtra sous Windows 10 dans la rubrique Ports (*Explorateur de fichier / PC / Gérer / Gestionnaire de périphérique*). Dans l'IDE, il doit être correctement spécifié sous *Outils Ports* où la liste des ports devrait apparaître de toute façon. En retirant et remettant le câble, le COM à utiliser sera aussi visible et identifiable.
Nous chargerons le fichier **relais_distance.ino**, voire le contenu qui suit, dans l'IDE de l'Arduino :

```
//Définition des broches
const int relayPin = 7;   //fil violet
const int trigPin  = 9;   //fil orange
const int echoPin  = 10; //fil vert

//Définition des variables
long duration;
int distance;
int nbignore = 0;
unsigned long time = 0;

void setup() {
   pinMode(relayPin, OUTPUT);
   digitalWrite(relayPin, LOW);

   pinMode(trigPin, OUTPUT); //trigPin comme Output
   pinMode(echoPin, INPUT);  //echoPin comme Input
   Serial.begin(9600);           //vitesse communication série

   Serial.println("Lancement de relais_distance");
}

void loop() {
  delay(100); //Pas trop souvent, 100ms

  //Clear le trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  //trigPin HIGH state pour 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  //Lecture echoPin pour l'onde sonore retour en microsecondes
  duration = pulseIn(echoPin, HIGH);

  //Calcul de distance
  distance = duration*0.034/2;

  nbignore++; //2 premiers ignorés
  if (nbignore > 2) {
     if (time == 0) {
        if ((distance > 5) && (distance < 100)) {
           //Montre la distance sur le port série
           Serial.print("Distance: ");
           Serial.println(distance);
           Serial.println("Relais ON");
           digitalWrite(relayPin, HIGH);
           time = millis();
        }
     }
     else {
        if ((millis() - time) > 4000) {
           if ((distance <= 5) || (distance >= 100)) {
              Serial.println("Relais OFF");
              digitalWrite(relayPin, LOW);
              time = 0;
           }
        }
     }
  }
}
```

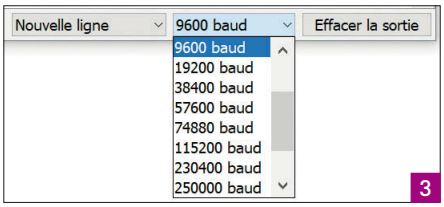
Le nom du fichier du sketch (croquis) est ici **relais_distance.ino** et il doit se trouver dans un répertoire **relais_distance**. Il est conseillé de stocker tous nos croquis Arduino ou ESP8266, dans un répertoire unique. Pour travailler avec l'IDE d'Arduino, je préfère y mettre un raccourci sur le bureau de mon PC, et ensuite je sélectionne le croquis dans le menu : Fichier / Ouvert récemment.
Les fichiers sources de cet article sont disponibles sur le Github de Programmez: https://github.com/francoistonic. 2
Dans cette image, représentant le premier morceau du code de relais_distance dans l'IDE de l'Arduino, nous remarquerons que j'ai déjà cliqué sur le second bouton du menu, en haut à gauche, la flèche direction droite, pour compiler et télécharger le croquis. Si le circuit de l'Arduino est correctement monté, nous entendrons déjà les clics significatifs du relais qui fonctionne. Nous pouvons évidemment juste compiler le croquis pour vérification, sans avoir connecté nécessairement notre Arduino, et ceci avec le premier



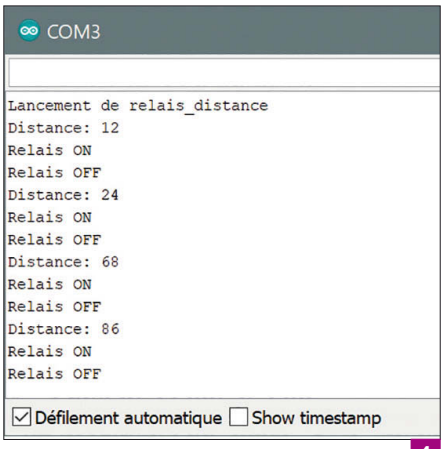
2

bouton V. Dans le chapitre 19 de mon livre sur le Raspberry Pi, dédié à l'utilisation d'un capteur à ultrasons et d'un bouton-poussoir, j'explique la difficulté que j'ai eue, sur un Raspberry Pi, de programmer le script Python pour les tests du capteur avant de passer à la version Java en utilisant le Pi4J. Je n'étais pas certain du bon fonctionnement de mon capteur, j'en ai donc commandé un second et l'ai tout d'abord branché sur un Arduino. Pour la petite histoire, la partie Echo sur le Raspberry Pi a besoin de résistances.
Les instructions clés sont les deux **delayMicroseconds()** qui permettent de générer le signal ultrasonique pendant 10 micro-secondes avant d'enclencher la broche Echo avec **pulseIn(echoPin, HIGH)** qui nous retourne la largeur du signal afin de calculer le temps de réponse.
Le 0.034/2 nous permet de déterminer la distance, avec la vitesse du son et le fait que le signal fait deux parcours (c'est un écho). De plus, nous ignorons les deux premières mesures qui apparaissaient incorrectes durant mes premiers tests, comme si le "moteur du capteur devait chauffer".
Si nous déposons notre capteur à ultrasons simplement sur une table, nous risquons d'avoir des réflexions étranges et aléatoires. Une idée m'est venue en créant par exemple un tube de 7 cm de diamètre à partir d'un carton au format A3 : ce n'est vraiment pas bon du tout. Les ondes partent dans tous les sens et sont réfléchies sur les parois du tube. Il faut donc bien orienter le capteur et libre d'objets aux alentours.
Ce sketch n'est pas grandiose, mais plus ou moins fonctionnel. Il demanderait quelques adaptations suivant des besoins particuliers (nous ferons des exercices). Ici nous activerons le relais pendant 4 secs, lorsque des objets seront en déplacement devant le capteur et entre 5 et 100cm. Le numéro des broches est indiqué en début de sketch. Les deux composants utilisent du 5V et une platine d'expérimentation est donc nécessaire.

Pour exécuter le croquis ci-dessus, nous devons le faire depuis l'IDE avec le menu Outils / Moniteur série. Si nous utilisons différents croquis ou micro-contrôleurs avec d'autres configurations, il faudra corriger la vitesse, soit dans le croquis, ici Serial.begin(9600); , soit dans l'IDE dans la fenêtre d'exécution : 3
Une mauvaise vitesse se verra rapidement par des caractères illisibles dans la fenêtre d'exécution. Cette vitesse n'a rien à voir avec la vitesse de téléchargement que j'ai toujours spécifiée au maximum, avec mon câble USB relativement court.
Voici donc l'exécution du script, où j'ai fait passer un objet, mon fameux tube, en dessus du capteur, plusieurs fois, et à une distance variable entre 12 et 86cm. 4
Le lecteur pourra déjà constaté que les messages des Serial.println("") du croquis sont les mêmes pour montrer le résultat dans cette fenêtre du moniteur série de l'IDE que ceux qui seront reçus par un script Python ou un programme Java : c'est une communication série sur le port et le câble USB.



3



4

Exercice 1 pour le lecteur : modifier le sketch Arduino afin que la variable **time** retrouve ses 4 secs, mais seulement si un déplacement significatif, disons 10%, a été constaté durant la phase ON du relais.

Exercice 2 pour le lecteur : remplacer l'Arduino par un ESP8266, par exemple un RobotDyn WiFi D1 R2 à 4$. Ce dernier possède moins de broches, mais nous aiderait dans notre apprentissage, où nous pourrions plus tard, par exemple, envoyer la distance avec un protocole de communication socket et non avec le câble USB. Il faudra modifier les broches du sketch Arduino pour utiliser les broches correctes de cet autre modèle de microcontrôleur. Par exemple ici :

```
//Définition des broches du RobotDyn D1 R2
const int relayPin = 16; //D0
const int trigPin  = 2;  //D4
const int echoPin  = 14; //D5
```

Il faudra évidemment corriger la position des fils sur le schéma Fritzing (et pourquoi pas installer ce joli outil sur notre PC pour jouer avec). Si nous essayons de télécharger le script Arduino ci-dessus, sans adaptations, nous pourrions nous retrouver avec un microcontrôleur totalement bloqué. Un reset serait nécessaire : brancher le GPIO 0 (D3) au GND et presser le bouton Reset. Retirer ce dernier pont et le câble USB, le remettre et refaire une compilation et un téléchargement (pour ce RobotDyn mettre le modèle de carte à WeMos D1 R2 & mini).

Préparation sur le Raspberry Pi

Avant de passer au code Java que nous développerons et vérifierons sur un PC Windows et avec Eclipse, nous allons vérifier la communication entre l'Arduino et le Raspberry Pi avec un script Python. J'ai utilisé le **Raspberry Pi 3 Model A+**, le dernier né des Pi (novembre 2018), puissant et bon marché, avec un Raspbian (le

Linux du Pi) installé sans interface écran, donc avec un accès depuis le PC Windows avec l'outil de communication ssh **Putty**. Les instructions d'installation et d'utilisation sont expliquées dans mon livre, voire sur de nombreux sites Web. Une simple "Raspberry Pi Putty" comme recherche Google nous indiquera par exemple aussi que **ssh** est l'alternative Linux.
Si nous avons un Raspberry Pi avec un Raspbian récent, la commande dans une console du Pi

```
sudo apt-get install python-serial
```

n'est pas nécessaire. Si nous l'exécutons tout de même, nous recevrons une indication que le logiciel Python, pour communiquer avec le port USB, est déjà installé.
Ensuite il faudra déterminer avec la commande Linux **ls** quel périphérique (**/dev**) est utilisé. Nous voyons ici que nous nous trouvons dans le répertoire **/home/pi/python** que j'ai créé pour y déposer mes scripts Python :

```
pi@raspberrypi:~/python $ ls /dev/tty*
```

Nous recevrons quelque chose comme :

```
.....
/dev/tty10 /dev/tty19 /dev/tty27 /dev/tty35 /dev/tty43 /dev/tty51 /dev/tty6 /dev/ttyprintk
/dev/tty11 /dev/tty2 /dev/tty28 /dev/tty36 /dev/tty44 /dev/tty52 /dev/tty60 /dev/ttyUSB0
/dev/tty12 /dev/tty20 /dev/tty29 /dev/tty37 /dev/tty45 /dev/tty53 /dev/tty61
.....
```

Nous essayerons la commande Linux **cat** sur ce device **/dev/ttyUSB0** qui nous semble être le bon :

```
pi@raspberrypi:~ $ cat /dev/ttyUSB0
Lancement de relais_distance

Distance: 11

Relais ON

Relais OFF
^C
pi@raspberrypi:~/python $
```

Le message **Lancement de relais_distance** est bien le premier **println** de notre script Arduino ci-dessus. Un **Ctrl-C** va interrompre la commande Linux cat.

Un script Python sur le Raspberry Pi

Il est difficile de faire plus simple pour ce script Python qui suit. Les programmeurs Python expérimentés, que je ne suis pas, en feront vite une extension.
Nous éditerons le script **arduino_usb1.py** directement sur le Pi avec un éditeur, par exemple **vi** pour moi, voire **nano,** et dans un répertoire comme par exemple **/home/pi/python** (attention à l'indentation du code) :

```
# coding: utf-8
import time
import serial

ser = serial.Serial('/dev/ttyUSB0', 9600)

while 1 :
    print(ser.readline())
    time.sleep(0.1)
```

Et il fonctionne comme un **cat** sous Linux. Dans une console Putty depuis notre PC :

```
pi@raspberrypi:~/python $ python arduino_usb1.py
Lancement de relais_distance
Distance: 75
Relais ON
Relais OFF
```

Exercice 3 pour le lecteur : modifier le script Python pour que l'interruption avec un Ctrl-C fonctionne correctement. De plus, n'afficher aucun des messages reçus, comme les ON et OFF. A partir de la ligne "Distance" reçue de l'Arduino, conserver les distances minimum et maximum et montrer la dernière distance mesurée suivie du minimum et du maximum (par exemple : objet détecté à 75cm (min 12 et max 99)).

Une classe Java pour communiquer avec l'Arduino

Pour cette partie il faudra maîtriser Eclipse et la programmation Java. Je ne vais ici ni expliquer l'installation d'Eclipse, ni son utilisation, ni sa configuration. La version 8 de Java est utilisée, car c'est celle supportée sur le Raspberry Pi 3.
Nous allons à présent écrire une jolie petite classe Java **USBPort**, dans un projet Java sous Eclipse et sur le même PC Windows utilisé pour l'IDE de l'Arduino. Cette classe USBport fera un travail similaire que l'exemple Python et pourrait être étendu à souhait :

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class USBPort implements Runnable {
    private Thread monThread;
    private RandomAccessFile rAF;
    private String portName;

    public USBPort(String portName) {
        this.portName = portName;
        monThread = new Thread(this);
    }

    public void start() {
        try {
            rAF = new java.io.RandomAccessFile(portName, "rwd");
        }
        catch (Exception e) {
            System.out.println("start " + e.toString());
        }

        monThread.start();
    }

    public void run() {
```

```
        System.out.println("Lecture du port " + portName);
        for (int i = 0; i < 10; i++) {
            String response = null;
            try {
                response = rAF.readLine();
            }
            catch (IOException e) {
                System.out.println(e.getMessage());
            }
            System.out.println(response);
        }

        System.out.println("USBPort terminé (run)");
        System.exit(0);
    }

    public static void main(String[] args) {
        String portDevice = "COM3";
        if (args.length > 0) {
            portDevice = args[0];
        }

        USBPort usbPort = new USBPort(portDevice);
        usbPort.start();

        try {
            Thread.sleep(60000); //1 minute
        }
        catch (InterruptedException e) {
        }

        System.out.println("USBPort terminé");
        System.exit(0);
    }
}
```

La partie communication se fait dans un thread (**Runnable**) et il faudra se méfier qu'il ne reste pas en suspend lors d'un débogage sous Eclipse. Une explication complète de ce code occuperait à nouveau trop de place ici. Le lecteur devrait pouvoir se débrouiller.

Nous rebrancherons notre câble USB, cette fois-ci sur le PC. Le port COM3 dans mon cas a été identifié avec l'IDE de l'Arduino et dépend du connecteur du PC utilisé. En ayant testé l'interface COM avec l'IDE, la bonne configuration sera mise. Si aucun résultat n'est retourné, il faudra adapter correctement le port COM correspondant.
mode COM3 dans une fenêtre DOS CMD va nous fournir l'information et une commande mode COM3 baud=9600 pourrait être nécessaire. Nous avions indiqué la vitesse avec l'instruction Serial.begin(9600); dans le sketch.
Si nous exécutions **java USBport** dans le répertoire **bin** de notre Workspace d'Eclipse, avec

```
D:\EclipseWorkspace.....\bin>java USBPort COM3
```

nous verrions le même résultat que dans la console d'Eclipse de ce projet Java. Par exemple :

```
Lecture du port COM3
Lancement de relais_distance
Distance: 13
Relais ON
Relais OFF
```

COM3 peut être passé par le **args[]** du main() de Java.
Exercice 4 pour le lecteur : adapter le code Java pour vérifier les cas d'erreurs avec le COM3 inexistant ou en utilisation (par exemple actif sur l'IDE de l'Arduino). Ne pas limiter à 10 messages et aussi coder l'exercice 3 précédent en Java (objet détecté à 75cm (min 12 et max 99)).

Exécuter USBPort sur le Raspberry Pi

Comme décrit dans mon livre à plusieurs occasions, nous transférerons le fichier USBPort.class, depuis le répertoire bin du projet dans le Workspace d'Eclipse, sur le Raspberry Pi dans un répertoire comme **/home/pi/java** et avec **WinScp**. Il faudra alors faire passer le câble USB du PC sur le Raspberry Pi, évidemment! C'est donc le Pi qui alimentera à présent l'Arduino. Magnifique !
La commande ici sera :

```
java USBPort /dev/ttyUSB0
```

avec le /dev précédemment déterminé. Le résultat sera le même que ci-dessus avec la commande **cat** ou le script Python **arduino_usb1.py**.
En cas de difficulté avec la configuration du port, la commande **stty**, équivalente à MODE sous DOS, pourra être utilisée. Par exemple :

```
sudo stty -F /dev/ttyUSB0 -a
sudo stty -F /dev/ttyUSB0 9600 cs8 -cstopb -parenb
```

Conclusion

Il nous restera à améliorer notre script Arduino, faire quelque chose d'un peu plus intelligent, comme envoyer des données dans l'autre direction et pareil en Java sur ce /dev/ttyUSB0. Le Raspberry Pi pourrait envoyer et adapter les deux 5 et 100cm, et l'Arduino les garder en EEPROM !
Nous pourrions par exemple ajouter une LED rouge, voire un buzzer, sur le GPIO du Pi pour indiquer si le relais est actif sur l'Arduino ou encore stocker chaque passage de notre chat dans une base de données SQLite et envoyer par courriel une statistique journalière avec les heures et les distances.
S'il fallait identifier si le chat s'éloignait ou non du capteur, il faudrait alors le faire sur l'Arduino. Enfin, pour un programmeur expérimenté et passionné de Java comme moi, il préférera déplacer le capteur à ultrasons sur le GPIO du Raspberry Pi, le vérifier avec Python et le programmer avec une multitude de fonctionnalités diverses en Java. En considérant les sujets présentés dans cet article, comme les langages C, Python et Java, l'incontournable Eclipse, les outils Windows et Linux (Raspbian), des circuits et composants électroniques, les protocoles et outils de communication, le lecteur se rendra vite compte que le Raspberry Pi 3 est définitivement un environnement exceptionnel de formation informatique et professionnelle. •

Vincent Rivière
Développeur à l'Université Paris 1 Panthéon-Sorbonne
http://vincent.riviere.free.fr/
https://www.youtube.com/c/Vretrocomputing

Les successeurs de l'Atari ST

niveau 100

L'Atari ST était un ordinateur extrêmement populaire à la fin des années 80. Ce fut le premier modèle d'une longue série, et des machines compatibles continuent à apparaître encore aujourd'hui.

Ah, l'Atari ST ! Voilà une machine qui a marqué les esprits. Au milieu des années 80, les micro-ordinateurs familiaux sont déjà bien installés dans les foyers. Ce sont des machines faciles d'accès, qui permettent aussi bien de découvrir la programmation que de s'éclater sur les jeux vidéo. Mais ces premiers ordinateurs sont équipés de microprocesseurs 8 bits dont la puissance est limitée. Alors que les premiers PC sont peu adaptés au multimédia, et que les Macintosh sont hors de prix, l'apparition de l'Atari ST permet d'introduire les ordinateurs 16 bits dans les familles. « Power without the Price », dit le slogan. Autrement dit : la puissance sans le prix.

1985 : Atari ST

C'est le premier modèle. Comme la plupart des ordinateurs de l'époque, l'Atari ST se présente sous la forme d'un boîtier monobloc : unité centrale et clavier intégré. On peut le brancher sur un moniteur couleur, ou sur un téléviseur équipé d'une prise péritel. Deux modes vidéo sont accessibles avec ces moniteurs. La basse résolution (320x200, 16 couleurs parmi 512) est principalement utilisée par les jeux. La moyenne résolution (640x200, 4 couleurs) est plutôt utilisée par les utilitaires. Mais il est aussi possible d'utiliser un moniteur monochrome tel que l'Atari SM124 pour bénéficier de la haute résolution (640x400, noir et blanc, 71 Hz) avec une taille et une qualité d'image inégalées. Ce mode est évidemment très prisé par les professionnels, car il est idéal pour profiter des utilitaires. L'Atari ST dispose en standard d'une souris (ce qui n'était pas forcément une évidence en ces temps-là). Il permet d'utiliser des joysticks compatibles avec la célèbre console Atari 2600, le standard de l'époque. Côté audio, l'ordinateur est équipé d'un générateur de sons assez basique sur 3 voies. Mais là où l'Atari ST se distingue particulièrement, c'est sur sa puissance brute : un processeur Motorola 68000 cadencé à 8 MHz, et 512 Ko de RAM, extensibles à 4 Mo. Du jamais vu à cette époque, surtout à prix abordable. Côté stockage, ce premier modèle utilise un lecteur de disquettes 3"1/2 externe, simple face. Il dispose par ailleurs d'un port disque dur ACSI (interface propriétaire Atari). Mais la grande originalité de cette machine, c'est d'être équipée de prises MIDI qui permettent de connecter claviers et synthétiseurs. Cette particularité fera le bonheur des musiciens, et contribuera grandement au succès de cet ordinateur. Le système d'exploitation de l'Atari ST s'appelle TOS (The Operating System). Il utilise des disquettes compatibles MS-DOS, et une interface graphique conviviale appelée GEM : bureau, icônes, fenêtres, tous les éléments modernes sont déjà là. Le TOS réside entièrement en ROM (à part sur les tout premiers modèles où il était chargé depuis une disquette). L'Atari ST original est équipé du TOS 1.0.

1986 : Atari STf

L'Atari STf est la première évolution de l'Atari ST. Il ressemble au modèle d'origine, avec quelques centimètres supplémentaires en profondeur. L'alimentation et le lecteur de disquettes ont été intégrés à l'unité centrale. Et au passage, ce dernier est passé en double face. Ce modèle a été extrêmement populaire, c'est le plus connu de toute la gamme.

520 ou 1040 ?

Le nombre mentionné dans la référence de l'ordinateur indique approximativement la quantité de RAM installée d'origine. Ainsi, un Atari 520 ST est équipé par défaut de 512 Ko de RAM, alors qu'un Atari 1040 STe dispose d'1 Mo de RAM.

source : https://en.wikipedia.org/wiki/Atari_ST

source : https://fr.wikipedia.org/wiki/Atari_ST

1987 : Atari Mega ST

Il s'agit d'un STf amélioré, dans un boîtier plus professionnel de type pizza box (« boîte à pizza », un desktop de faible hauteur). Le clavier est séparé de l'unité centrale. L'ordinateur dispose d'un blitter pour accélérer l'affichage, ainsi que d'une horloge pour conserver l'heure même après avoir coupé l'alimentation. La machine est équipée de 1, 2 ou 4 Mo de RAM, ainsi que du TOS 1.2. Le Mega ST était souvent accompagné d'un disque dur MegaFile de 20 ou 30 Mo qui se glissait sous l'unité centrale.

PORTABLES

Atari a aussi commercialisé des versions portables du ST : le Stacy (1989) et le ST Book (1990). Ces machines sont relativement rares.

1989 : Atari STe

Le STe est une évolution significative du STf, dans un boîtier identique. Le blitter est fourni en standard. La palette graphique passe de 512 à 4096 couleurs. Le circuit vidéo est amélioré, permettant écrans virtuels et scrollings. Un second processeur sonore appelé « DMA soundchip » est ajouté, permettant de jouer des sons digitalisés en stéréo, ainsi qu'un mixer pour régler le volume, les graves et les aigus. Deux ports joystick supplémentaires font leur apparition, ils permettront plus tard de connecter des joypads Jaguar. Le STe est équipé du TOS 1.6. Ce modèle était techniquement meilleur que les ST/STf, mais malheureusement ses capacités furent peu exploitées à l'époque.

MOTOROLA 68000

Le processeur Motorola 68000 était très populaire au milieu des années 80. Puissant et agréable à programmer, il a équipé beaucoup de machines : Apple Lisa, Macintosh, Atari ST, Amiga, SEGA Mega Drive, Neo Geo, et de nombreuses bornes d'arcade.

1990 : Atari TT030

Il s'agit d'une évolution majeure, à destination des professionnels, en boîtier pizza box élargi pour accueillir un disque dur. Il dispose de 2 Mo de RAM en standard, extensible à 10 Mo. Cette RAM polyvalente est renommée ST-RAM pour l'occasion. Et pour la première fois, le processeur est remplacé par un modèle plus puissant : le TT est équipé d'un 68030 cadencé à 32 MHz. Ce processeur fonctionne intégralement en 32 bits, cela lui permet de gérer un nouveau type de mémoire plus rapide appelée TT-RAM. Des extensions de 16 à 256 Mo sont disponibles. Un coprocesseur arithmétique (FPU) est inclus en standard. Le TT bénéficie des améliorations du STe : son stéréo, scrolling… De plus, il dispose d'un port VME pour brancher des cartes graphiques ou des cartes réseau, et d'un port SCSI. Les modes graphiques sont améliorés : apparition du mode VGA 640x480 en 16 couleurs, 320x480 en 256 couleurs, et même 1280x960 en monochrome avec un moniteur spécial. Bref, à sa sortie le TT était la Rolls-Royce des machines Atari, très prisé par les professionnels, notamment pour les applications gourmandes en puissance. Pour l'occasion, le TOS passe directement en version 3.

ST/TT

Le sigle « ST » signifie « Sixteen/Thirty-two », autrement dit : « Seize/Trente-deux ». Cela fait référence à l'architecture du processeur 68000 : les traitements sont effectués sur 32 bits en interne, alors que le bus de données externe a une largeur de 16 bits seulement. En revanche, « TT » signifie « Thirty-Two », en référence au 68030 qui équipe cette machine et qui travaille intégralement en 32 bits.

1991 : Atari Mega STe

Le Mega STe ressemble à s'y méprendre au TT, avec le même boîtier pizza box large. D'ailleurs, ces deux machines sont sorties presque en même temps. Mais alors que le TT est une évolution majeure, le Mega STe est juste une extension du STe pour lui conférer une dimension plus professionnelle. Les nouveautés sont toutefois appréciables. Le processeur reste un 68000 comme sur l'ancienne gamme, mais pour la première fois on peut doubler sa fréquence par logiciel : 8 ou 16 MHz. En même temps, 16 Ko de mémoire cache font leur

Atari Mega ST

source : http://www.computinghistory.org.uk/det/28563/Atari-Mega-ST-2/

Atari TT

source : https://fr.wikipedia.org/wiki/Atari_TT

apparition. Un coprocesseur arithmétique est installable en option. De plus, certaines évolutions du TT sont incluses, comme le port VME, les ports série RS-232 supplémentaires, le port réseau local, et le support des lecteurs de disquettes haute densité (1,44 Mo). Le Mega STe est équipé de 2 ou 4 Mo de RAM, et du TOS version 2 (qui est très proche du TOS 3 du TT).

1992 : Atari Falcon 030

L'arrivée du Falcon marque incontestablement un changement de génération. Et pourtant, cette machine cache bien ses atouts. Son boîtier monobloc ressemble comme deux gouttes d'eau à celui des STf/STe. Seul le clavier intégré se démarque, avec ses touches gris foncé. Mais à l'intérieur, tout a été amélioré. Le processeur est un Motorola 68030, comme sur TT, et il est cadencé à 16 MHz. Un coprocesseur arithmétique est disponible en option. 1 ou 4 Mo de ST-RAM sont installés d'origine, extensibles à 14 Mo. Et surtout, de nombreuses nouveautés apparaissent : disque dur IDE 2"1/2 interne, port SCSI-2, connecteur LAN RS422... Mais le plus marquant, ce sont les composants vidéo et audio qui ont été complètement repensés. Le nouveau circuit graphique s'appelle VIDEL. Il permet des modes entrelacés pour doubler le nombre de lignes sur les moniteurs couleur classiques. Mais surtout, il supporte les moniteurs VGA. Ainsi, il rend accessibles tous les modes vidéo du ST sur n'importe quel moniteur, plus les nouveaux modes spécifiques au Falcon. De 2 à 256 couleurs en mode bitplane, et même jusqu'à 65536 couleurs simultanées en mode 16 bits True Color. Une vraie révolution. Le son n'est pas en reste : un nouveau circuit audio fait son apparition, avec support stéréo numérique 16 bits 50 KHz, en sortie comme en entrée. Cerise sur le gâteau : ce circuit audio est couplé à un puissant DSP Motorola 56001 à 32 MHz, qui permet d'effectuer des traitements sonores complexes en temps réel. Le Falcon permet ainsi d'enregistrer des musiques en « direct to disk », autrement dit, directement vers le disque dur. C'était une fonctionnalité inédite à l'époque, très appréciée par les musiciens. Toutes ces nouveautés ont fait du Falcon un ordinateur d'exception.


Atari Mega STe

source : https://fr.wikipedia.org/wiki/Atari_Mega_STE

source : https://fr.wikipedia.org/wiki/Atari_Falcon030

1993 : La fin ?

Atari souhaite se concentrer sur sa nouvelle console 64 bits, la Jaguar, et arrête la production de ses micro-ordinateurs. En bref, dans les années qui suivent, la société est rachetée par JTS Corporation, puis par Hasbro Interactive, et finalement par Infogrames qui finit par se renommer… Atari. Mais la carrière officielle du ST et de ses descendants est bel et bien terminée.

Heureusement, l'histoire ne s'est pas arrêtée là ! Car même si Atari a abandonné ses ordinateurs, la communauté a continué de les utiliser… et a même fabriqué de nouveaux modèles.

On peut commencer par citer les nombreux composants additionnels qui ont été développés par des sociétés ou des passionnés au fil des années : extensions mémoire, adaptateurs clavier et souris, émulateurs de disquettes et disques durs à partir de cartes SD ou CompactFlash… Et surtout, des cartes accélératrices pour augmenter significativement la puissance des machines. Les modifications passent généralement par de l'overclocking ou un changement de processeur.

SYSTÈMES D'EXPLOITATION

Le TOS (The Operating System) est le système d'exploitation de l'Atari ST et de ses successeurs. EmuTOS est une alternative libre et compatible. FreeMiNT est un noyau multitâche optionnel qui étend les fonctionnalités du TOS sous-jacent.

1995 : Medusa T40, puis Hades 040/060

Ce sont les premiers ordinateurs compatibles Atari qui sont sortis sur le marché. Ils ont été conçus et fabriqués par la société suisse Medusa Computer Systems (Fredi Aschwanden). Le T40 est basé sur un processeur Motorola 68040 à 64 MHz. Il dispose de 128 Mo de RAM, avec possibilité d'extension. Un bus ISA est présent pour connecter une carte graphique. Un port IDE est disponible en standard, et une carte SCSI peut être ajoutée en option.

Par la suite, le T40 a été remplacé par l'Hadès, avec processeur 68040 ou 68060. Ce nouveau modèle dispose de 4 emplacements PCI et d'une carte graphique ATI Rage Pro. Il est équipé de 32 Mo de RAM en standard, extensibles à 1 Go. Il est livré avec un clavier de PC. Ces machines sont généralement fournies dans un boîtier moyenne tour. Les ordinateurs Medusa sont de sérieux concurrents pour l'Atari TT. En fait ils sont même plus puissants. Ils sont fournis avec un TOS 3 modifié. Plusieurs cartes additionnelles sont supportées : DSP, Ethernet, SCSI…

1998 : Milan 040/060

Ces ordinateurs sont produits par la société MILAN-Computersystems GbR (Uwe Schneider). L'objectif est d'utiliser des composants de PC à bas coût pour produire un système compatible Atari. La carte mère est construite autour d'un processeur Motorola 68040 cadencé entre 25 et 33 Mhz. Elle supporte de la RAM EDO jusqu'à 512 Mo. Des ports ISA, PCI et IDE sont présents. La particularité du Milan est d'inclure des composants modernes, mais différents du matériel original d'Atari. Fatalement, la compatibilité logicielle en pâtit. Le Milan est fourni avec un TOS 4 fortement modifié. Les utilitaires qui se contentent de faire

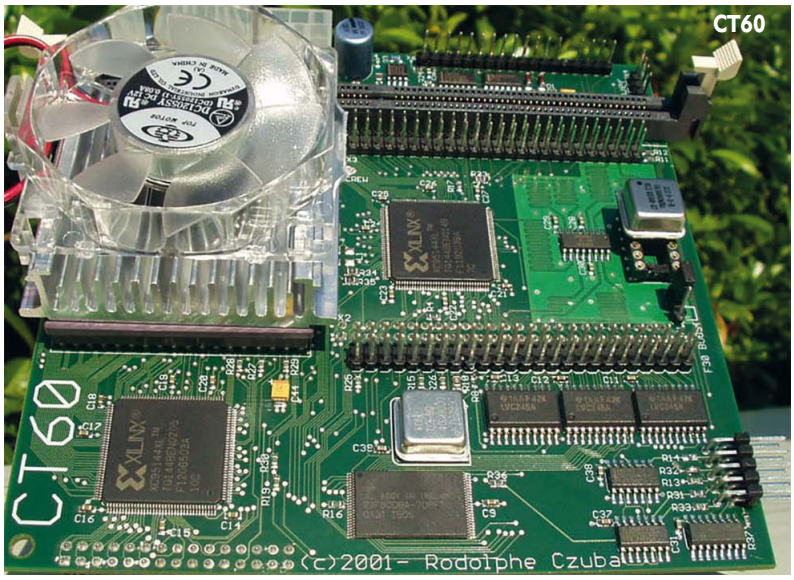
appel au système d'exploitation fonctionnent vite et bien. En revanche, les jeux et autres logiciels qui tentent d'accéder directement au hardware Atari marchent rarement. Le Milan reste une très bonne machine pour utiliser les applications GEM.

2002 : Carte CT60 pour Falcon

Si on ne devait citer qu'une seule carte accélératrice, ce serait bien celle-ci. Conçue par Rodolphe Czuba, elle propulse le Falcon dans une dimension supérieure. Le 68030 est remplacé par un 68060 fréquencé à 66 MHz, overclockable jusqu'à 100 MHz. Les 16 MHz du Falcon d'origine font pâle figure en comparaison. La carte permet aussi l'ajout de FastRAM à partir de barrettes de SDRAM PC133. Cela permet des extensions mémoire allant jusqu'à 512 Mo, une capacité considérable pour cette machine. Un port d'extension est inclus, il permettra par la suite d'ajouter des cartes filles comme le SuperVidel (carte graphique) ou la CTPCI (pour connecter des cartes PCI, notamment ATI Radeon). Plus tard, la CT63 viendra succéder à la CT60. Ces cartes accélératrices sont incontournables pour tout possesseur de Falcon.

FPGA

Les FPGA (Field-Programmable Gate Array) sont des circuits intégrés qui contiennent des milliers de cellules logiques programmables. En les combinant, ils permettent de réaliser des opérations logiques complexes, et même de simuler le comportement de composants électroniques existants. On les programme avec des langages spécifiques tels que VHDL et Verilog. Les FPGA sont reprogrammables à volonté.



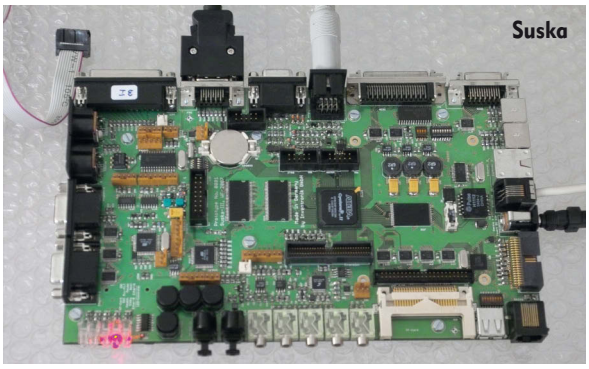
source : http://powerphenix.com/ct60/french/present.htm

2006 : Suska

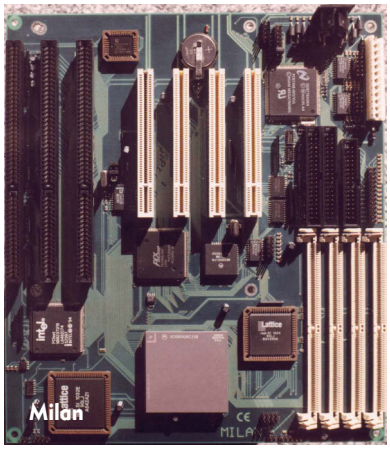
La carte Suska est le fruit de l'incroyable projet de Wolfgang Förster. L'idée est simple : implémenter un Atari ST complet dans un FPGA (voir encadré). Mais la tâche est ardue : comment s'y prendre ? L'Atari ST renferme de nombreux composants électroniques. Wolfgang a décidé de progresser pas à pas à partir d'un vrai Atari. Pour chaque composant, il a extrait la puce d'origine, et il l'a remplacée par un FPGA qu'il a programmé pour remplir la même fonction. Cela lui a permis de débugger chaque composant un par un, y compris le 68000 et les custom chips. Puis il les a tous réunis dans un même FPGA, sur une carte indépendante qu'il a appelée Suska. Ce travail titanesque lui a demandé plusieurs années. Il a par la suite créé d'autres modèles, en réduisant la taille de la carte et en ajoutant de nouveaux ports. Ce projet démontre l'incroyable potentiel des FPGA : ils renferment une telle quantité de portes logiques programmables qu'il est possible de simuler des composants très complexes, jusqu'à un ordinateur tout entier.

2011 : FireBee

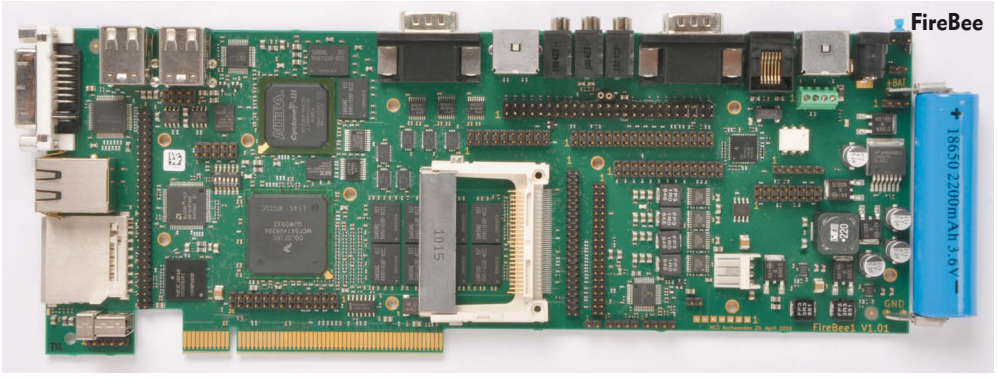
Le FireBee est l'aboutissement du projet ACP (Atari Coldfire Project). Il s'agit d'un or-



source : http://www.experiment-s.de/en/



source : http://www.uweschneider.de/en/projects_milan.php

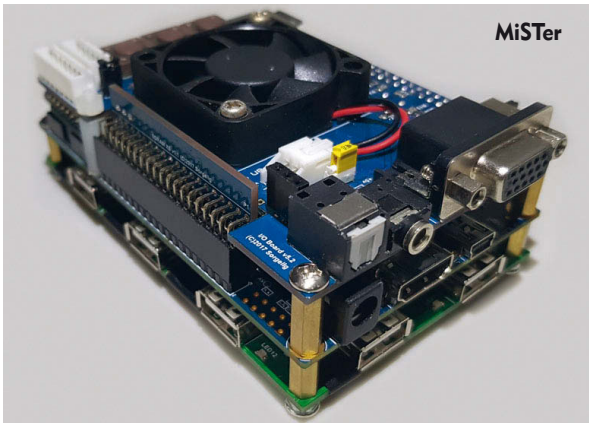


source : http://firebee.org/fb-bin/page?label=press_pictures&lng=FR

dinateur compatible Atari Falcon, à base de processeur ColdFire 264 MHz et de FPGA. Il est construit par Medusa Computer Systems (Fredi Aschwanden). Cette idée était dans l'air depuis des années. Elle est partie d'un constat : le dernier processeur Motorola de la famille 68000 à avoir été produit fut le 68060 à 75 MHz (parfois overclockable à 100 MHz). Par la suite, la société Freescale (nouveau nom de Motorola Semiconductor) est repartie de zéro avec la gamme ColdFire pour un usage industriel. En gros, l'architecture ColdFire reprend les bases du 68000, mais en plus simple, plus moderne et plus rapide. Le jeu d'instructions est simplifié : certaines instructions et modes d'adressage du 68000 sont purement et simplement abandonnés sur ColdFire. En contrepartie, la fréquence est augmentée, et les fonctionnalités avancées comme la mémoire cache et la MMU sont incluses. En fait, le ColdFire est bien plus qu'un CPU : c'est un microcontrôleur qui renferme de nombreuses interfaces : PCI, DDRAM, Ethernet, RS-232, timers… La carte FireBee inclut en plus de nouveaux composants : port CompactFlash/IDE, circuit graphique HD, sortie DVI, ports USB, circuit audio AC'97, batterie rechargeable, et la quasi-totalité des interfaces Atari d'origine. Le FPGA intervient pour la compatibilité matérielle : il reprend presque tous les composants Atari ST du projet Suska (sauf le processeur), et ajoute les composants du Falcon (VIDEL…).

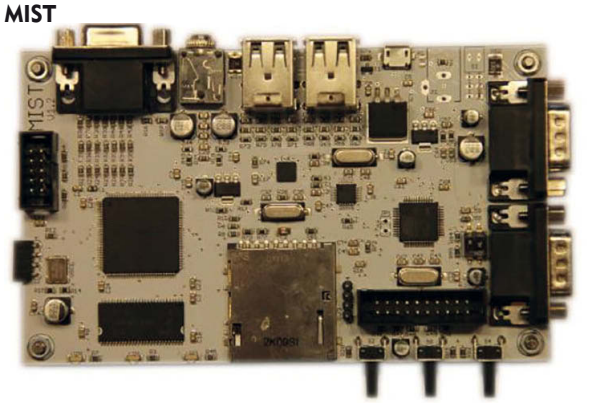
Le système d'exploitation principal est FireTOS : il s'agit du TOS 4 modifié par Didier Méquignon. En fait, c'est une variante du TOS modifié de la CT60, du même auteur. Il émule les instructions 68060 manquantes, supporte les nouvelles interfaces (comme les clavier/souris USB, Ethernet…) et les nouveaux modes graphiques jusqu'en Full HD. Au final, tout cela fonctionne étonnamment bien. Toutefois, la compatibilité est mitigée, pour plusieurs raisons. L'émulation 68060 est bonne, mais imparfaite, car une poignée d'instructions se comporte différemment sur ColdFire. Par ailleurs, la simulation des composants du Falcon par le FPGA est incomplète : il manque le DSP et le blitter, le support audio est partiel, et le VIDEL souffre de quelques bugs. Mais si on choisit minutieusement les bons logiciels, on se retrouve avec la machine compatible Atari la plus rapide de tous les temps, avec en prime des modes graphiques modernes.

MiSTer



source : https://github.com/MiSTer-devel/Main_MiSTer/wiki

MIST



source : https://github.com/mist-devel/mist-board/wiki/TheBoard

2013 : MIST

La carte MIST est développée par Till Harbaum (MasterOfGizmo). Tout comme la carte Suska, elle a été conçue autour d'un FPGA pour simuler un Atari ST complet. Mais elle s'est aussi révélée capable de simuler un Amiga. Et petit à petit, de nouveaux « cores » sont apparus, permettant de simuler de nombreuses machines des années 80/90 : ordinateurs 8 bits (Atari 800, Commodore 64, Amstrad CPC…), ordinateurs 16 bits (Atari ST, Amiga, Macintosh…), mais aussi consoles de jeux (Nintendo NES & Game Boy, SEGA Master System & Mega Drive…) et même quelques bornes d'arcade. Vous l'aurez compris, la carte MIST est la plus versatile des cartes FPGA. C'est une très bonne alternative aux solutions logicielles telles que les émulateurs. A noter que la plupart des cores sont écrits en langage VHDL et distribués sous licence libre.

2017 : MiSTer

Le projet MiSTer est développé par Alexey Melnikov (Sorgelig). L'idée principale est de faire fonctionner les cores du MIST (et donc l'Atari ST) sur du matériel plus standard, plus puissant, et moins cher. C'est la carte Terasic DE10-nano qui a été choisie, à laquelle peuvent se joindre en option des cartes filles. On dispose donc du matériel suivant : FPGA Altera Cyclone V, processeur ARM Cortex A9 @ 800 MHz, sortie HDMI, 1 Go de RAM DDR3… Bref, cette carte a une réserve de puissance très confortable. Il est donc possible de simuler encore plus de machines, telles que la Super Nintendo, l'Amiga 1200 et de nombreuses bornes d'arcade. Tous ces avantages font que le MiSTer jouit d'une grande popularité.

En 2018, un nouveau palier a été franchi par Jorge Cwik (ijor) avec son core FX CAST. Ce dernier simule un Atari ST complet au cycle près. Cela fait du MiSTer la machine compatible Atari ST la plus fidèle de tous les temps. Même les jeux et démos fonctionnent à merveille. A noter que l'auteur a extrait la partie 68000 de son projet, appelée FX68K, et l'a publiée sous licence GPLv3.

EMULATEURS

Steem émule les Atari ST/STe, et Hatari émule en plus les TT et Falcon. ARAnyM est un cas à part : il émule partiellement un Falcon, mais propose de nombreuses fonctionnalités supplémentaires comme des modes vidéo étendus et des accès disques plus performants.

Conclusion

Comme nous venons de le voir, de nombreuses machines ont vu le jour après l'Atari ST. D'abord ses successeurs officiels, puis des machines construites par d'autres sociétés ou même des passionnés. Toutes ces machines ont une compatibilité variable avec l'Atari ST d'origine. Et encore, je ne vous ai pas parlé de certaines machines exotiques qui sont très différentes de l'Atari ST mais qui permettent pourtant de faire fonctionner ses programmes.

Que retenir de tout cela ? Si on veut écrire des programmes Atari qui fonctionnent sur un maximum de plateformes, il faudra tenir compte de toutes ces machines et de leurs différences. Pas de panique, en règle générale, si on respecte le système d'exploitation, tout se passe très bien. Mais ça, ce sera pour une autre fois. •

François Tonic

µKenbak-1 : la résurrection d'un ordinateur précurseur

Remontons le temps à 1971, lorsqu'une étrange machine électronique est proposée à la vente : le Kenbak-1. Aujourd'hui, il est considéré comme le premier ordinateur que l'on pouvait acheter et « utiliser », une révolution. Mais il ne s'agit pas d'un micro-ordinateur, car le microprocesseur n'était pas encore sorti.

Cette machine, créée par John Blankenbaker, ne possédait donc pas de processeur. Le tout premier microprocesseur est l'Intel 4004. Il contient un circuit intégré pour la logique de calculs. La machine visait surtout l'éducation et était vendue à 750 $.

Cette première tentative n'a pas réellement pris car seulement 40 exemplaires du Kenbak-1 seront vendus, sur la cinquantaine d'unités produites. Et posséder un exemplaire authentique relève du miracle.

µKenbak-1 : une belle réplique !

Chris Davis et Adwater&Stir avaient déjà réalisé la superbe réplique de l'Altair 8800, le premier micro-ordinateur américain : l'Altair-duino. Nous l'avions présenté dans Programmez!.

µKenbak-1 comprend une PCB sur laquelle, nous trouvons en façade les boutons et les leds. Il respecte rigoureusement la disposition originelle. Pour animer l'ensemble, Chris a utilisé un Atmel Atmega318p, un microcontrôleur très répandu.

Pour la partie logicielle, µKenbak-1 s'appuie sur le code développé par Mark Wilson pour son projet Kenbak-uino. Le projet est open source :

https://github.com/funnypolynomial/Kenbakuino

Mark avait créé un prototype du Kenbak-1 en 2011 :

http://funnypolynomial.com/software/arduino/kenbak.html

Le manuel de référence de 1971 :

https://www.adwaterandstir.com/wp-content/uploads/2018/10/KENBAK-Programming_Reference.pdf

Le Kenbak-1 propose une interaction limitée : 17 interrupteurs et 12 LEDs. Vous l'aurez compris, les interactions sont restreintes ainsi que les opérations possibles. Ainsi, les opérations basiques consistent à entrer des valeurs avec les commutateurs 8 bits de gauche (numérotés 0 à 7). Comme le précise la documentation, « nos adresses et données sont en 8 chiffres où chaque chiffre est un 0 ou un 1. ». Bref, nous sommes en pur binaire.

Le bouton CLR permet de tout effacer. STOR permet de stocker la valeur affichée. DISP affiche l'adresse courante. READ lit cette adresse courante. Chaque byte est adressable. Le registre A est le registre primaire de la machine.

Pour entrer une valeur :

- On rentre l'adresse de destination avec les boutons puis on appuie sur SET.
- On appuie sur CLEAR pour effacer l'affichage
- On entre la donnée à stocker (à l'adresse indiquée plus haut)
- On appuie sur Store. On stocke la donnée rentrée dans l'adresse mémoire.

Plusieurs extensions ont été rajoutées au Kenbak d'origine.

Pour commander la machine :

https://www.adwaterandstir.com/product/kenbak-1/

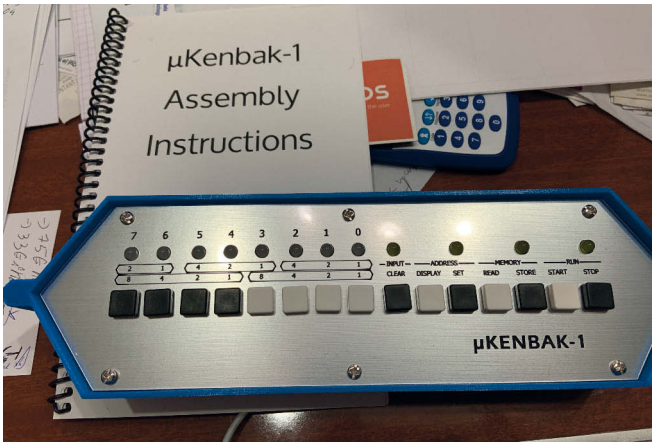
- 65 $ pour la version en kit à assembler, livré avec l'adaptateur européen
- 80 $ pour la version assemblée (et testée)

+ 24 $ de frais de port vers la France

Attention : il s'agit d'un prix promo, le prix normal du kit est de 75 $

Si vous optez pour le kit, il faudra un peu de patience et une bonne dose de concentration pour souder l'ensemble des composants et des sockets… Temps nécessaire : 4-5 heures.

C'est une jolie expérience à faire et la qualité du kit est de très bon niveau.



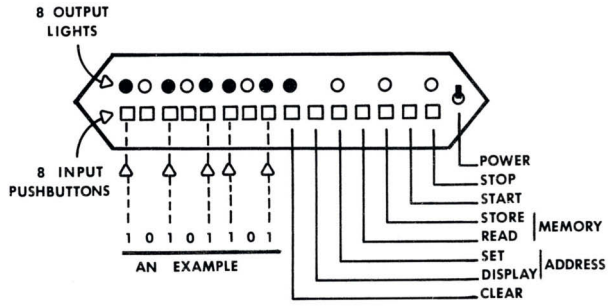


Figure 1.2 The Front Panel of the Computer

LES BOUTONS & LAMPES

Les lampes (leds) du haut sont les affichages de sorties (=résultat). Les boutons du bas permettent de rentrer les valeurs, par exemple : 10101101.

Par rapport au modèle de 71, nous n'avons pas de bouton POWER à droite.

Partie RUN

STOP : arrête l'exécution du programme

START : exécute le programme

Partie Memory

STORE : stocke le nombre entré

READ : lit le nombre entré à l'adresse

Partie ADDRESS

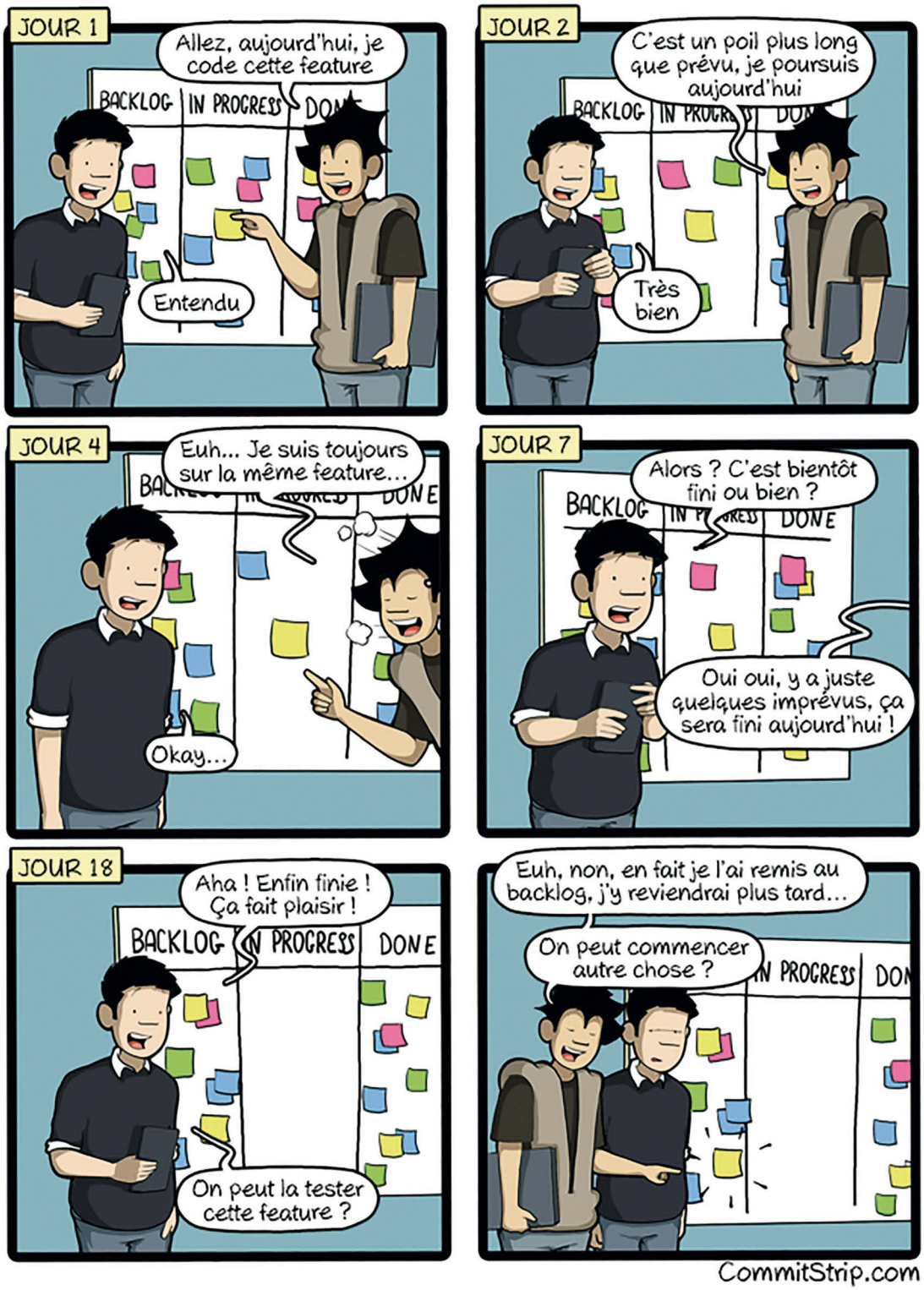
SET : stocke l'entrée dans le registre d'adresse

DISPLAY : affiche la valeur du registre d'adresse

Zone INPUT

CLEAR : on efface tout

La danse du kanban



Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : ZDnet, S. Saurel
Nos experts techniques : V. Loquet, M. Eveillard, L. Avrot, H. Iqbal, A. Giretti, V. Fabing, D. Duplan, C. Pichaud, T. Bollet, V. Koukponou, J-P Gervasoni, S. Angelloz-Nicoud, A. Vache, J-B Boichat, V. Rivière, CommitStrip
Maquette : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, mars 2019

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information

- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ? *2 sites répondent à votre profil*
- La cybersécurité vous concerne ? *Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc*
- L'emploi, les salaires, les formations, les offres vous intéressent ? *Le site sur l'Emploi dans le numérique est à votre disposition*

www.solutions-numeriques.com

VENEZ GOÛTER NOS DERNIÈRES MOUTURES DE CODE !

NOS PROCHAINES SESSIONS DE FORMATION SUR PARIS

WEB ET MOBILE
Asp.Net Core 2.2 (C#) : 14-17 Mai & 17-20 Septembre
NodeJS : 20-23 Mai & 16-19 Septembre
Angular 7 : 15-19 Avril & 24-28 Juin & 23-27 Septembre
Flutter : 17-19 Juin & 23-25 Septembre

BIG DATA ET DEEP LEARNING
TensorFlow : 23-26 Avril & 10-13 Juin & 23-26 Septembre
Cassandra : 15-17 Avril
Kafka : 20-22 Mai & 9-11 Septembre
Spark Machine Learning : 10-12 Juin

DEVOPS
Kubernetes/Docker : 3-5 Juin & 23-25 Septembre
Kubernetes Avancé : 3-5 Juillet

Avec le code « PROG! », un mug collector *Programmers Fuel* vous sera offert pour toute inscription avant septembre 2019.

AMBIENT·IT
Organisme de formation ultra-spécialisé pour les développeurs

128 boulevard Macdonald
75019 PARIS

01 40 34 11 53

formation@ambient-it.net

https://www.ambient-it.net/prochaines-sessions/

Financement possible via le plan de formation de votre entreprise (OPCO) ainsi qu'en AIF (Aide Individuelle à la Formation) pour les demandeurs d'emploi.