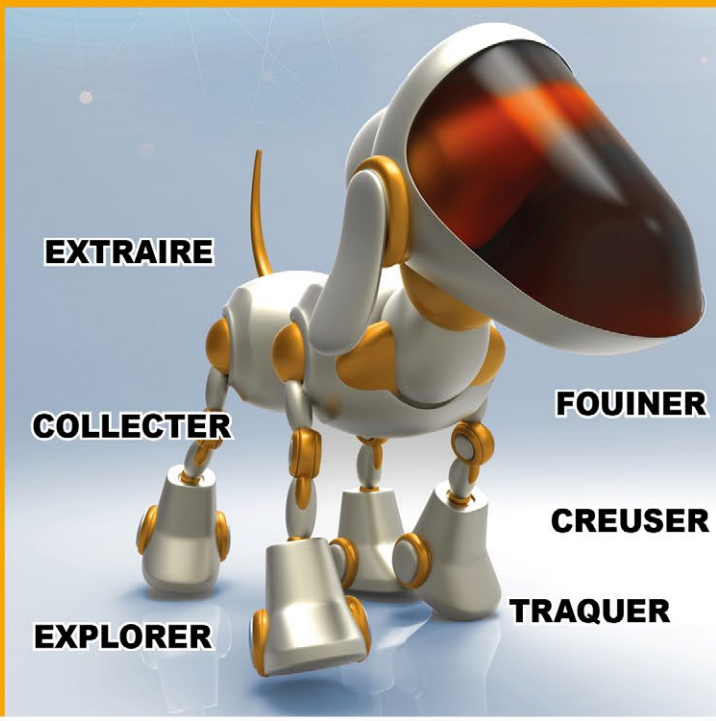
N°199 décembre 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

CONFIDENTIALITÉ / WEB CRYPTOGRAPHY API

Chiffrez les données de vos utilisateurs côté client p.68



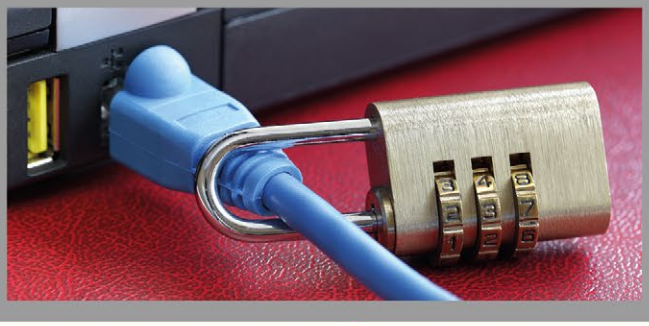
MOTEUR DE RECHERCHE

CRÉEZ UN ROBOT QUI FOUILLE LE WEB POUR VOUS ! p.58

- Développez votre web crawler
- Implémentez le moteur d'indexation

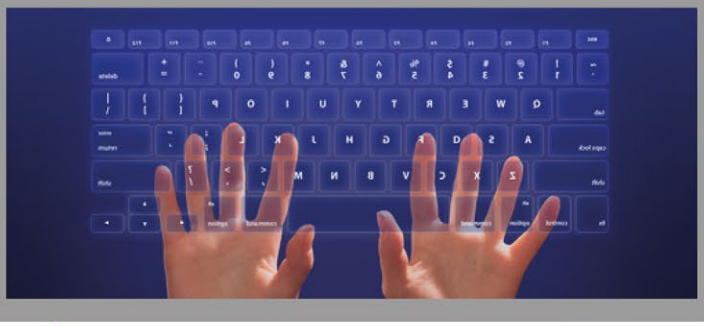
SYSADMIN / SÉCURITÉ

Dissimulez vos services avec le port knocking p.32



SONARQUBE / QUALITÉ LOGICIELLE

Adoptez des règles de codage et vérifiez leur application ! p.14



PYTHON / ALGO

Intégrez des icônes dans vos QR codes p.22

TYPESCRIPT / COMPILATION

Codez enfin proprement en JavaScript p.74

PANDOC / FORMAT

Convertissez vos fichiers grâce à des extensions Lua p.50

ET AUSSI : *Nouveautés PHP 7.1 - Virtualisation avancée avec Proxmox VE - Le moteur de templates Twig*

L 19275 - 199 - F: 7,90 € - RD

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

SERVEURS DÉDIÉS EN PROMOTION
QUANTITÉ LIMITÉE* À PRIX EXCEPTIONNEL

ikoula HÉBERGEUR CLOUD

OFFRES SANS ENGAGEMENT DE DURÉE

SERVEURS	CPU	PROCESSEUR	RAM	DISQUE DUR	SETUP (3)	PRIX HT/MOIS
A LA UNE **Green G460**	Intel® Celeron® G460 HT	1 CPU (1C/2T) @1,8 GHz	8 Go DDR3	2 To SATA (2)	OFFERT	~~39,99 €~~ 12,99 €
A LA UNE **Crazy Fish**	Intel® Xeon® 3000	1 CPU (2C/2T) @1,86 GHz min.	4 Go DDR2	2 To SATA (5 400 tr/min)	OFFERT	~~29,99 €~~ 14,99 €
TOP VENTE **IKX-G620**	Intel® Pentium® G620	1 CPU (2C/2T) @2,6 GHz	8 Go DDR3	1 To SATA	19 €	~~39,99 €~~ 9,99 €
TOP VENTE **IKX-Core i3**	Intel® Core™ i3 2100T HT	1 CPU (2C/4T) @2,5 GHz	8 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 12,99 €
TOP VENTE **IKX-Core i5**	Intel® Core™ i5 2400S	1 CPU (4C/4T) @2,5 GHz	16 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 17,99 €
IKX-Core i7	Intel® Core™ i7 2600	1 CPU (4C/8T) @3,4 GHz	16 Go DDR3	1 To SATA (2)	19 €	22,99 €
IKX-3430	Intel® Xeon® Quad Core 3430	1 CPU (4C/4T) @2,4 GHz	8 Go DDR3 (1)	2 x 1 To SATA (2)	39 €	~~189,99 €~~ 25,99 €
Green i5	Intel® Core™ i5 Quad Core 3450	1 CPU (4C/4T) @3,1 GHz	32 Go DDR3	2 To SATA (2)	29 €	25,99 €
TOP VENTE **Green i7**	Intel® Core™ i7 Quad Core 3770 HT	1 CPU (4C/8T) @3,4 GHz	32 Go DDR3	2 To SATA (2)	29 €	34,99 €
IKX-1220L	Intel® Xeon® E3-1220L HT	1 CPU (2C/4T) @2,2 GHz	32 Go DDR3	2 x 1 To SATA (2)	39 €	~~129,99 €~~ 34,99 €
IKX-R410	Intel® Bi-Xeon® Quad Core 5000	2 CPU (4C/8T) @2 GHz	32 Go DDR3	4 x 1 To SATA Raid 1 Hard (2)	49 €	~~429,99 €~~ 109,99 €
IKX-R710	Intel® Bi-Xeon® Quad Core E5520 HT	2 CPU (4C/8T) @2,26 GHz	32 Go DDR3 (1)	6 x 1 To SATA Raid 1 Hard (2)	49 €	~~499,99 €~~ 189,99 €
IKX-R910	Intel® Quad-Xeon® Hexa Core E7540 HT	4 CPU (6C/12T) @4 GHz	64 Go DDR3 (1)	6 x 300 Go SAS Raid 1 Hard 2,5 (2)	49 €	~~569,99 €~~ 249,99 €

SYSTÈMES LINUX : debian CentOS ubuntu

*Serveurs dédiés disponibles en quantité limitée et sous réserve de disponibilité sur le site : express.ikoula.com/serveur-dedie#promo

(1) Possibilité d'augmenter le niveau de RAM.

(2) Possibilité d'augmenter la taille du / des disque(s) ou d'avoir une alternative SSD / SAS et RAID HARD.

(3) Frais de setup OFFERTS dans le cas d'un engagement annuel non cumulable avec les promotions en cours.

TOUTES LES PROMOTIONS SUR : EXPRESS.IKOULA.COM

ikoula HÉBERGEUR CLOUD

sales@ikoula.com

01 84 01 02 50

GNU/Linux Magazine France
est édité par Les Éditions Diamond

10, Place de la Cathédrale - 68000 Colmar - France
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 – v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

En version papier et PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITORIAL

À quoi sert le Web si l'on ne peut pas le parcourir et trouver les informations que l'on souhaite ?

Aujourd'hui, il suffit d'ouvrir un moteur de recherche (bien souvent Google) et de taper sa requête pour obtenir une réponse pertinente en quelques secondes... pour peu que l'on sache construire la requête correctement. Le fait d'obtenir cette information paraît naturel mais, pour les plus vieux d'entre nous (ou les moins jeunes devrais-je dire), rappelez-vous qu'il y a quelques dizaines d'années nous utilisions des annuaires (Yahoo ! étant le premier d'entre eux). Excite introduisit ensuite les bases des véritables moteurs de recherche. Ces moteurs prirent alors l'aspect de portails et se livrèrent une guerre sans merci. Les sociétés qui proposaient ces moteurs perdirent peu à peu de vue le fait que les utilisateurs souhaitaient se focaliser sur la recherche et elles multiplièrent les sources de distraction (puisque nous sommes sur un portail) de manière à engranger un maximum de revenus publicitaires. De plus, la recherche n'était pas particulièrement efficace... et c'est alors qu'apparût Google, qui écrasa tous les autres.

De nos jours peu de moteurs résistent encore à l'hégémonie de Google. On peut citer par exemple DuckDuckGo, Qwant ou encore Bing. Chacun fera alors son choix en estimant la pertinence des réponses. Notons également la présence de méta-moteurs : des sites qui interrogent différents moteurs de recherche et qui agrègent les réponses. Certains de ces méta-moteurs sont un peu particuliers : ils n'interrogent qu'un seul moteur, mais permettent d'assurer la confidentialité de vos recherches (comme Startpage).

Derrière tous les moteurs de recherche se trouve la même mécanique : parcourir le Web, collecter des informations et les indexer en fonction de critères propres à chaque moteur. Dans ce numéro, nous vous proposons de revenir sur les bases des moteurs de recherche, de créer votre Web Crawler pour l'envoyer fouiller les pages web à la recherche des informations qui vous intéressent. Vous pourrez en quelque sorte créer vos propres espions du Web numériques : lancez vos robots le soir et récupérez vos données au petit matin. Parcourez les pages à la recherche d'erreurs, de liens morts, étudiez l'évolution des sites dans le temps... devenez le maître du Web (à ponctuer par un rire guttural :-)).

Pour finir, comme vous l'aurez sans doute remarqué, le numéro 200 approche à grands pas (ainsi que la nouvelle année, comme cela avait été calculé au lancement du magazine... si, si :-)). Nous avons décidé de nous servir de ce numéro un peu spécial pour marquer le coup en vous proposant une nouvelle formule qui se veut un peu plus proche de l'esprit des débuts. De nouvelles rubriques verront le jour. Lesquelles ? Comment ? Vous le saurez le mois prochain...

Tristan Colombo

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°87 !

PLONGEZ AU COEUR DE VOTRE SYSTÈME GNU/LINUX

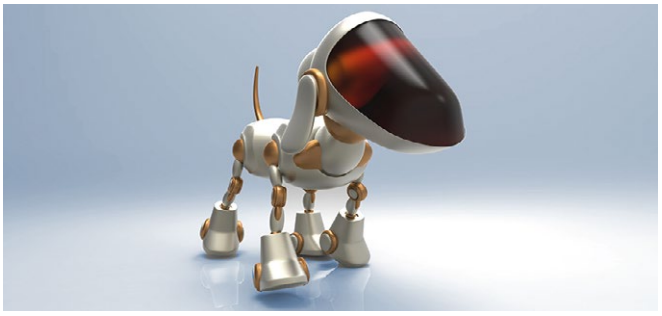
NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°199



DÉCOUVREZ LES NOUVEAUTÉS DE PHP 7.1

Stéphane MOUREY - Mousse sur le Seeraiwer

PHP 7.1 est arrivé. Il ne s'agit pas d'une version majeure maquillée comme nous y avait habitué PHP 5, mais il apporte tout de même de nombreuses améliorations, renforçant la consistance du langage et améliorant le confort du développeur.

Mots-clés : PHP, type, closure, tableaux, classes, itérable

Résumé

Les nouveautés les plus importantes de PHP 7.1 concernent des adaptations suite à l'introduction des typages des paramètres et des valeurs de retour des fonctions, la possibilité d'intercepter plusieurs classes d'exception à l'aide d'une seule instruction `catch`, la généralisation du support des index négatifs sur les chaînes de caractères, les syntaxes liées aux tableaux, la création de fermetures à partir de tout élément *Callable*.

PHP 6 s'est fait tellement attendre avant d'être annulé que chaque version mineure de PHP 5 représentait une véritable version majeure. Maintenant que PHP 7 a enfin vu le jour, il n'en est plus ainsi et PHP est maintenant en mesure d'évoluer plus sereinement. PHP 7.1 est arrivé et si ce n'est pas une révolution, il amène tout de même son lot de nouveautés. Voyons ce qu'il en est.

1 Types éventuellement nuls

Cela sonne mieux en anglais : *Nullable types*. De quoi s'agit-il ?

PHP 7 a introduit la possibilité de typer les paramètres et les valeurs de retour des fonctions et des méthodes, ce qui permet une amélioration des performances et un code plus cohérent. Toutefois, la syntaxe adoptée ne permet pas d'indiquer qu'un paramètre pourrait être éventuellement nul. Le problème pouvait être contourné en ce qui concerne les paramètres en indiquant une valeur par défaut nulle. Mais du coup, l'information concernant le type attendu, si le paramètre est non nul, ne pouvait figurer.

Pour la valeur de retour d'une fonction, le problème est un peu différent : il est fréquent qu'une fonction ne renvoie pas de valeur si elle n'est pas parvenue à produire de résultat, ce qui contrevient à l'indication du type de retour. Une solution est de renvoyer une valeur du même type assimilable à `null` par transtypage, comme une chaîne vide si une chaîne de caractères est attendue. Mais cela peut poser quelques problèmes, par exemple lors des comparaisons strictes, avec l'opérateur `===` par exemple.

Avec PHP 7.1, il est désormais possible d'indiquer qu'un paramètre ou qu'une valeur de retour peuvent être nuls, tout en précisant le type s'il ne l'est pas. Il suffit de faire précéder l'indication du type par un point d'interrogation :

```
function test(?int $entierOuNul) : ?string {
  return is_null($entierOuNul)?null:(string)$entierOuNul;
}
```

Attention, cela ne remplace pas une valeur par défaut : avec la fonction que nous venons d'écrire, un appel sans paramètre provoquera une erreur :

```
test(); // erreur !
test(null); // là, cela fonctionnera
```

Par ailleurs, cette nouveauté a également des conséquences sur l'héritage. Si une classe a une méthode dont la

valeur de retour peut être éventuellement nulle, une classe fille peut supprimer cette éventualité, mais l'inverse n'est pas possible. Au contraire, pour un paramètre de méthode, l'éventualité peut être ajoutée, mais pas supprimée.

```
interface test2 {
  function alpha(int $entier) : ?int;
}
interface test3 extends test2 { // cette interface est valide
  function alpha(?int $entier) : int;
}

interface test4 {
  function beta(?int $entier) : int;
}
interface test5 extends test4 { // cette interface n'est pas valide
  function beta(int $entier) : ?int;
}
```

2 Retour de type void

En suivant une préoccupation similaire, il est maintenant possible d'indiquer qu'une fonction ne renvoie aucune valeur de retour. Il faut pour cela indiquer le type **void** :

```
function test6() : void {}
```

Le terme **void** a été préféré à **null**, bien que le résultat soit équivalent, car l'usage en est répandu en général, en particulier en **C** et dans la documentation PHP.

L'intérêt de cette innovation est d'indiquer explicitement qu'une fonction ne renvoie pas de valeur, qu'il s'agit de son fonctionnement normal et que cela ne correspond pas à une erreur.

3 Catch multiple

PHP 7.1 apporte une modification intéressante dans la gestion des exceptions. Quel est le problème ? PHP intercepte les exceptions en fonction de leurs classes et de leurs héritages, toutes devant étendre en premier la classe **Exception** native. Il est ainsi possible de différencier leur traitement en s'appuyant sur cette donnée, en allant du plus fin au plus général. Mais il se peut que deux exceptions demandent le même traitement sans partager de parents autre que l'exception native. Avec PHP 7.0, il est nécessaire d'écrire le code de ce traitement deux fois :

```
try {
  // votre code
} catch (ExceptionType1 $e) {
  // traitement de l'exception
} catch (ExceptionType2 $e) {
  // copie du même traitement !
} catch (Exception $e) {
  // traitement de toutes les autres exceptions
}
```

Bref, nous obtenons là du code redondant, et c'est mal ! PHP7.1 permet de l'éviter en permettant l'interception de plusieurs types d'exception dans le même bloc **catch**, en les séparant par une barre verticale **|**. L'exemple précédent peut ainsi s'écrire :

```
try {
  // votre code
} catch (ExceptionType1|ExceptionType2 $e) {
  // un seul traitement pour deux types d'exception
} catch (Exception $e) {
  // traitement de toutes les autres exceptions
}
```

Voilà qui est tout de même mieux !

4 Support généralisé des index négatifs sur les chaînes de caractères

Nombreuses sont les fonctions sur les chaînes de caractères acceptant une valeur négative pour indicateur de position. Dans ce cas, PHP calcule la position demandée en allant à rebours depuis la fin de la chaîne. Cela se révèle parfois bien pratique, à ceci près que certaines fonctions ne le supportent pas. Du coup, le développeur, même chevronné, doit se référer trop souvent à la documentation pour savoir s'il peut utiliser cette facilité. Par exemple, la fonction **strrpos** qui recherche la dernière occurrence d'une chaîne dans une autre accepte un paramètre optionnel indiquant depuis quelle position commencer la recherche. Cet argument peut être négatif, **-2** signifiant que la recherche doit commencer à partir de l'avant-dernier caractère. Par contre, la fonction **strpos** qui effectue la recherche de la première occurrence accepte également un tel argument, mais uniquement positif. Il y a là une inconsistance du langage, maintenant réparée.

Les accès à un caractère individuel grâce aux notations **[]** et **{}** ont même été étendus pour le supporter également :

```
$str='abcdef';
var_dump($str[-2]); // => string(1) "e"

$str{-3}='.';
var_dump($str);  // => string(6) "abc.ef"
```

Voilà un nouveau gain de confort.

5 Visibilité des constantes de classes

Les seuls membres d'une classe sur lesquels le développeur ne pouvait pas indiquer de visibilité étaient les constantes. Celles-ci étaient toujours publiques. La principale conséquence en est une confusion possible entre les constantes destinées à un usage interne à la classe et celles destinées à être utilisées depuis l'extérieur. Il convenait donc de permettre une encapsulation correcte de ces constantes. C'est maintenant chose faite :

```
class MaClasse {
  private const MA_CONSTANTE_PRIVE = 0;
  protected const MA_CONSTANTE_PROTEGEE = 0;
  public const MA_CONSTANTE_PUBLIC = 0;
}
```

L'indication de la visibilité est optionnelle, sa valeur étant publique par défaut. À noter que les interfaces ne supportent que des constantes publiques.

6 Améliorations concernant les tableaux

6.1 La fonction list

La fonction **list** est utilisée pour recevoir dans une liste de variables les éléments d'un tableau :

```
$arr = [1,2,3];
list($a,$b,$c) = $arr; // $a égal 1, $b égal 2, $c égal 3
```

Jusqu'ici, cette syntaxe ne considérait les valeurs du tableau que selon leur index numérique, et donc leur ordre d'entrée dans le tableau en ignorant complètement les clés des tableaux associatifs. Or, justement, l'intérêt de ces derniers est qu'en nommant les clés, on peut ignorer l'ordre d'entrée des valeurs.

Pour remédier à cela, la fonction **list** permet d'adresser maintenant les valeurs du tableau en utilisant leurs clés :

```
$arr = array('c'=>3,'a'=>1,'b'=>2);
list('c'=>$c,'b'=>$b,'a'=>$a) = $arr; // $a égale 1, $b égale 2,
$c égale 3
```

6.2 Support des clés sur la syntaxe avec crochets

PHP supporte la création de tableaux en utilisant deux syntaxes différentes : soit par l'utilisation du mot-clé **array**, soit en utilisant des crochets **[]**. Mais jusqu'ici, seule la première permettait de créer un tableau associatif. Ce n'est plus le cas avec PHP 7.1 :

```
$arr = ['k1'=>1,'k2'=>2,'k3'=>3];
```

6.3 Syntaxe avec crochets pour la fonction list

Afin d'homogénéiser la syntaxe de la fonction **list** avec la création de tableau, il est maintenant possible d'utiliser les crochets pour réaliser la même opération :

```
['c'=>$c,'b'=>$b,'a'=>$a] = $arr;
```

est équivalent à :

```
list('c'=>$c,'b'=>$b,'a'=>$a) = $arr;
```

Cela fonctionne également pour un tableau indexé, il suffit de ne pas préciser les clés :

```
[$a,$b,$c] = $arr;
```

est équivalent à :

```
list($a,$b,$c) = $arr;
```

Remarquez la clarté de cette syntaxe pour les boucles récursives :

```
foreach ($arr as ['a' => $a, 'b' => $b]) {
  var_dump($a,$b);
}
```

7 Apparition du pseudo-type itérable

Les tableaux sont itérables, c'est-à-dire que leurs valeurs peuvent être parcourues par itérations successives, avec l'instruction **foreach** par exemple. PHP avait introduit l'interface **Traversable** pour permettre à des objets de devenir eux-mêmes itérables, de la même façon que les tableaux. Depuis que PHP 7 a ajouté la spécification optionnelle des types lors des appels de fonctions, il n'existait aucun moyen d'indiquer qu'une fonction pouvait recevoir un tableau ou un objet implémentant **Traversable**. PHP 7.1 ajoute le pseudo-type **iterable** qui regroupe les deux et permet donc de résoudre le problème.

```
// N'écrivez plus :
// function test(array $arr) {
// mais :
function test(iterable $arr) {
  foreach ($arr as $value) {
    var_dump($value);
  }$
}
```

8 Création de fermetures à partir de Callable

La classe **Closure**, ou fermeture en français, est utilisée en PHP pour manipuler des fonctions anonymes sous la forme d'objets. L'implémentation du principe des fermetures diffère de celles faites dans d'autres langages, à tel point que fermetures et fonctions anonymes sont pratiquement synonymes en PHP, alors que ce n'est habituellement pas le cas.

PHP 7.1 offre la possibilité de créer des fermetures à partir de n'importe quel élément appelable, c'est-à-dire appartenant au pseudo-type **Callable** regroupant tous les éléments qui peuvent être appelés comme des fonctions. Il peut s'agir d'une chaîne de caractères contenant le nom d'une fonction, ou même encore un tableau à deux éléments, un objet et le nom d'une de ses méthodes. Pour ce faire, il faut recourir à la méthode statique **Closure::fromCallable**, qui n'attend qu'un seul argument, l'élément appelable.

```
$a = 'strlen';
$b = Closure::fromCallable($a);
```

La méthode est sensible au contexte et la visibilité ; ainsi, une méthode protégée ne pourra pas donner lieu à une fermeture si celle-ci est écrite hors de la classe :

```
class MaClasse {
  protected function hello(){
    echo 'Hello world !';
  }
  public exportHello() {
    return Closure::fromCallable([$this,'hello']);
  }
}

$a = new MaClasse();
$b = [$a, 'hello'];
$c = Clossure::fromCallable($b); // cette ligne provoquera une erreur,
car 'hello()' est protégée

$d = $a->exportHello(); // cette ligne fonctionnera, car la fermeture
est créée à partir d'une méthode visible dans son contexte de création
```

L'intérêt d'une telle innovation est triple. Tout d'abord, il offre une nouvelle possibilité d'un point de vue algorithmique : on peut ainsi exporter une méthode protégée ou privée dans une fermeture sans avoir à la rendre publique, comme nous venons de le voir dans l'exemple précédent.

En second lieu, cela permet de mieux détecter des erreurs d'affectation. Dans l'exemple suivant :

```
$a = 'dtrlen'; // faute de frappe, on aurait voulu taper `strlen`
// beaucoup de code suit, jusqu'à l'appel fatidique
$a(); // l'erreur n'est signalée qu'ici par PHP, car l'appel échoue.
```

Avec **Clossure::fromCallable()**, l'erreur est détectée dès l'affectation, car celle-ci échoue. Reprenons ce dernier exemple, mais avec cette méthode :

```
$a = Closure::fromCallable('strlen'); // l'erreur a été détectée
à l'endroit où la faute de frappe a eu lieu
// de nouveau beaucoup de code
$a(); // là, on aura une autre erreur, car 'strlen' attend au
moins un argument obligatoire
```

Le dernier intérêt concerne les performances. En effet, si vous utilisez beaucoup de fonctions attendant des appelables, vous serez tenté de tirer parti des fonctionnalités de PHP 7 en l'indiquant explicitement dans la signature :

```
function maFunction(Callable $fn) {
```

Il se trouve que cette façon de faire demande plus de travail à PHP, car **Callable** est un pseudo-type qui regroupe plusieurs types. Pour valider cette exigence de type d'argument, PHP est amené à faire des tests assez nombreux. Une solution est alors de n'accepter que le type **Closure**. Une unique conversion manuelle sera nécessaire pour commencer, petit prix pour de meilleures performances.

Conclusion

Nous n'avons détaillé ici que les améliorations les plus significatives au regard du travail quotidien d'un développeur. Listons-en tout de même rapidement quelques autres, moins importantes : support de la méthode HTTP PUSH par la librairie CURL ainsi que d'autres améliorations de cette librairie ; réparation du comportement inconsistant de la variable **$this** dans certains cas d'affectation ; meilleure précision sur certains traitements des nombres à virgule flottante, notamment lors de leurs exports (avec **json_encode()** ou **var_export()**).

PHP 7.1 n'est donc pas une révolution, mais nous constatons tout de même des progrès significatifs qui renforcent grandement la consistance du langage et améliorent le confort du développeur. ■

FICHTRE ET SI J'ÉTAIS LE PROBLÈME DE MON ÉQUIPE ?

The Cheshire CAT - [Everyone here is mad]

Depuis quelques numéros maintenant je parle dans ces colonnes de bonnes pratiques, de travail en équipe, de choses à faire et à ne pas faire. Pour conclure ce cycle, je vais terminer par un article essayant de donner des pistes si vous vous êtes reconnus dans le portrait du Mister Hyde que j'ai dépeint au cours de mes articles précédents.

Mots-clés : Humeur, Développeur, Empathie, Équipe, Junior, Senior

Résumé

On écrit que rarement du code seul. La plupart du temps, et c'est une bonne chose, on travaille en équipe. Mais travailler en équipe demande plus que d'être juste bon pour écrire du code. Travailler en équipe demande de savoir communiquer, de faire preuve d'empathie, de compassion et d'humilité. Au cours de mes précédents billets d'humeur, j'ai souvent esquissé le portrait d'un Mister Hyde du développement, celui ou celle par qui l'implosion d'une équipe arrive. Si vous vous êtes reconnu dans ce portrait en creux, tout n'est pas perdu, continuez à lire et vous retrouverez peut-être le chemin pour redevenir le bon docteur Jekyll.

Pour commencer, je vais répondre à une objection que l'on m'oppose souvent lorsque je pars dans une grande discussion sur les choses à faire ou à ne pas faire lorsque l'on travaille en équipe. On me rétorque souvent que si je tiens ce discours c'est parce que je tiens à mon code et que je ne veux pas qu'on me dise qu'il est mal foutu et que donc je me lance dans de grands palabres sur les relations au sein d'une équipe, sur le fait d'être gentil quand on parle de code, etc. À cela je rétorque toujours la même chose. Que premièrement, je ne m'attache pas au code, je m'attache aux gens. Il est tout à fait normal que l'on critique le code que j'ai écrit et que je critique le code que mes camarades de travail ont eux-mêmes écrit. Mais il n'est pas normal que quiconque puisse se sentir blessé par une critique formulée sur son travail. Et deuxièmement, je pense que la façon dont le code est écrit est au moins aussi importante que le code lui-même. Du code écrit par une équipe, qui fonctionne de manière efficiente, sera à mon sens plus efficace, plus propre et de bien meilleure qualité qu'un code écrit par une équipe dysfonctionnelle, entravée par des problèmes d'égo et de rancœur cachée.

1 Êtes-vous vraiment un Mister Hyde qui s'ignore ?

Vous avez lu mes précédents billets et à de nombreuses reprises vous vous êtes dit « mais oui, c'est exactement comme cela que je ferais, c'est tout moi ça ! » et juste après vous avez été très surpris de lire que je n'avais pas un avis positif sur une telle façon de faire ? Et cela vous est arrivé régulièrement, plusieurs fois par billet d'humeur ? Du coup, un doute s'est mis à grandir en vous. Est-ce que votre façon d'interagir avec les autres ne serait pas une façon peu efficiente de faire les

choses ? Est-ce qu'il n'y aurait pas une façon différente de faire qui vous permettrait d'être plus apaisé dans votre travail. Pour apaiser vos craintes ou finir de vous ouvrir les yeux, je vais rapidement lister les différents comportements qui, pour moi, sont ceux d'un Mister Hyde, triste personnage qui finira inévitablement par conduire à l'implosion de son équipe.

1.1 Le positionnement « d'ancien »

Ce n'est pas un point obligatoire, mais c'est bien souvent le point commun des Mister Hyde. Ce sont des gens qui se définissent comme senior. Ils sont soit les plus anciennes personnes sur le projet, soit ils pratiquent la techno en question depuis « au moins 15 ans tu comprends mon petit ». Alors forcément c'est normal qu'ils connaissent bien les choses et qu'ils prennent les décisions. Parce qu'ils ont le savoir et la compétence.

1.2 Choix impactant l'équipe

Ici je veux aussi bien parler de *coding style* que du choix des outils de développement ou de gestion de versions des sources. Dans tous les cas, vous avez décidé, parce que vous savez. Dans le cas où vous développez dans un langage fournissant son propre *coding style* (comme en **Java** ou même en **C++**), vous avez modifié celui-ci pour définir le vôtre, parce qu'il est mieux et que c'est comme cela. Dans tous les cas, quand un nouveau ou une nouvelle arrive dans l'équipe, il faudra alors absolument lui asséner au plus vite les tables de la loi de l'équipe. Tables de la loi qui par définition ne souffriront pas d'être remises en question.

1.3 Review de code

Ici le Mister Hyde ne sait pas du tout ce que veut dire d'être intransigeant avec le code, mais gentil avec le développeur. Son crédo c'est plus « qui aime bien châtie bien » ou « ce qui ne tue pas rend plus fort ». Donc autant dire que ses phrases favorites sont « mais ce code c'est de la merde » ou « je vais devoir tout réécrire, c'est vraiment pas bon, mais c'est normal, je suis meilleur ». En fait, bien souvent, une *code review* faite par un Mister Hyde n'est pas là pour vérifier que le code est correct, fonctionne et ne va pas envoyer de bug en production. Non, elle est là pour vérifier que les autres membres de l'équipe ont écrit du code aussi proche que possible de ce que lui aurait écrit. Et pour leur expliquer dans les grandes largeurs ce qu'il faut faire et ce qu'il ne faut pas faire. Bien entendu personne n'a le droit de relire votre code, à part pour apprendre, bien entendu ; mais le relire pour le vérifier ne sert à rien, cela serait du temps perdu, vu que vous *reviewez* vous-même votre propre code et que personne n'a le niveau pour le faire mieux que vous.

1.4 Solitude et turnover

On ne s'en rend pas forcément compte tout de suite... mais il est étrange de constater alors que les petits nouveaux vous posaient un certain nombre de questions au départ, assez rapidement ce flot de questions triviales se soit tari. Cela ne vous dérange pas, vous aviez l'impression d'un bruit de fond qui vous déconcentrait et puis vous avez autre chose à faire. Vous avez tout le code des autres à relire d'abord et à réécrire ensuite, pour qu'il soit vraiment parfait. Mais cela vous embête tout de même un peu que les membres de votre équipe ne restent pas plus qu'un ou deux ans, voire trois au grand maximum avant de s'en aller.

2 En route vers la guérison

C'est maintenant sûr et certain : vous êtes donc bien un mister Hyde et cela ne vous satisfait plus. Vous avez fait un premier pas, vous acceptez le fait que vos pratiques peuvent changer et surtout s'améliorer pour la plus grande joie des autres membres de votre équipe et donc de vous-même.

Mais comment donc changer. Je vous propose quelques idées qui pourraient bien fonctionner. Mais comme ce sera difficile de lutter contre votre nature, je vous propose de mettre en place un garde-fou pour vous aiguillonner. Rien de bien méchant, je suis contre le *shaming* des gens. Non, si lors de votre cure dans le chemin pour redevenir un docteur Jekill, vous vous rendez compte que vous retombez dans vos vieux travers, je vous propose de vous obliger à boire une grande cuillère à soupe d'huile de foie de morue. C'est très bon pour la santé et cela devrait vous empêcher de récidiver trop souvent [les puristes me diront qu'il faut utiliser un grand verre d'huile d'arachide, mais cela n'aurait aucune vertu pour votre santé, donc je préfère celle de foie de morue].

2.1 Concertation et remise en cause des pratiques

Il n'y a rien de plus sclérosant que les règles définies à un moment donné, sans aucune justification et que l'on applique simplement parce qu'elles sont là. Ou simplement parce que le senior a décidé qu'elles devaient s'appliquer. C'est l'exemple parfait du théorème du singe et de la banane. Forcez-vous donc à remettre en question les pratiques mises en place dans votre équipe. Même si c'est vous qui les avez mises en place. Demandez aux autres ce qu'ils pensent de la façon de fonctionner actuelle et proposez de mettre en place les modifications qu'une majorité de personnes pense intéressantes. Et cela même si vous êtes sûr que rien ne sera mieux que ce que vous, vous aviez défini il y a 8 ans.

2.2 Bannir l'argument d'autorité

Infantiliser votre équipe n'est pas une bonne chose. Reprocher à vos camarades de ne pas être vos clones n'est pas non plus une bonne chose. Interdisez-vous donc en toute circonstance de mettre un véto sur quelques discussions qui soient si votre seul argument est « je ne ferais pas comme cela ». Si c'est votre seul argument, allez plutôt siroter une cuillère d'huile. Sinon, discutez-en avec de vrais arguments.

2.3 Bannir la rugosité dans le langage

Non, le code ce n'est pas de la merde et cela n'est pas normal d'émettre ainsi un tel jugement de valeur. Non, dire à quelqu'un « je vais le réécrire cette nuit, c'est vraiment trop mal fait, j'aurais mieux fait de le faire tout de suite » n'est pas non plus une phrase qui donne envie de progresser ou même simplement de faire bien la prochaine fois. Adoptez donc un langage plus positif, sans vulgarité ou terme employé uniquement dans le seul but de blesser.

2.4 Review de code intelligente

Oubliez vos vieilles revues de code, faites la nuit dans votre bureau, en autarcie avec pour unique but de démolir le code que vous relisez parce que c'est forcément de la merde. Définissez en équipe une *checklist* de ce que doit vérifier une *code review* et tenez-vous-y. Et faites votre *review* de code en étant assis à côté de la personne qui a écrit le code que vous *reviewez* pour pouvoir en parler et échanger avec celle-ci. Et pliez-vous à l'exercice : faites relire votre code par les autres membres de votre équipe, et acceptez les critiques constructives sans vous sentir automatiquement attaqué. Ici il est primordial de ne pas oublier votre petite bouteille d'huile de foie de morue, croyez-moi, dans les premiers temps, vous allez en avoir besoin.

2.5 Pair programming

Une fois que vous aurez réussi à faire une revue de code gentille et respectant la *checklist* définie, vous pouvez passer à un exercice beaucoup plus difficile pour votre Mister Hyde intérieur : le *pair programming* sans que vous ayez le clavier. Jusqu'à présent les sessions de *pair programming* que vous faisiez ressemblaient à un cours magistral ou à une vidéo de développeur sur **Twitch**. Vous tapiez le code, vous expliquiez et les autres prenaient des notes. Là, vous n'allez pas avoir le droit au clavier. Vous allez devoir regarder, discuter, argumenter, définir les tests qui valideront le code et c'est tout. Et lorsque la session de code sera finie, si les tests passent, et bien alors on *push*.

Conclusion

On a tous, malheureusement, à un moment ou un autre, un peu de Mister Hyde dans la pratique de notre métier. C'est triste, mais c'est ainsi. Et alors qu'on pourrait croire que l'expérience et l'âge nous protègeraient de tels travers, c'est bien souvent l'inverse qui se produit. Il faut simplement faire attention, écouter les autres, et garder autant que possible un état d'esprit bienveillant et légèrement innocent dans sa façon de vivre avec les autres. Et ainsi nous resterons tous de bons Docteur Jekill. Je finirai ce billet par, une fois n'est pas coutume, un conseil de lecture. Pour ceux qui ne l'auraient pas déjà fait, je vous conseille de lire « Cristal qui songe » et « Les plus qu'humains » de Theodore Sturgeon. Les vacances d'été sont finies depuis quelque temps déjà, c'est donc trop tard pour les lire sur la plage, mais vous trouverez bien une ou deux soirées d'hiver pour cela... ■

GNU

COMPRENDRE, UTILISER ET ADMINISTRER LINUX

LINUX PRATIQUE

ESSENTIEL

À partir du 23 décembre,

Linux Pratique change...

...découvrez sa

Nouvelle formule !

+ de Raspberry Pi

+ de tutoriels

+ d'initiation à la programmation

+ de logithèque

Les nouvelles rubriques de votre magazine :

- Cahier Raspberry Pi & débutant Linux
- Programmation & scripts
- Logithèque & applicatif
- Mobilité & objets connectés
- Système & personnalisation
- Web & réseau
- Terminal & ligne de commandes
- Entreprise & organisation
- Réflexion & société ...

COMPRENEZ, UTILISEZ & ADMINISTREZ LINUX SUR PC, MAC & RASPBERRY PI AVEC LINUX PRATIQUE !

UTILISEZ UN CODE SOURCE DE QUALITÉ EN RESPECTANT UN STANDARD

Stéphane LONKENG TOULEPI - [Ingénieur en Télécommunication et Entrepreneur en Électronique médicale]

La plupart des tutoriels et livres de programmation disposent d'un chapitre lié aux conseils sur la manière de nommer les variables, les fonctions, les commentaires, etc. Mais pourquoi diantre se tuer à essayer de respecter ces fichues règles alors que je peux aller droit au but ? Cette question, je suis sûr que tous nous nous la somme posée à nos débuts en programmation. La plupart d'entre nous a, avec le temps, fini par se faire un ensemble de règles personnelles pour quand même s'y retrouver dans le code lorsqu'il y a problème. Certains sont obligés de respecter des règles de codage [1] parce qu'il faut bien que leurs collègues du projet comprennent facilement. Nous allons voir dans la suite comment améliorer la qualité de son code source grâce aux standards de codage existants.

Mots-clés : Standards, GNU C, Règles de codage, SonarQube, Analyse, Plugins

Résumé

Les standards de programmation étant nombreux, dans cet article nous allons vous présenter de façon rapide le standard GNU C utilisé par les programmeurs de la FSF (*Free Software Fundation*) et autres programmes *open source* écrits en C. La vérification du code relativement au standard requérant généralement un analyseur, nous allons prendre le cas du logiciel *open source* SonarQube et présenter un exemple d'analyse de code relativement aux règles prédéfinies pour un projet.

Tous les fabricants de logiciels informatiques vous diront que leur logiciel est de bonne qualité **[2]**. C'est à se demander s'il s'agit de la qualité de la conception du logiciel, de la qualité perçue par le client (ergonomie, etc.), de la facilité de maintenance, de la performance, la portabilité ou encore de la fiabilité. Une chose est sûre c'est qu'il n'y a pas de logiciel de bonne qualité sans que son code source ne le soit. Nous nous intéressons ici à la qualité du code source et aux instruments qui permettent de la mesurer. Il existe des standards de codage qui ont été mis sur pied pour normaliser la programmation en certains langages et permettent aujourd'hui de faciliter la mesure de la qualité du code.

1 C'est quoi « un code de qualité » ?

Dire objectivement qu'un code source est de bonne qualité **[3][4]** est très difficile. L'objectivité en terme de qualité de code source est recherchée depuis des années. En effet, on a assisté à plusieurs tentatives de normalisation dans ce sens parmi lesquelles les standards **ISO9126 [5]** ou **ISO/IEC 15939 [6]** qui n'ont apporté que des pistes.

La qualité d'un code source dépend fermement des objectifs du projet. Ainsi, dans l'idéal, avant de démarrer un projet, il est impératif de choisir les critères de qualité à prendre en compte. Ceci permet de tenir compte de la vérification, des outils à utiliser et même du management à mettre sur pied pour mieux suivre la qualité du code.

Note

La norme ISO/CEI 9126 définit un langage commun pour la définition de la qualité d'un logiciel. Il introduit des termes tels que **fiabilité**, **maintenabilité**, **portabilité**, etc. La norme ISO 15939 quant à elle est encore moins connue et propose une méthodologie pour la mesure des caractéristiques d'un logiciel.

Comme critères de qualité de code, on peut citer **[5]** :

- La **maintenabilité [7][8][9]** : il s'agit de la capacité de l'application issue du code à être maintenue, de manière cohérente et à moindre coût, en état de fonctionnement. Cette notion s'étend généralement au niveau de la conception ; en effet, pour un logiciel très mal conçu, la bonne qualité du code ne garantit pas sa maintenabilité. Les deux qualificatifs du code qui affectent sa maintenabilité sont :
  - → Un **code commenté** : en effet, les commentaires de code facilitent la maintenance. En règle générale, il faut atteindre le juste milieu en terme de densité de commentaire. Trop de commentaires rendent le code utile illisible et noyé dans les informations inutiles tandis qu'avec trop peu de commentaires, il risque d'y avoir beaucoup d'incompréhensions de la part du lecteur dû au manque d'informations sur les rôles du code utile.
  - → Un **code compréhensible** : ici, il peut être subdivisé en trois aspects ; une lisibilité du code source, son découpage clair et une mutualisation des fonctions. Il faut par ailleurs remarquer qu'il est très rare qu'un code soit maintenu par son auteur original.
- La **portabilité** : c'est la capacité du code à fonctionner dans un environnement matériel différent de celui pour lequel il a été initialement fait.
- L'**organisation** : il s'agit de l'organisation des répertoires et fichiers sources. Ce critère prend également en compte le fait qu'il y ait une utilisation de modèles comme MVC, les contrôles, les règles de gestion et autres structures permettant d'améliorer l'architecture du code.
- Convention de nommage : il s'agit d'une convention qui précise la façon de nommer les fonctions, les variables et les constantes. L'avantage d'adopter une convention c'est la mise en avant des informations sur l'objet utilisé ; en effet, lorsqu'une convention de nommage est utilisée, il est facile pour un programmeur de savoir s'il s'agit d'une fonction, d'une variable ou d'une constante. Il existe des conventions de nommage standards comme la notation hongroise **[10]** très utilisée dans le cas du langage C.
- Facilité de test unitaire : enfin, l'automatisation du test unitaire du code est un critère important. Il s'agit en fait d'évaluer si la vérification du bon fonctionnement d'une partie précise est facilement faisable.

Note

Un test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme.

Après avoir choisi ces critères, il faut sélectionner une métrique pour chacun d'entre eux. Enfin, on définira les seuils limites en dessous desquels il ne faudra pas descendre pour se dire qu'on a un code de qualité. Les métriques ici sont des valeurs numériques qui permettent de mesurer les attributs de la qualité du code.

Attention !

Mathématiquement, une métrique est une fonction qui permet de mesurer la distance entre deux objets. Ainsi, parler de métrique de qualité de code est un abus ; on devrait plutôt parler de mesures de qualité de code. Les mesures peuvent être directement observables comme le nombre de lignes, la durée de programmation, etc.

Des métriques peuvent être associées aux critères précédents. Il existe plusieurs métriques que nous n'avons pas le temps d'aborder dans cet article. On peut parler à titre

d'exemple de l'indice de maintenabilité (**MI3** ou **MI4**) qui mesure la complexité de maintenance **[7][8]**.

Complexité de maintenance

La complexité de maintenance intègre le nombre de lignes de code, la complexité cyclomatique du code et la complexité d'Halstead **[11]** du code. Pour information, la complexité cyclomatique représente le nombre de chemins possibles au travers d'un programme présenté sous la forme d'un graphe tandis que celle d'Halstead est basée sur le nombre de marqueurs, classifiés comme opérateur ou opérande comme le `+`, le `-`, le OR, le ET, etc. La formule de l'indice de maintenance **MI** est :

```
MI = (171-5,2xln(V)-0,23x(CC)-16,2xln(loc))
x100/171
```

où `V` = *Halstead Volume*, `CC` = *Cyclomatic Complexity*, et `loc` = *count of source Lines Of Code* (SLOC).

2 Standard de codage GNU C

Le standard GNU **[12]** a été écrit par Richard Stallman et un ensemble de développeurs volontaires. Leur objectif était de faire en sorte que les codes des programmes du système GNU soient portables, robustes, fiables et surtout compréhensibles par d'autres développeurs vu que ces programmes sont la plupart du temps appelés à évoluer et à être modifiés. Ce standard de codage est focalisé sur le langage C, cependant, les règles et les principes qu'il intègre peuvent être très utiles pour d'autres langages de programmation. Ce standard donne également le minimum pour mettre sur pieds un *package* GNU.

Ce standard est régulièrement mis à jour par les développeurs de la FSF et les contributions des utilisateurs de programmes sous licences GNU. Il faut noter que le standard GNU peut être utilisé de façon concurrente dans un même programme avec d'autres standards pourvu que les règles ne soient pas contradictoires. En effet, utiliser deux standards sur un même code peut être intéressant, mais peut également entraîner des problèmes liés aux contradictions de règles.

Dans ce qui suit, nous présentons les règles pratiques du standard GNU utiles pour le langage C lorsqu'on écrit un programme GNU.

2.1 Formatage du code source

Le principal conseil ici repose sur les accolades qui marquent le début d'une fonction. Elles doivent être placées à la première colonne de la ligne. Par ailleurs, aucune autre accolade de la même fonction ne devrait être placée à la première colonne. Le standard conseille de ne pas écrire de lignes de code qui excèdent 79 colonnes (ou 79 caractères).

2.2 Comment commenter le code

Le standard met en exergue les endroits où doivent être placés des commentaires ; par exemple, chaque programme doit commencer par un commentaire expliquant brièvement son utilité en 15 mots maximum. Ce commentaire doit être en haut du fichier source contenant la fonction `main`. Le standard précise également qu'il faut insérer un commentaire au début de chaque fonction pour décrire son utilité. La langue à utiliser pour les commentaires est l'anglais qui est de loin la plus utilisée par les développeurs à ce jour.

2.3 Conventions syntaxiques

Dans cette rubrique, le standard décrit les spécificités liées aux déclarations d'objets (type, variable, entier, constante, etc.). Par exemple, il suggère de ne pas effectuer la déclaration de plusieurs variables sur une même ligne de code ; il vaut mieux aller à la ligne pour effectuer la déclaration d'une nouvelle variable.

2.4 La nomenclature des objets

La nomenclature des objets présente comment nommer les variables, les fonctions et les constantes. Ici, les noms de ces éléments sont considérés comme étant des commentaires vu qu'ils sont censés donner des informations sur leur utilité. À titre d'exemple, le standard recommande d'utiliser le caractère de soulignement `_` pour un nom intégrant plusieurs mots et les majuscules pour les constantes et les macros. Par exemple, il faut écrire `ignore_space_change_flag` au lieu de `IgnoreSpaceChangeFlag`.

2.5 Portabilité du programme

Ici, il est question de faire en sorte que les programmes respectant le standard GNU soient compatibles avec tous les systèmes GNU basés sur Linux. Le standard incite les développeurs à rendre également compatibles leurs programmes sur d'autres systèmes d'exploitation libres et d'autres systèmes de type Unix. Mais cette dernière incitation n'est pas

une obligation si elle s'avère difficile. En outre, la meilleure façon d'assurer la portabilité de la plupart des systèmes Unix est l'utilisation d'**Autoconf**.

> **Note**
>
> L'outil Autoconf est un programme qui permet de produire des scripts shell qui configurent automatiquement le code source d'un logiciel pour l'adapter à l'environnement dans lequel il va être exécuté. Ceci facilite donc la gestion de la portabilité.

2.6 Portabilité des processeurs

Les systèmes GNU diffèrent à cause des types de CPU sur lesquels ils seront exécutés. Le cas le plus commun est l'**endianness** ; certains processeurs sont gros-boutistes (*big-endian*), i.e. l'octet de poids fort est en tête à l'enregistrement tandis que d'autres sont petit-boutistes (*little-endian*) qui représente le cas contraire du précédent. Il faut donc en tenir compte lors de la programmation de façon à ce que le programme puisse tenir sur les deux types de CPU. Par ailleurs, le standard précise que GNU ne supporte pas les machines 16 bits ; il n'est donc pas nécessaire d'essayer de rendre les programmes compatibles avec ce type de processeurs.

2.7 Portabilité des librairies standards

Il est recommandé d'utiliser des interfaces standards si possible ; cependant, dans le cas où l'utilisation des **extensions GNU** rend le programme plus puissant, portable et maintenable, il faudra plutôt les utiliser.

2.8 Internationalisation du code

Le standard incite à utiliser la librairie **GNU gettext** qui facilite la traduction des messages du programme en plusieurs langues. Ainsi, il faut utiliser comme langue l'anglais et laisser l'utilitaire gettext se charger de la traduction en d'autres langues. Je vous renvoie à *GNU/Linux Magazine Hors-série n° 70* **[13]** qui traite de l'utilitaire en question.

2.9 Caractères utilisables

Il s'agit à ce niveau du codage des caractères. Le standard demande d'utiliser les caractères ASCII sous 7-bits pour les commentaires, le code et autres documents relatifs au programme.

2.10 Utilisation de Mmap

Le standard précise que l'utilisation de **Mmap** nécessite au préalable un test hors programme afin de s'assurer qu'il fonctionne bien sur le fichier à traiter. En effet, Mmap ne fonctionne pas forcément sur tous les fichiers. Dans le cas où Mmap ne fonctionne pas normalement, il vaut mieux utiliser d'autres outils comme **read** et **write**.

Comme on peut le constater, l'application de ces différentes règles peut s'avérer fastidieuse si l'on s'y met de façon manuelle. C'est la raison pour laquelle il vaut mieux se servir d'un outil de vérification automatique notamment **SonarQube**.

3 Application avec l'outil SonarQube

3.1 Présentation

L'outil SonarQube **[14][15]** est un logiciel libre permettant de mesurer la qualité d'un code source. Il supporte plus de 25 langages de programmation (C/**C++**, **Java**, **Objective-C**, **PHP**, etc.). Après analyse, il effectue un rapport sur :

- les duplications de code ;
- le niveau de documentation ;
- le respect des règles de programmation ;
- la détection des bugs potentiels ;
- l'évaluation de la couverture du code par tests unitaires ;
- la complexité du code ;
- le design et l'architecture de l'application.

3.2 Installation sur une distribution Linux (Debian Jessie)

En termes de matériel, un PC avec 2 GB de RAM devrait largement suffire en fonction de l'utilisation. En effet, étant donné qu'il s'agit d'un logiciel serveur, plus il y a d'utilisateurs, plus les caractéristiques matérielles du serveur doivent être élevées.

Il est important de noter entre autres qu'il existe une version de **SonarQube** en ligne et gratuite ; elle s'appelle **Nemo**. Avec cette version, on peut librement *uploader* son projet et le faire analyser. Dans ce qui suit, nous allons voir comment installer SonarQube sur un PC qui jouera le rôle de serveur en local.

Comme prérequis, il faut installer **OpenJDK7** et un serveur de base de données ; nous allons utiliser **MySQL** dans le cas d'espèce. SonarQube fonctionne également avec **Oracle**, **PostgreSQL** et **Microsoft SQLServer**.

```
# apt-get install openjdk-7-jre
# apt-get install mysql-server
```

Il faut veiller à ne rien mettre comme mot de passe pour le super-utilisateur *root* de MySQL. Ensuite, on crée une base de données et un utilisateur **sonarqube** auquel on attribue les droits pour créer, supprimer et modifier la base de données :

```
# mysql
mysql > CREATE DATABASE SONARDATABASE;
mysql > CREATE USER 'sonarqube'@'localhost' IDENTIFIED BY '';
mysql > GRANT ALL PRIVILEGES ON * TO 'sonarqube'@'localhost';
mysql> ALTER DATABASE SONARDATABASE CHARACTER SET utf8;
```

Pour installer SonarQube, on commencer par télécharger et décompresser la distribution, le scanner et les projets exemples dans le répertoire **/etc** :

```
$ wget https://sonarsource.bintray.com/Distribution/sonarqube/
sonarqube-5.6.zip
$ unzip sonarqube-5.6.zip -d /etc/
$ wget https://sonarsource.bintray.com/Distribution/sonar-scanner-cli/
sonar-scanner-2.5.1.zip
# unzip sonar-scanner-2.5.1.zip -d /etc/
$ wget https://github.com/SonarSource/sonar-examples/archive/master.zip
# unzip master.zip -d /etc/
```

Note

SonarQube 5.6 est une version LTS. SonarQube 6.0 est également disponible, donc à vous de choisir quel type d'installation vous préférez.

Il faut ensuite installer le serveur web. Pour ce faire, on commence par configurer l'accès à la base de données par le fichier **/etc/sonarqube-5.6/conf/sonar.properties**. Il faut noter que des *templates* sont disponibles dans ce fichier pour chaque type de base de données. Il faut juste retirer les commentaires sur les lignes relatives à MySQL que nous utilisons dans ce cas. Voici les lignes que j'ai eu à commenter :

```
sonar.jdbc.username=sonarqube
sonar.jdbc.password=
sonar.jdbc.url=jdbc:mysql://localhost:3306/SONARDATABASE?
useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=
true&useConfigs=maxPerformance
```

Il s'agit des informations de connexion à la base de données. La dernière ligne est spécifique au système de base de données utilisé et dans notre cas il s'agit de MySQL. Il faut rajouter le driver JDBC dans le cas d'Oracle. Pour les autres systèmes de gestion de base de données, il est intégré automatiquement à l'installation.

Les informations à modifier concernant le serveur web sont consignées dans le fichier précédent. Les lignes à modifier concernent le port, l'adresse IP et le contexte du serveur web. Voici les lignes de code du fichier dans mon cas :

```
sonar.web.host=127.0.0.1
sonar.web.port=9000
sonar.web.context=/sonar
```

Il faut veiller à ce que le chemin du fichier binaire (exécutable) Java corresponde bien à ce qui est spécifié dans le fichier **/etc/sonarqube-5.6/wrapper.conf** au niveau de la ligne spécifiant la variable **wrapper.java.command**. Ensuite, on peut lancer SonarQube. Pour des besoins de *debugging*, on peut ouvrir un autre terminal pour observer avec la commande **tailf** les logs dans le fichier **/etc/sonarqube-5.6/logs/sonar.log** afin de détecter d'éventuelles erreurs.

Terminal A :

```
# /etc/sonarqube-5.6/bin/linux-x86-64/sonar.sh start
```

Terminal B :

```
# tailf /etc/sonarqube-5.6/logs/sonar.log
```

Pour confirmer que l'installation a été bien faite, vous devez avoir la page de la figure 1 à l'adresse localhost:9000.

À ce niveau, on peut se connecter avec les paramètres d'administrateur par défaut qui sont **admin**/**admin**. Après s'être connecté en tant qu'administrateur, il faut aller dans l'onglet ***Administration > System > Update Center***. La page qui apparaît montre les différents *plugins* installés.

3.3 Installation d'un plugin

Avant d'analyser le code programmé dans un langage spécifique, il faut au préalable installer le *plugin* qui correspond au langage de programmation en question. Dans notre cas, nous allons installer le *plugin* libre **C++ (Community)**. Avant de le télécharger, il faut s'assurer que la version du *plugin* est compatible avec la version de SonarQube installée ; la matrice du lien https://github.com/SonarOpenCommunity/sonar-cxx/wiki/SonarQube-compatibility-matrix résout ce problème.

Dans notre cas, nous allons télécharger la version V0.9.6. Après l'avoir téléchargée, on met le fichier **.jar** dans le répertoire **/etc/sonarqube-5.6/extensions/plugins/**.

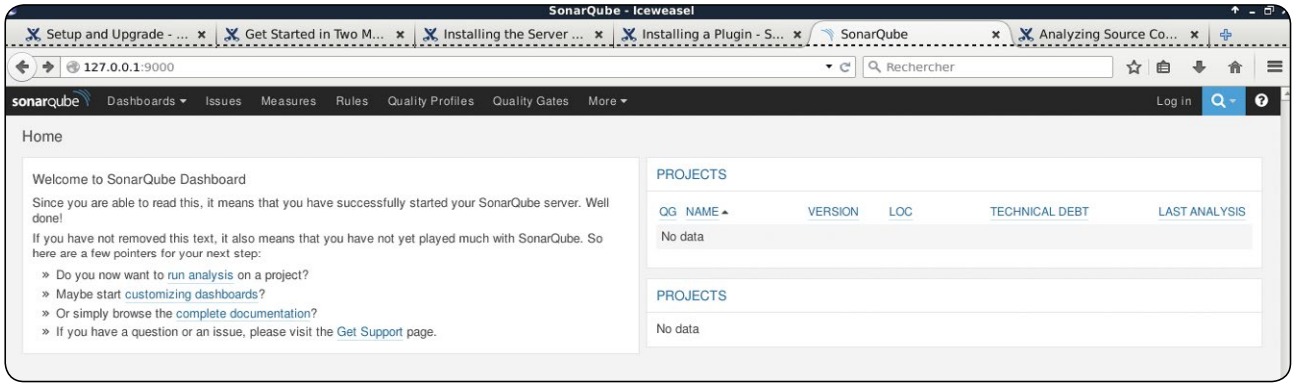


Fig. 1 : Écran d'accueil de SonarQube après installation et lancement.

Ensuite, on redémarre SonarQube.

```
/etc/sonarqube-5.6/extensions# wget
https://github.com/SonarOpenCommunity/
sonar-cxx/releases/download/cxx-0.9.6/
sonar-cxx-plugin-0.9.6.jar
/etc/sonarqube-5.4/extensions# /etc/
sonarqube-5.6/bin/linux-x86-64/sonar.sh
restart
```

De retour à la page ***Update Center***, on se rend compte que le nouveau *plugin* C++ (Community) a bien été installé.

3.4 Analyse du code

Nous allons analyser un exemple de code. Il s'agit d'un code Java se trouvant dans le répertoire **`/etc/sontar-examples-master/projects/languages/java/sonar-runner/java-sonar-runner-simple`** :

```
/etc/sonar-examples-master# cd projects/
languages/java/sonar-runner/java-sonar-
runner-simple
```

Avant de lancer le scan, il faut éditer le fichier **`sonar-project.properties`**. Il s'agit du fichier de configuration du scan. Il contient plusieurs champs qui permettent de spécifier le langage utilisé dans le projet, le *plugin*, le répertoire des sources à compiler, la version, etc. La liste des variables avec leurs significations est disponible sur le lien : https://github.com/SonarOpenCommunity/sonar-cxx/wiki/Supported-configuration-properties.

Voici les variables du fichier que j'ai avant exécution du scanner :

```
# Required metadata
sonar.projectKey=org.sonarqube:java-simple-sq-scanner
sonar.projectName=Java :: Simple Project Not Compiled :: SonarQube Scanner
sonar.projectVersion=1.0
# Comma-separated paths to directories with sources (required)
sonar.sources=src
# Language
sonar.language=java
# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

Le scan se fait en ligne de commandes dans le répertoire du projet où se trouve le fichier de configuration précédent. Il suffit dès lors de lancer la commande suivante dans ce répertoire :

```
[18:48 root@lonkeng java-sonar-runner-simple] > /etc/sonar-scanner-2.5.1/bin/sonar-runner
```

Une fois le scan terminé, on peut observer les résultats sur la page d'accueil du serveur SonarQube. Si on clique sur le projet ***Java :: Simple Project Not Compiled :: SonarQube Scanner***, on constate qu'il y a six problèmes dans le code. Le nombre de lignes de code est de 31. Après un clic sur le nombre six, on voit affiché à l'écran l'ensemble des violations de règles dans le code. Par exemple, dans ce projet, SonarQube détermine la complexité cyclomatique (définie plus haut) de la fonction **`intToEnglishValue`** à **`15`**. Cette valeur est beaucoup plus élevée que la valeur autorisée qui est de **`10`**.

Par ailleurs, on peut observer les règles par plugins à prendre en compte par SonarQube dans la rubrique ***Rules*** de l'interface graphique de SonarQube. On peut sélectionner par exemple le champ ***Java*** et le champ ***brain-overload*** dans la barre latérale de gauche tel que présenté sur la figure 2.

On constate qu'il existe donc une règle du *plugin* Java intitulée « *Methods should not be too complex* ». Il s'agit en fait de la règle qui a été appliquée dans le code précédent. En double-cliquant sur la règle, on peut lire que la complexité maximale pour une méthode est de 10. Il faut donc corriger le code et effectuer d'autres analyses.

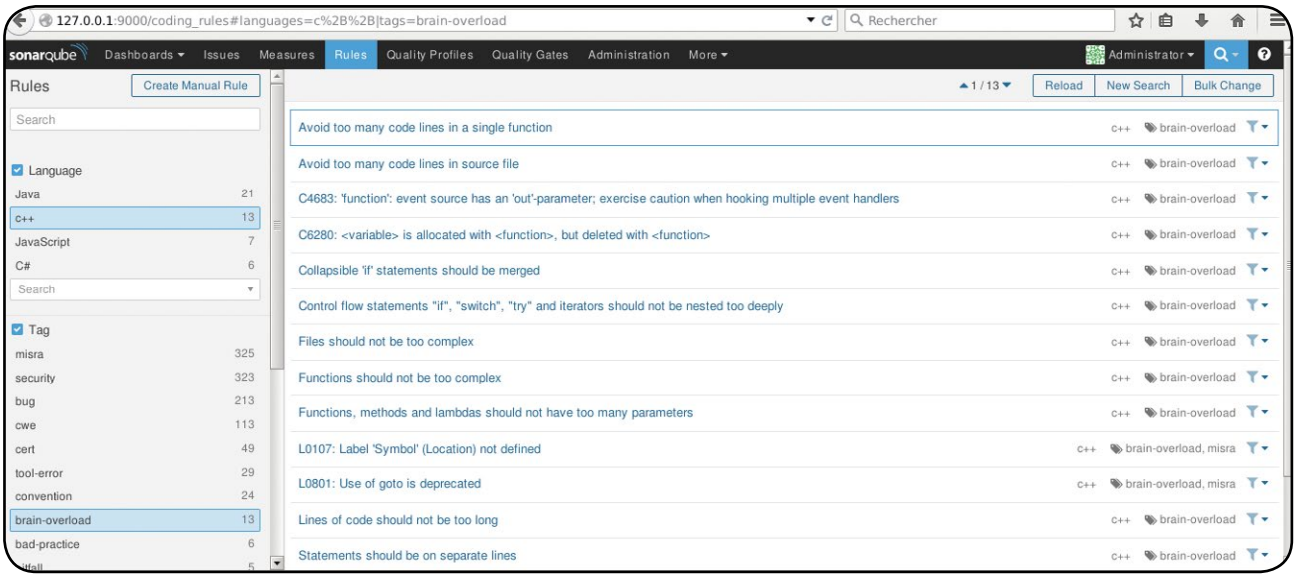


Fig. 2 : Onglet « rules » de SonarQube. La barre latérale gauche donne la liste des langages avec les filtres. La partie principale liste l'ensemble des règles disponibles pour le langage sélectionné en tenant compte des filtres.

Conclusion

Les standards de codage sont extrêmement importants dans les projets communautaires, *open source* ou évolutifs. Il faut bien noter ici que, pour un projet, le choix d'un standard doit se faire par rapport à l'équipe, au langage et même au système de management. On trouve aussi d'autres standards de codage créés par des entreprises comme celui de Google pour le code Java **[16]** (*Google Java Coding Standard*). Pour finir, on dira que le plus important est de fixer un ensemble de règles avant de commencer à programmer pour un projet. ■

Références

[1] Wikipédia, *« Règles de codage »* : https://fr.wikipedia.org/wiki/R%C3%A8gles_de_codage

[2] Wikipédia, *« Qualité logicielle »* : https://fr.wikipedia.org/wiki/Qualit%C3%A9_logicielle

[3] Institut d'électronique et d'informatique Gaspard-Monge, *« Qualité logicielle »* : http://www-igm.univ-mlv.fr

[4] Ensiwiki, *« Écrire un code de qualité »*, http://ensiwiki.ensimag.fr/index.php/%C3%89crire_du_code_de_qualit%C3%A9

[5] ISO/IEC 9126-1, *« Information technology-Software product quality- Part 1 : Quality Model »*, décembre 2001

[6] ISO/IEC 15939, *« Information technology-Software engineering-Software measurement process »*, Sept. 2001

[7] Wikipédia, *« Maintenabilité »* : https://fr.wikipedia.org/wiki/Maintenabilit%C3%A9

[8] Wikipédia, *« Indice de maintenabilité»* : https://fr.wikipedia.org/wiki/Indice_de_maintenabilit%C3%A9

[9] Projet CodeMeter, *« Maintainability Index(MI) »* : http://www.projectcodemeter.com/cost_estimation/help/GL_maintainability.htm

[10] Wikipédia, *« Notation hongroise »* : https://fr.wikipedia.org/wiki/Notation_hongroise

[11] Wikipédia, *« Métriques d'Halstead »* : https://fr.wikipedia.org/wiki/M%C3%A9triques_d'Halstead

[12] STALLMAN R. et al, *« GNU Coding Standards »*, dernière mise à jour le 23 avril 2015

[13] MORÈRE Y., *« Internationaliser/Régionaliser vos programmes C »*, *GNU/Linux Magazine HS n°70*

[14] Site Web officiel de SonarQube : http://www.sonarqube.org

[15] Wikipédia, *« SonarQube »* : https://fr.wikipedia.org/wiki/SonarQube

[16] D. BIENVENIDO (traduction N. FRANKEL), *« Les standards de codage Google »* : http://www.infoq.com/fr/news/2014/02/google-java-coding-standards, 14 février 2014

ACTUELLEMENT DISPONIBLE

HACKABLE N°15 !

CONTRÔLEZ VOTRE **ARDUINO** DEPUIS VOTRE **SMARTPHONE !**

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com


PRÉPARER UN CODE QR

Nicolas PATROIS - [Enseignant dans le secondaire]

Les articles précédents [1-6] vous ont donné envie de créer vos propres codes QR ? Ce sera chose faite ici même, pour le moment on prépare les données binaires.

Mots-clés : Python, Programmation orientée objet, Correction d'erreur, Reed-Solomon, Algorithme

Résumé

Dans cet article, on transforme le contenu d'un fichier en des données binaires directement collables dans un code QR.

Maintenant que nous savons lire le contenu des codes QR, même s'ils sont un peu entachés d'erreurs ou décorés d'un logo, nous allons faire l'inverse : créer un code QR avec un petit logo au milieu. N'en abusez pas, un code QR n'est lisible en pratique que par une application dédiée. Moi qui n'ai pas de téléphone portable, je déteste qu'on me propose un code QR tout nu sans explication comme au minimum l'adresse de la page *ouèbe*.

J'ai utilisé Wikiversity **[7]** et le *QR-code tutorial* **[8]** pour rédiger cet article. Les codes et fichiers sont sur le **GitHub** du magazine.

1 Préparation du tableau

On va reprendre en sens inverse (à peu près), les étapes de la lecture d'un code QR. Les fonctions et constantes communes ou utiles sont factorisées dans **`qrcodeoutils.py`** et dans **`qrcodestandard.py`**, je n'expliquerai leur contenu que s'il n'a pas déjà été étudié précédemment.

1.1 Initialisation

On va reprendre la structure de la classe **`qrdecode.py`** qui lit une image, mais en partant d'un fichier quelconque.

```
01: #!/usr/bin/python3
02:
03: from PIL import Image
04: from sys import exit,stderr
05: from qrcorps import *
06: from qrcodestandard import *
07: from argparse import *
```

On importe les mêmes bibliothèques que lors de la lecture.

```
09: class qrencode:
10:
11:   def __init__(self):
12:     [self.arguments,self.nivcor,self.masque,self.message,self.
mode,self.longclair,
13:       self.version,self.longtoutclair,self.clair,self.longueur,
14:       self.tout,self.entrelac,self.dim,self.tabmat,self.
gris,self.bontab]=[None]*16
```

Les noms des attributs sont encore parlants, ils sont donnés dans l'ordre de leur apparition dans le code.

1.2 Gestion des options

Je réutilise la bibliothèque **`argparse`** pour les options de la ligne de commandes.

```
01:   def options(self):
02:     parser=ArgumentParser(description="Crée un code QR.")
03:     parser.add_argument("-m",choices=["0","1","2","3","4","5",
"6","7"],help="Force le choix du masque.")
04:     parser.add_argument("-i",required=True,metavar="fichier",
```

```
help="Fichier texte d'entrée.")
05:     parser.add_argument("-c",metavar="image",help="Imagette
placée au milieu du code.")
06:     parser.add_argument("-t",type=int,metavar="N",default=4,
help="Taille d'un module.")
07:     parser.add_argument("-o",required=True,metavar="image",
help="Image de sortie.")
08:     parser.add_argument("-n",choices=["L","M","Q","H"],
default="L",help="Niveau de correction.")
09:     parser.add_argument("-a",choices=["0","1"],default="0",
help="Afficher l'image ou non.")
10:     self.arguments=parser.parse_args()
```

Ici :

- **-m** permet d'imposer le choix du masque, en son absence, le calcul du meilleur est effectué ;
- **-i** est le nom du fichier d'entrée (texte ou binaire) ;
- **-c** est le nom d'une éventuelle imagette décorative (canal alpha fonctionnel) collée au milieu de l'image du code QR ;
- **-t** précise la taille d'un module en pixels ;
- **-o** est le nom de l'image de sortie ;
- **-n** le niveau de correction (de *Low* à *High*) ;
- **-a** fait afficher l'image *via* la bibliothèque PIL **[9]**.

Voici ce qu'affiche le programme en cas d'absence d'entrées :

```
> ./qrencode.py
usage: qrencode.py [-h] [-m {0,1,2,3,4,5,6,7}] -i fichier [-c
image] [-t N] -o image [-n {L,M,Q,H}] [-a {0,1}]
qrencode.py: error: the following arguments are required: -i, -o
```

Et l'aide :

```
> ./qrencode.py -h
usage: qrencode.py [-h] [-m {0,1,2,3,4,5,6,7}] -i fichier [-c
image] [-t N] -o image [-n {L,M,Q,H}] [-a {0,1}]

Crée un code QR.

optional arguments:
  -h, --help            show this help message and exit
  -m {0,1,2,3,4,5,6,7}  Force le choix du masque.
  -i fichier            Fichier texte d'entrée.
  -c image              Imagette placée au milieu du code.
  -t N                  Taille d'un module.
  -o image              Image de sortie.
  -n {L,M,Q,H}          Niveau de correction.
  -a {0,1}              Afficher l'image ou non.
```

Tâchons maintenant d'ouvrir le fichier donné en entrée et de traduire les options dans nos attributs.

```
12:   def entrees(self):
13:     if self.arguments is None:
14:       self.options()

16:     if self.arguments.t<=0:
17:       print("Il faut choisir une taille de module positive.")
18:       exit(1)
19:     self.nivcor=self.arguments.n
20:     if self.arguments.m is None:
21:       self.masque=None
22:     else:
23:       self.masque=int(self.arguments.m)
```

En fait, **arguments** est une instance de la classe **argparse.Namespace** et chacun de ses attributs est une option. Ainsi, **arguments.n** contient la chaîne de caractères pour le niveau de correction (stocké dans **nivcor**) et **arguments.m** une chaîne qui sera convertie en un entier pour le **masque** s'il y a lieu.

Note

J'omettrai systématiquement le préfixe **self.** dans les noms des attributs et des méthodes donnés en explication.

Si l'on cherche à entrer une taille de module négative ligne 16 (les autres tests sont gérés par **argparse**), on sort.

```
24:     self.message=""
25:     try:
26:       with open(self.arguments.i,"r") as f:
27:         for l in f:
28:           self.message=self.message+l
29:     except IOError:
30:       print("Fichier %s inaccessible."%self.arguments.i)
31:       exit(1)
```

On teste si le fichier existe et est accessible (on lit alors le fichier ligne à ligne, on stocke son contenu binaire dans **message**), sinon on se plaint et on s'en va.

1.3 Détermination du mode

Selon le contenu du fichier envoyé, on détermine s'il faut utiliser le mode numérique, alphanumérique ou binaire (en fait, utf-8 et je rappelle que le mode kanji est laissé de côté, il est inclus dans le mode binaire).

```
01:   def carac(self):
02:     if None in [self.nivcor,self.masque,self.message]:
03:       self.entrees()

05:     caracteres=set(self.message)

07:     if self.message.isdecimal():
08:       self.mode=0
```

```
09:       q,r=divmod(len(self.message),3)
10:       self.longclair=10*q
11:       if r==1:
12:         self.longclair+=4
13:       elif r==2:
14:         self.longclair+=7
```

La méthode **isdecimal** retourne **True** si une chaîne de caractères ne contient que des chiffres décimaux. Si c'est le cas, on calcule le nombre de bits nécessaires pour les stocker.

Je rappelle qu'on utilise le fait que 10 bits suffisent à coder un nombre de trois chiffres, 7 pour un nombre à deux chiffres et 4 pour un nombre à un chiffre (voir les articles sur la lecture de codes QR **[1-3]**).

```
15:     elif caracteres.issubset(set(alphanum)):
16:       self.mode=1
17:       q,r=divmod(len(self.message),2)
18:       self.longclair=11*q+r*6
```

Dans **qrcodestandard.py**, **alphanum="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ $%*+-./:"**. On teste donc si tous les caractères y sont et dans ce cas, on calcule encore le nombre de bits nécessaires.

```
19:     else:
20:       self.mode=2
21:       self.message=self.message.encode("utf-8")
22:       self.longclair=len(self.message)*8
```

Enfin, dans le cas binaire ou kanji, on convertit le message unicode en une liste d'octets. En effet, **len("☺")** retourne **1** en Python 3 alors qu'il faut trois octets pour l'écrire (**hex(ord("☺"))** retourne **'0x1f603'**).

1.4 Détermination de la version

On va déterminer la taille du code QR, c'est-à-dire le nombre de modules total du carré.

```
01:   def detversion(self):
02:     if None in [self.mode,self.longclair]:
03:       self.carac()
04:     for self.version in tableau:
05:       self.longtoutclair=8*sum(blocs(tableau[self.version]
[self.nivcor])[1::2])
06:       s=self.longtoutclair-4-longbin(self.version,self.mode)
07:       if s>self.longclair:
08:         break
```

blocs est une fonction présente dans **qrcodestandard.py**, présentée lors de la lecture :

```
def blocs(l):
  return list(eval(l.replace("x","*").replace(")",")+")))
```

Elle transforme le contenu brut du tableau en un équivalent exploitable par Python. Ici, elle transforme par exemple **"2×(139,111),7×(140,112)"** (version 22 et niveau de correction *Low*) en la liste **[139,111,139,111,141,112,141,112,141,112,141,112,141,112,141,112,141,112]**. On additionne ensuite les termes d'indice impairs (les longueurs des blocs de données, soit 2×111+7×112) et on multiplie par 8 pour obtenir le nombre total de bits utilisables ligne 5. On a donc le nombre de bits du message (**longclair**), à quoi il faut ajouter les quatre bits du mode et les bits codant la longueur du message ligne 6 (**longbin** est expliquée dans l'article sur le décodage **[3]**) pour tester si l'ensemble rentre ou non dans le code QR de la version testée de contenance **longtoutclair**.

Tant que le message est trop long, on passe à la version supérieure.

```
10:     if s<self.longclair:
11:       print("Le message est trop long, veuillez le raccourcir
ou choisir un niveau de correction moins fort.")
12:       exit(1)
```

Si le message est toujours trop long (je n'ai pas codé la possibilité de créer un message découpé dans plusieurs codes QR consécutifs), on se plaint et on s'en va.

2 Construction des données

Il est temps de construire les données à écrire dans le code QR, tout est prêt.

2.1 Construction de la liste des données

Une fois la version choisie, on peut coder l'en-tête du code QR : le mode puis la longueur en caractères et non en bits ou en octets du message. Il est immédiatement suivi du message et complété éventuellement par un quadruplet de **0** et d'octets de bourrage. La correction d'erreur suit.

```
01:   def donnees(self):
02:     if None in [self.longtoutclair,self.version]:
03:       self.detversion()
04:     self.clair=[0]*(3-self.mode)+[1]+[0]*self.mode
05:     self.longueur=dec2bin(len(self.message),longbin(self.
version,self.mode))
06:     self.clair+=self.longueur
```

Le mode est déterminé par la position du **1** dans le premier quadruplet et on code la longueur en sa valeur binaire.

```
08:     if self.mode==0:
09:       q,r=divmod(len(self.message),3)
10:       for i in range(q):
11:         self.clair+=dec2bin(int(self.message[i*3:i*3+3]),10)
12:       if r==1:
13:         self.clair+=dec2bin(int(self.message[-1]),4)
14:       elif r==2:
15:         self.clair+=dec2bin(int(self.message[-2:]),7)
```

Dans le mode numérique, on convertit chaque triplet de chiffres en sa valeur binaire ligne 10 en prenant soin de bien obtenir dix bits, c'est la raison d'être du deuxième argument de **dec2bin**. De même, un éventuel chiffre final est codé sur quatre bits et deux éventuels chiffres finaux sur sept bits.

```
16:     elif self.mode==1:
17:       q,r=divmod(len(self.message),2)
18:       for i in range(q):
19:         self.clair+=dec2bin(45*alphanum.index(self.message[2*i])\
20:                     +alphanum.index(self.message[2*i+1]),11)
21:       if r==1:
22:         self.clair+=dec2bin(alphanum.index(self.message[-1]),6)
```

On lit chaque paire de caractères pour coder la somme pondérée de leurs indices dans **alphanum** sur onze bits. Plus précisément, si i_1 est l'indice du premier symbole et i_2 celui du deuxième, alors on écrit la valeur en binaire de $45 \times i_1 + i_2$. S'il reste un caractère final, son indice est codé sur six bits.

```
23:     else:
24:       for b in self.message:
25:         self.clair+=dec2bin(b,8)
```

Enfin, dans le cas binaire, on code chaque octet en sa valeur binaire. Un message écrit en kanji utilisera ce mode.

```
27:   def fin(self):
28:     if None in [self.clair,self.longueur]:
29:       self.donnees()
30:     bourrage=dec2bin(0xec11,16)
31:     self.clair+=[0,0,0,0]
32: #     self.clair+=[0]*(8-len(self.clair)%8)
33:     while len(self.clair)<self.longtoutclair:
34:       self.clair+=bourrage
35:     self.clair=self.clair[:self.longtoutclair]
```

On termine le message par quatre zéros consécutifs, puis selon les sources, par un bourrage de zéros pour que la longueur du message soit un multiple de huit ou non. Enfin, on complète jusqu'à la longueur finale par une suite de **1110110000010001**, soit **ec11** en hexadécimal.

2.2 Calcul du code de Reed-Solomon

Le message est prêt à être découpé en blocs pour calculer les codes correcteurs de Reed-Solomon.

```
01:   def reedsolomon(self):
02:     if None in [self.clair,self.longueur]:
03:       self.fin()
04:     posclair,posredondant=court2blocs(tableau[self.version]
[self.nivcor])
```

court2blocs transforme la chaîne présente dans **tableau** (**"2×(139,111),7×(140,112)"** dans le cas d'une version 22 et de niveau de correction *Low*), la fonction est dans **qrcodestandard.py** :

```
def court2blocs(l):
  l1=blocs(l)
  for i in range(0,len(l1),2):
    l1[i:i+2]=l1[i:i+2][::-1]
  l2,l3=[0],[0]
  for i in range(len(l1)//2):
    l2.append(l2[-1]+l1[2*i])
    l3.append(l3[-1]+l1[2*i+1]-l1[2*i])
  return l2,l3
```

Elle retourne les positions des octets de début de chaque bloc de la partie en clair puis de la partie redondante, le dernier octet de la première liste est la longueur totale de la partie en clair, de même pour la partie redondante. Dans notre cas, elle retourne **[0, 111, 222, 334, 446, 558, 670, 782, 894, 1006] [0, 28, 56, 84, 112, 140, 168, 196, 224, 252]**.

```
06:     blocsclair=[]
07:     for i in range(len(posclair)-1):
08:       blocsclair.append(self.clair[8*posclair[i]:8*posclair[i+1]])
09:     blocsredondant=[]
```

Ici, on crée chaque bloc de données, le premier contient les 111 premiers octets donc les 888 premiers bits, son bloc correcteur contiendra 28 octets, soit 224 bits.

```
10:     for i in range(len(blocsclair)):
11:       bloc=blocsclair[i]
12:       longredondant=8*(posredondant[i+1]-posredondant[i])
13:       polyclair=message2poly(bloc+[0]*longredondant)
14:       modulo=Polynome.construction([1])
15:       for i in range(longredondant//8):
16:         modulo*=Polynome.construction([1,F256.exp(i)])
17:       blocsredondant.append(poly2message(polyclair%modulo))
```

On calcule le complément correcteur de chaque bloc en clair qui n'est que le reste d'une division euclidienne de polynômes qu'on concatène aux données en clair, voir le détail dans la section sur la correction des erreurs **[6]**.

```
19:     self.tout=[]
20:     for bloc in blocsclair:
21:       self.tout+=bloc
22:     for bloc in blocsredondant:
23:       self.tout+=bloc
```

On concatène tout en une seule liste, les blocs en clair à la queue leu leu suivis (dans le même ordre) par les blocs correcteurs.

2.3 Entrelacement du message binaire final

Pour qu'un petit dessin n'empêche pas la lecture du message d'un code QR, les données sont en fait entrelacées dès qu'un code QR est assez grand selon le tableau suivant :

Niveau de correction	Low	Medium	Quality	High
Entrelacement à partir de la version	6	4	3	3

On le voit par exemple dans le dictionnaire `tableau` de `qrcodestandard.py` où `tableau[5]["L"]` vaut `"(134,108)"` alors que `tableau[6]["L"]` vaut `"2×(86,68)"`.

Les octets des blocs en clair puis correcteurs sont mélangés et donc distribués partout dans l'image. Dans notre cas, on a neuf blocs. On écrit les octets des neuf blocs dans un tableau comme celui ci-dessous.

Le bloc n°0 contient les 111 premiers octets, soit les octets 0 à 110, le `–1` signifie qu'il faut sauter ce rang et passer au bloc suivant. La partie en clair est construite orthogonalement : on lit le tableau de haut en bas d'abord puis de gauche à droite, il s'agit en quelque sorte du tableau transposé de celui-ci en sautant les valeurs négatives. Ainsi, la partie en clair entrelacée contient les octets 0, 111, 222, 334, 446, 558, 670, 782, 894, 1, 112, 223; ... 892, 1004, 333, 445, 557, 669, 783, 893, 1005.

On fait la même chose pour les octets des blocs correcteurs qui ont tous la même taille.

```
01:   def entrelacement(self):
02:     if self.tout is None:
03:       self.reedsolomon()
04:
05:     posclair,posredondant=court2long(tableau[self.version]
[self.nivcor])
```

On récupère, dans le `tableau` (qui est un dictionnaire de dictionnaires), la chaîne qui nous intéresse `"2×(139,111),7×(140,112)"` et on la transforme en son équivalent qui ne contient que des coordonnées absolues (plus un zéro initial). Les données en clair et leurs corrections sont regroupées en blocs pas trop grands pour des raisons de temps de calcul. Par exemple, pour le niveau de correction L de la `version` 22, les données sont en deux blocs de 139 octets dont les 111 premiers contiennent la partie en clair (il reste donc 28 octets pour la correction dans chacun) et les sept blocs suivants contiennent 140 octets dont les 112 premiers pour la partie en clair (et toujours 28 pour la correction).

Voici le début du contenu de `tableau`, stocké dans `qrcodestandard.py` :

```
tableau={
  1:{"L":"(26,19)","M":"(26,16)","Q":"(26,13)","H":"(26,9)"},
  2:{"L":"(44,34)","M":"(44,28)","Q":"(44,22)","H":"(44,16)"},
  3:{"L":"(70,55)","M":"(70,44)","Q":"2×(35,17)","H":"2×(35,13)"},
  4:{"L":"(100,80)","M":"2×(50,32)","Q":"2×(50,24)","H":"4×(25,9)"},
  5:{"L":"(134,108)","M":"2×(67,43)","Q":"2×(33,15),2×(34,16)",
"H":"2×(33,11),2×(34,12)"},
  6:{"L":"2×(86,68)","M":"4×(43,27)","Q":"4×(43,19)","H":"4×(43,15)"},
[...]
 22:{"L":"2×(139,111),7×(140,112)","M":"17×(74,46)","Q":"7×(54,24),
16×(55,25)",[...]
```

Et ainsi de suite jusqu'à 40. J'ai graissé le niveau de correction illustré ci-dessus.

Plutôt que de transformer à la main les informations données dans **[7]**, j'ai préféré créer des fonctions *ad hoc* qui le font à ma place, elles aussi dans `qrcodestandard.py` :

```
def blocs(l):
  return list(eval(l.replace("×","*").replace("),",")+")))
```

Dans notre cas, `blocs` retourne `[139, 111, 139, 111, 140, 112, 140, 112, 140, 112, 140, 112, 140, 112, 140, 112, 140, 112]`, voir les articles précédents **[1-6]**.

Bloc	Octets								
n°0	0	1	2	3	...	108	109	110	–1
n°1	111	112	113	114	...	219	220	221	–1
n°2	222	223	224	225	...	330	331	332	333
...	...	...	...	...	...	...	...	...	...
n°7	782	783	784	785	...	890	891	892	893
n°8	894	895	896	897	...	1002	1003	1004	1005

Comme la manière initiale, relative, n'est pas très utilisable, je l'ai convertie, à l'aide de la fonction **court2long**, en des nombres absolus directement utilisables par l'opérateur crochets **[]** de Python. Je prends l'exemple du niveau de correction Q de la **version** 3.

```
def court2long(l):
  l1=blocs(l)
  for i in range(0,len(l1),2):
    l1[i:i+2]=l1[i:i+2][::-1]
```

On permute les valeurs deux à deux, **l1=[111, 139, 111, 139, 112, 140, 112, 140, 112, 140, 112, 140, 112, 140, 112, 140]** après cette étape.

```
  l2=[[-1]]
  for i in range(len(l1)//2):
    l2.append([1+j+l2[-1][-1] for j in range(l1[2*i])])
  del l2[0]
```

On récupère la taille de chaque bloc de données en clair et on crée pour chaque valeur, une liste qui donne les positions des bits de chaque bloc. Par exemple, **l2=[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,...,110],[111,112,113,114,...],[...,1002,1003,1004,1005]]**.

```
  l3=[]
  for i in range(max(len(l) for l in l2)):
    for j in range(len(l2)):
      try:
        l3.append(l2[j][i])
      except IndexError:
        l3.append(-1)
```

Ici, on crée l'enchevêtrement des bits des blocs : on écrit les listes ci-dessus les unes sur les autres dans un tableau et on le lit verticalement. Il s'agit donc d'un **zip** comme ci-dessus, mais on ne crée qu'une seule liste qui contient l'enchevêtrement. Ici, par exemple, **l3=[0,111,222,334,446,558,670,782,894,1,112,223, 335,447,...,892,1004,-1,-1,333,445,557,669,781,893,1005]**. Que se passe-t-il si les blocs n'ont pas la même longueur ? On remplace les valeurs manquantes par **-1**.

```
  l4=[[max(l3)]]
  for i in range(len(l1)//2):
    l4.append([1+j+l4[-1][-1] for j in range(l1[2*i+1]-l1[2*i])])
  del l4[0]
  l5=[]
  for i in range(max(len(l) for l in l4)):
    for j in range(len(l4)):
      l5.append(l4[j][i])
  return l3,l5
```

Et on fait la même chose pour les blocs de la correction d'erreur sans vérification des tailles, ils ont toujours tous la même longueur.

```
06:     pos=posclair+posredondant
07:     self.entrelac=[0]*len(self.tout)
08:     j=0
09:     for i in range(len(self.entrelac)//8):
10:       if pos[i]!=-1:
11:         self.entrelac[8*j:8+8*j]=self.tout[8*pos[i]:8+8*pos[i]]
12:         j+=1
```

Enfin, le j-ième octet du message entrelacé (partie en clair et correction) est le **pos[i]**-ième du message non entrelacé. **i** est la position dans la sortie de **court2long** alors que **j** est la position dans le message entrelacé, c'est pour cette raison que **j** n'est incrémenté que si **pos[i]** indique la position d'un octet (**-1** indique qu'on est à la fin d'un bloc plus court).

Conclusion

On a calculé les blocs correcteurs de Reed-Solomon, les blocs sont entrelacés et le message binaire brut est prêt à être déversé dans notre matrice. Ce sera l'objet de notre tout dernier article en page suivante où nous allons dessiner notre code QR avec une petite cerise sur le gâteau. ■

Références

[1] PATROIS N., *« Expliquer un code QR »*, *GNU/Linux Magazine n°193*, p. 22 à 26

[2] PATROIS N., *« Déchiffrer un code QR »*, *GNU/Linux Magazine n°193*, p. 28 à 31

[3] PATROIS N., *« Décoder un code QR »*, *GNU/Linux Magazine n°194*, p. 32 à 41

[4] PATROIS N., *« Expliquer un corps fini »*, *GNU/Linux Magazine n°195*, p. 16 à 19

[5] PATROIS N., *« Fabriquer un corps fini »*, *GNU/Linux Magazine n°196*, p. 32 à 37

[6] PATROIS N., *« Réparer un code QR »*, GNU/Linux Magazine n°198, p. 34

[7] *Reed–Solomon codes for coders* : https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders et le complément plus précis https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders/Additional_information.

[8] *QR Code tutorial* : http://www.thonky.com/qr-code-tutorial/

[9] *The Python Imaging Library Handbook* : http://effbot.org/imagingbook/.


DESSINER UN CODE QR

Nicolas PATROIS - [Enseignant dans le secondaire]

Et voilà le dernier article de cette longue série sur les codes QR. À son issue, nous saurons les dessiner.

Mots-clés : Python, Code QR, Programmation orientée objet, Algorithme

Résumé

Dans l'article précédent, nous avons préparé les données binaires, il reste à créer la matrice, le masquage et l'image. Nous reprendrons à l'envers la lecture des premiers articles.

Les données sont créées et prêtes à être rangées dans la matrice. Le code et les fichiers sont sur le GitHub du magazine.

1 Création du tableau

1.1 Les invariants

On commence par placer les zones obligatoires : les grandes cibles, les petites si nécessaire, les pointillés et le module toujours noir.

```
01:   def matrice(self):
02:  if self.entrelac is None:
03:   self.entrelacement()

05:   self.dim=17+4*self.version
06:   self.tabmat=[[0 for _ in range(self.dim)] for _ in range(self.dim)]
```

On crée d'abord la matrice carrée de dimensions `17+4×version`.

```
07:     for i in range(self.dim):
08:       self.tabmat[6][i]=1-i%2
09:       self.tabmat[i][6]=1-i%2
```

On place les échelles (les lignes noires et blanches en haut et à gauche).

```
10:     for i in range(7):
11:       self.tabmat[i][:7]=cible[i]
12:       self.tabmat[-i-1][:7]=cible[i]
13:       self.tabmat[i][-7:]=cible[i]
```

On place les trois grandes cibles dans tous les coins sauf celui en bas à droite.

```
14:     self.tabmat[-8][8]=1
```

On place le module toujours noir en haut de la cible inférieure.

```
15:     liste=minicibles[self.version]
16:     for ci in liste:
17:       for cj in liste:
18:         if (ci,cj) not in {(6,6),(6,max(liste)),(max(liste),6)}:
19:           for i in range(ci-2,ci+3):
20:             self.tabmat[i][cj-2:cj+3]=minicible[i+2-ci]
```

Et on place les mini-cibles sauf si elles chevauchent les grandes.

1.2 Remplissage du tableau

On a encore besoin de préparer la zone où le petit serpent dépose ses données.

```
01:   def remplissage(self):
02:     if None in [self.dim,self.tabmat]:
03:       self.matrice()

05:     self.gris=griser(self.dim,self.version)
06:     i,j=self.dim-1,self.dim-1
07:     dire=-1
08:     k=0

10:     while k<len(self.entrelac):
11:       if self.gris[i][j]:
12:         self.tabmat[i][j]=self.entrelac[k]
13:         k+=1
14:       i,j,dire=suivant(i,j,dire,self.dim)
```

C'est à peu près le même code que pour la lecture, mais on écrit au lieu de lire. Je rappelle qu'on parcourt les modules disponibles (ceux pour lesquels **gris[i][j]** vaut **True**) en serpentant dans des colonnes de deux modules de largeur en partant du bas de la grille.

1.3 Calcul du code de contrôle de la version

Il s'agit ici de placer les deux rectangles de 3×6 modules présents uniquement si la **version** est supérieure ou égale à 7. L'un est placé au-dessus de la cible inférieure entre l'échelle et le bord, l'autre est placé verticalement à gauche de la cible de droite.

```
01:  def codecontrolev7(self):
02:    if None in [self.dim,self.tabmat]:
03:      self.remplissage()

05:    if self.version>=7:
06:      pol=[int(i) for i in "111110010010l"]
07:      liste=dec2bin(self.version,6)
08:      liste+=[0]*12
09:      while liste[0]==0:
10:        del(liste[0])
11:      while len(pol)<len(liste):
12:        pol.append(0)
13:      while len(liste)>12:
14:        liste=[i^j for (i,j) in zip(liste[:len(pol)],pol)]+
liste[len(pol):]
15:        while liste[0]==0:
16:          del(liste[0])
17:      liste=[0]*(12-len(liste))+liste
18:      liste=dec2bin(self.version,6)+liste
19:      liste.reverse()
```

Vous avez reconnu une division euclidienne dans $\mathbb{F}_2$ similaire à celle de la zone de formats, mais par le polynôme $X^{12}+X^{11}+X^{10}+X^9+X^8+X^5+X^2+1$ des lignes 13 à 16. Ensuite, on bourre avec des **0** pour obtenir exactement douze bits : six bits de version ($40 \leqslant 2^6 = 64$) plus douze bits de correction font bien dix-huit bits au total.

```
21:        for i in range(3):
22:          self.tabmat[-11+i][:6]=liste[i::3]
23:        for i in range(6):
24:          self.tabmat[i][-11:-8]=liste[3*i:3*i+3]
```

On place la zone inférieure puis la zone supérieure.

1.4 Calcul du meilleur masque et placement des zones de formats

1.4.1 Choix du masque

Il reste à choisir le meilleur masque parmi les huit possibles pour éviter des cibles mal placées, un déséquilibre trop criant entre le blanc et le noir ou des zones monochromes trop grandes.

```
01:   def choixmasque(self):
02:     if None in [self.dim,self.tabmat]:
03:       self.codecontrolev7()

05:     cor=correctionniveau[self.nivcor]
06:     dmin=None

08:     for m in range(8):
09:       if self.masque is not None and self.masque!=m:
10:         continue
```

Si le **masque** est imposé par l'option **m**, on ne calcule que pour ce **masque**. Sinon, on teste les huit masques un par un et on choisit le meilleur en tenant compte des zones de formats.

```
11:       tab=[list(l) for l in self.tabmat]
12:       ma=tuple(dec2bin(m,3))
13:       forma=cor+ma
14:       forma=forma+tuple(dec2bin(resteformat(bin2dec(forma)),10))
15:       forma=[i^j for (i,j) in zip(forma,masquef)]
16:       tab[8][:6]=forma[:6]
17:       tab[8][7:9]=forma[6:8]
18:       tab[7][8]=forma[8]
19:       for i in range(6):
20:         tab[i][8]=forma[-1-i]
21:       tab[8][-8:]=forma[-8:]
22:       for i in range(7):
23:         tab[-i-1][8]=forma[i]
```

Pour chaque masque **m**, on place les zones de formats correspondantes dans **tab**, une copie de la matrice **tabmat**.

```
24:       masquet=masques[ma]
25:       for i in range(self.dim):
26:         for j in range(self.dim):
27:           if self.gris[i][j]:
28:             tab[i][j]^=masquet(i,j)
```

On effectue le masque binaire sur **tab**.

```
29:       mal=malus(tab)
30:       if dmin is None or mal<dmin:
31:         dmin=mal
32:         self.bontab=tab
```

On calcule le **malus** avec la fonction idoine. Si on trouve mieux ou si le malus n'est pas encore défini, on stocke le plus petit malus dans **dmin** et la meilleure matrice dans **bontab**.

1.4.2 Calcul du malus

Détaillons le calcul du malus pour une matrice. Il existe quatre malus différents qui s'accumulent. On peut remarquer qu'une matrice de code QR ne peut pas avoir de malus nul.

```
def malus(table):
 m=0

 def m1(ligne):
   ch="".join(map(str,ligne))
   c=0
   for cinq in ["11111","00000"]:
     i=-1
     while i<len(ch):
       i+=1
       if cinq in ch[i:]:
         i+=ch[i:].index(cinq)
         c-=2
         while ch[i]==cinq[0]:
           i+=1
           c+=1
           if i==len(ch):
             break
   return c
```

Le malus des suites constantes trop longues : une suite de cinq modules ou plus de la même couleur est pénalisée de sa longueur moins **2**, autrement dit, **1111111** (sept **1**) est pénalisé de **5** points et **0000000000** (dix **0**) est pénalisé de **8** points. La fonction le calcule pour une seule ligne.

Le pointeur **i** se place à la première position où on a repéré le motif, mais seulement à partir de **i** dans le but de ne pas compter plusieurs fois le même motif. À partir de cette position, on compte le nombre de bits successifs identiques qu'on ajoute au malus **c**.

```
for l in table:
  m+=m1(l)
for i in range(len(table)):
  m+=m1([l[i] for l in table])
```

On additionne le malus pour chaque ligne puis pour chaque colonne.

```
def m2(table):
 c=0
 for i in range(len(table)-1):
   for j in range(len(table)-1):
    c+=3*(table[i][j]==table[i+1][j]==table[i][j+1]==table[i+1][j+1])
  return c
 m+=m2(table)
```

Chaque carré monochrome de **2×2** est pénalisé de **3** points, un carré monochrome de **3×3** contient donc quatre carrés de **2×2** et est pénalisé de **3×4=12** points.

```
 def m3(ligne):
   ch="".join(map(str,ligne))
   return 40*(ch.count("10111010000")+ch.count("00001011101"))

 for l in table:
   m+=m3(l)
 for i in range(len(table)):
   m+=m3([l[i] for l in table])
```

La présence du motif central des cibles suivi ou précédé de quatre **0** est pénalisé de **40** points, horizontalement comme verticalement, vers la droite comme vers la gauche.

```
 def m4(table):
   m4n=sum(sum(l) for l in table)
   m4t=len(table)**2
   pourcent=100*m4n//m4t//5*5
   return min(abs(50-pourcent),abs(50-pourcent+5))//5*10

 m+=m4(table)
```

m4n compte le nombre de **1** dans la matrice, **m4t** le nombre de modules au total. On calcule le pourcentage de **1** arrondi par défaut et par excès à **5** près. Le malus est l'écart entre le pourcentage arrondi le plus près de **50** multiplié par **2**.

```
 return m
```

Et on retourne le malus total.

2 Création de l'image

La matrice est prête, il reste à déverser son contenu dans une image.

2.1 Création et sauvegarde

```
01:   def creation(self):
02:     if self.bontab is None:
03:       self.choixmasque()

05:     self.taille=(8+self.dim)*self.arguments.t
06:     im=Image.new("RGB",(self.taille,self.taille),"white")
07:     blanc=(255,255,255)
08:     self.image=[blanc]*self.taille*4*int(self.arguments.t)
```

On crée **image**, la matrice des pixels qui doit avoir quatre modules blancs supplémentaires tout autour, d'où le **8+dim** ligne 05. Pour le moment, elle ne contient que la première ligne blanche de quatre modules de haut. On crée aussi **im**, l'image qui sera gérée par **PIL**, c'est un carré de fond blanc.

```
09:     for i in range(self.dim):
10:       for _ in range(self.arguments.t):
11:         self.image.extend([blanc]*4*self.arguments.t)
12:         for j in range(self.dim):
13:           couleur=255-255*self.bontab[i][j]
14:           self.image.extend([(couleur,couleur,couleur)]*self.
arguments.t)
15:         self.image.extend([blanc]*4*self.arguments.t)
16:     self.image.extend([blanc]*self.taille*4*self.arguments.t)
```

On ajoute quatre modules blancs au début de chaque ligne, on crée les modules de largeur **arguments.t** et on termine par quatre modules blancs. On crée **arguments.t** lignes identiques pour que les modules soient bien carrés et on termine par les quatre lignes de modules blancs.

```
17:     im.putdata(self.image)
```

Et on déverse l'**image** dans **im**, c'est plus rapide ainsi. PIL s'occupe lui-même de gérer les dimensions réelles de l'image, car **image** est en fait une seule liste et non une liste de listes comme **bontab**.

```
18:     try:
19:       logo=Image.open(self.arguments.c)
20:       m=max(logo.width,logo.height)
21:       logo=logo.resize((self.taille//4*logo.width//m,self.
taille//4*logo.height//m),\
                Image.ANTIALIAS)
22:       im.paste(logo,(self.taille//2-logo.width//2,self.
taille//2-logo.height//2),logo)
23:     except AttributeError:
24:       pass
25:     except IOError:
26:       print("Fichier %s inaccessible."%self.arguments.c)
27:       exit(1)
```

Si l'imagette est donnée en ligne de commandes, on la colle, réduite, lissée et avec transparence, au milieu du code QR. Elle prend, dans sa plus grande dimension, le quart du code QR pour ne pas le rendre illisible. Si l'imagette n'est pas définie, on passe et si son fichier n'est pas lisible (inexistant, ou droits insuffisants), on s'arrête.

```
29:     if self.arguments.a=="1":
30:       im.show()
31:     im.save(self.arguments.o)
```

Si on a demandé à afficher le code QR, on le fait via **xv** sous Unix selon la documentation ou **paint** sous Windows. Enfin, on sauvegarde l'image.

2.2 Résultat

Testons notre beau code.

```
01: if __name__=="__main__":
02:   code=qrencode()
03:   code.creation()
```

Et voilà, la figure 1 est le résultat de la commande suivante où **linus.txt** est le message de Linus décodé précédemment et **Tux.svg.png** notre manchot préféré :

```
> ./qrencode.py -i linus.txt -o linux2.png -t 2 -c Tux.svg.png
```

Un œil exercé verra une petite différence entre cette image et celle lue dans l'article sur la correction, tout en bas à gauche, à la fin des blocs de correction. J'ai un léger bogue, mais comme les codes QR sont auto-correcteurs...

```
> ./qrdecode.py -i linux2.png -a1
Fichier : linux2.png
Niveau de correction : Low
Masque :
 ..  ..
 ..  ..
 ..  ..
 ..  ..
 ..  ..
 ..  ..
Dimensions : 105×105
Version : 22
Il y a eu besoin de corriger 44 erreur(s) dans la zone de données.
Mode : Byte
Longueur du message : 949
Message :
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
```

Attention aussi au tout dernier caractère du fichier en entrée, si vous voulez obtenir la même chose que https://www.the-qrcode-generator.com/, vous devez veiller à supprimer le dernier retour à la ligne (ouvrez votre fichier avec un éditeur hexadécimal comme **ghex** pour comprendre) ou à l'ajouter dans le texte copié sur le site.

Fig. 1 : Le code QR obtenu.

Conclusion

La création de cette image prend sept secondes, c'est beaucoup trop. Une étude sérieuse du code permettrait de trouver où l'améliorer et quelles parties du code sont lentes. ■

Pour aller plus loin

Par ailleurs, il reste à coder le dernier mode (kanji) ainsi que les autres codages de texte et les codes QR à long contenu partagé. La lecture pourrait aussi s'accommoder de codes QR dont les modules ne sont plus carrés ainsi que de codes QR photographiés ou filmés (donc pas exactement vus de face).

Enfin, on peut s'attaquer aux autres versions des codes QR (micro QR et version 1) comme aux autres codes à barres comme le code Aztec, le MaxiCode ou le DataMatrix.

KNOCK KOCK KNOCK'IN ON HEAVEN'S DOOR

Arnaud FÉVRIER - [Maître de conférences en Informatique, Aix Marseille Université, équipe eRISCS]

Dans les années 80, nous nous connections sur les ordinateurs distants en utilisant telnet ou rlogin. Ces protocoles souffrent de leur ancienneté. En particulier, ils n'utilisent pas de technique de chiffrement. Depuis, les connexions à distance reposent principalement sur ssh ou un autre VPN. Pour aller encore plus loin, il est possible de dissimuler les services avec la technique de port knocking.

Mots-clés : port knocking, sécurité, firewall, ssh

Résumé

Nous commençons par présenter la technique de *port knocking* utilisée par knockd. Ensuite, nous montrons comment installer et surtout configurer un client et un serveur. Enfin, nous présentons une brève analyse de la sécurité de cette technique.

Avec Internet il est possible d'avoir accès à son (ou ses) réseau(x) de partout. Il n'est plus nécessaire d'être devant l'ordinateur pour se connecter à celui-ci et effectuer les tâches désirées. En fait, même avant Internet, il était déjà possible de se connecter à distance sur les ordinateurs pour travailler. Dans les années 50, un terminal et un modem fournissaient l'accès aux *mainframes* depuis de longues distances. Dans les années 80, le réseau commençait à s'étendre et les stations de travail Unix offraient déjà des possibilités de connexion graphique à distance à partir d'un serveur telnet ou rlogin.

Avec la généralisation des ordinateurs et les facilités d'accès, les protocoles de connexion en clair ne sont plus considérés comme sûrs quand la totalité du réseau parcouru n'est pas sécurisée par d'autres moyens. Un simple scan du réseau peut ainsi facilement récupérer les *logins* et mots de passe utilisés et fournir ainsi des connexions illégitimes à des malfaisants.

Internet a alors introduit des possibilités cryptographiques assurant la sécurité des communications et la confidentialité des *logins* et surtout des mots de passe. Ceux-ci deviennent même inutiles, remplacés par des clefs beaucoup plus sécurisantes, mais c'est le sujet d'autres articles. Il devient donc aisé d'utiliser soit un protocole chiffré, comme ssh par exemple, soit d'utiliser une couche de chiffrement (OpenVPN, TLS, IPsec) pour se connecter en toute confiance.

Prenons l'exemple d'un serveur ssh. La configuration classique permet de se connecter en utilisant le couple *login* - mot de passe qui autorise la connexion en local. Les utilisateurs souffrent de difficultés avec les mots de passe et de nombreuses techniques permettent de les trouver, soit parce qu'ils sont faibles, soit parce qu'ils sont mal protégés. De plus, il est possible que l'implémentation ou le protocole souffre d'une vulnérabilité permettant à un attaquant de se connecter.

Les malandrins tentent en permanence de découvrir des serveurs ssh mal protégés pour s'introduire sur de nouveaux ordinateurs. Pour cela, ils scannent l'intégralité du réseau à la recherche de serveurs mal configurés ou vulnérables. Le scan de ports est une réalité. Sur un serveur ssh exposé sur Internet, le fichier `/var/log/auth.log` montre les connexions ayant échouées :

```
Sep  1 06:55:05 framboise sshd[13832]: Failed password for
invalid user pi from 91.226.50.92 port 50893 ssh2
Sep  1 06:55:05 framboise sshd[13832]: Failed password for
invalid user pi from 91.226.50.92 port 50893 ssh2
```

Le fichier montre l'adresse IP source de l'attaque (ici, une adresse en Pologne) et l'utilisateur testé. Le succès commercial des Raspberry Pi, avec le login par défaut augmente la probabilité de trouver des machines vulnérables. Il faut donc éviter de laisser les utilisateurs par défaut des systèmes préconfigurés. Sur ce serveur, il y a entre 100 000 et 400 000 tentatives de connexions par mois (et pas par moi)!

Nous présentons dans cet article la technique du *port knocking* utilisée par **knockd** **[1]**. Ensuite, nous présentons l'installation facile du serveur et du client et présentons les méthodes principales : ouverture ou fermeture d'un port, automatisation de la fermeture puis utilisation de séquences uniques pour empêcher le rejeu des séquences. Enfin, nous présentons quelques aspects de sécurité liés à knockd.

1 Port Knocking

Nous allons présenter comment fonctionne knockd. Nous partons d'une configuration classique qui consiste à protéger un serveur Internet. Ce serveur sert un site web avec **Apache**, par exemple. Le service Internet est donc disponible sans restriction. Pour des raisons d'administration, ce serveur doit être accessible en ssh.

Nous allons interdire l'accès ssh pour l'autoriser uniquement en cas de besoin. Pour cela, nous allons utiliser le *firewall* Linux. Nous bloquons tous les accès sur tous les ports sauf, évidemment, le port **80** pour notre serveur web comme dans la figure 1.

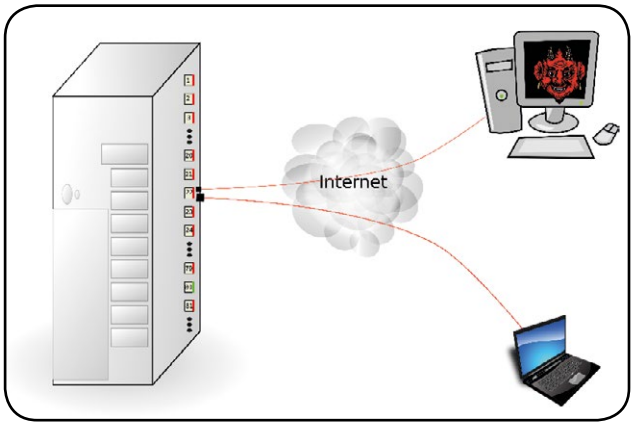


Fig. 1 : Un serveur web, protégé par son firewall.

Les ports verrouillés par le *firewall* sont en rouge. Le port ouvert (**80**) est en vert. Le portable de l'utilisateur nomade ne peut accéder au port ssh. Les malfaisants qui scannent le port ssh (**22**) à la recherche d'une version vulnérable ou assortie d'un mot de passe faible ne voient rien.

L'utilisateur nomade veut accéder au serveur ssh de son serveur. Il va donc lancer la procédure de *port knocking*. Il va utiliser un secret partagé entre le serveur et lui : la liste des ports à utiliser. Dans la figure 2, le nomade lance la séquence : port **3**, puis port **81** et enfin port **24**. Bien entendu, si le poste nomade dispose d'un vrai accès internet, alors il peut utiliser n'importe quelle succession des **65535** ports TCP ou UDP.

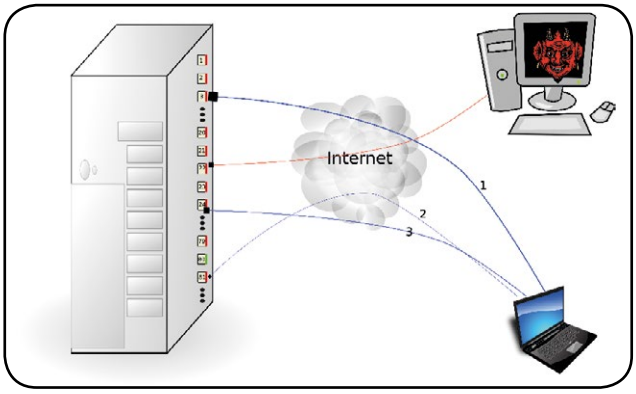


Fig. 2 : Le nomade envoie la séquence secrète de paquets.

Le serveur knockd va modifier les règles du *firewall* pour autoriser la connexion depuis l'adresse IP (publique) du poste nomade. L'utilisateur va pouvoir se connecter en ssh. Comme il s'agit d'une connexion utilisant le protocole de transport TCP, nous allons pouvoir refermer l'accès rapidement en laissant la communication établie, comme dans la figure 3. Même si le malfaisant est sur le même réseau privé que le nomade, un collègue indélicat par exemple, il ne pourra tenter des connexions que pendant les quelques secondes pendant lesquelles le port reste ouvert.

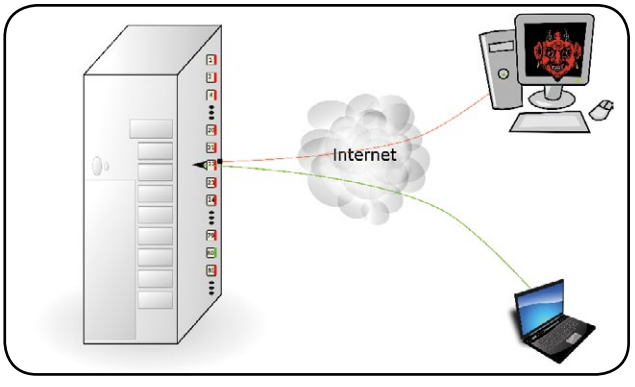


Fig. 3 : Le nomade est connecté en ssh, les malfaisants ne détectent toujours rien.

Rappels : TCP et firewall

Pour pouvoir profiter pleinement de knockd, il convient de connaître quelques éléments des réseaux et du *firewall*. Le modèle de réseau Internet **[2]** est un modèle en couches élaboré dans les années 70. Les applications (par exemple le Web ou la résolution de noms) utilisent un protocole de transport (TCP ou UDP) qui utilise un protocole réseau (IP) qui utilise lui-même un support physique (par exemple Ethernet) - voir figure 4. Le Web et ssh utilisent TCP et la résolution de noms utilise UDP. Le traitement des connexions TCP et des communications UDP sera différent pour le *firewall*. TCP fonctionne en mode orienté connexion. C'est-à-dire qu'une communication commence par une phase d'ouverture de connexion (trois paquets syn, syn-ack, ack). Si l'ouverture s'est bien passée alors les échanges peuvent avoir lieu (c'est le protocole au-dessus). Enfin, la communication peut être terminée avec trois (fin, fin-ack, ack) ou quatre paquets (fin, ack, fin, ack) comme dans la figure 5.

Le *firewall* **[3]** va pouvoir traiter les connexions TCP en deux règles : une règle autorisant ou non le paquet initial (syn) vers un service et une autre autorisant les paquets suivants pour une connexion considérée comme établie par le *firewall*.

UDP fonctionne différemment, il n'y a pas, comme pour TCP, une demande de connexion. Il est possible de filtrer les ports UDP en entrée, mais alors les réponses ne pourront pas être reçues. En particulier, les réponses aux interrogations DNS n'arriveront pas. Ceci peut bloquer les autres protocoles s'ils essaient de se connecter à un nom de machine. De nombreux programmes vont aussi essayer de connaître le nom associé à une adresse IP. Ceci peut ralentir les programmes qui vont attendre l'expiration des délais. Il convient donc de bien recevoir les réponses DNS.

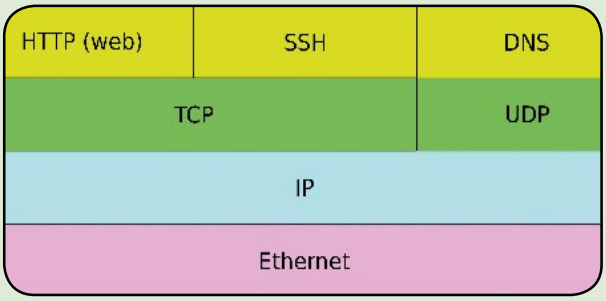


Fig. 4 : Le modèle TCP/IP en couches.

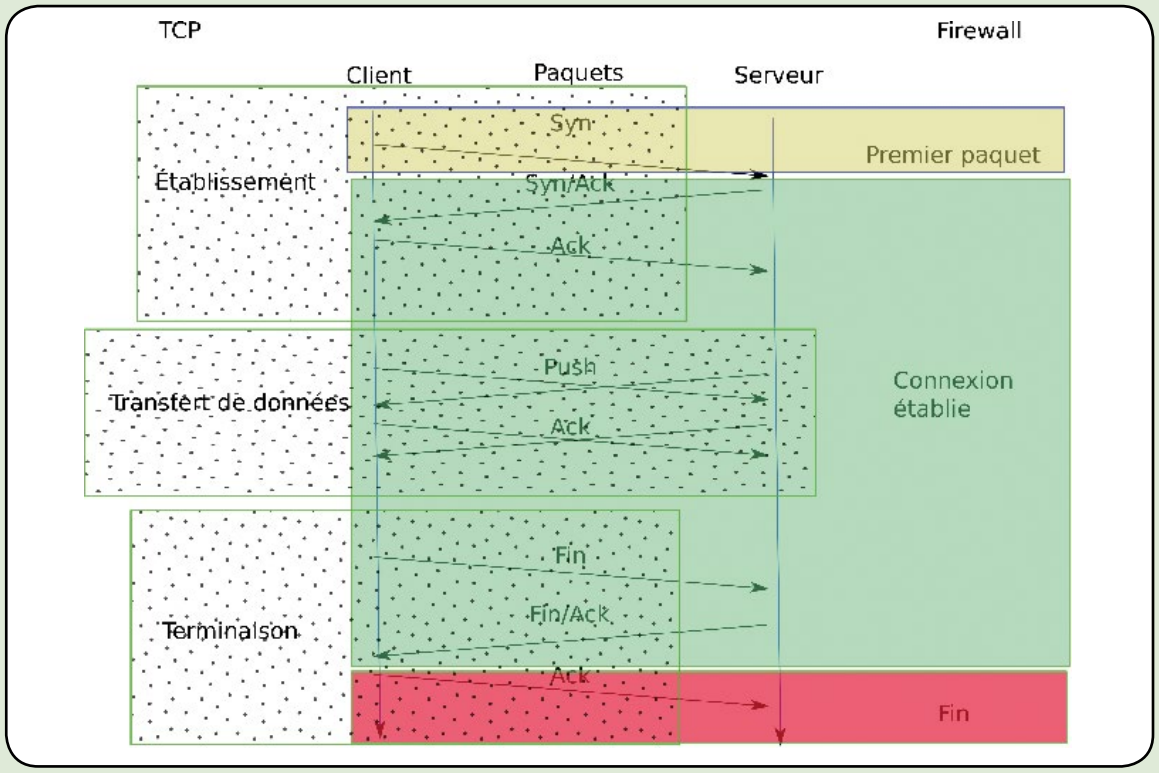


Fig. 5 : Connexion TCP.

2 Installation et premiers pas

Knock fonctionne en mode client-serveur. La machine à protéger est une machine exposée sur Internet. Il peut s'agir d'un serveur public ou d'un serveur caché. Quels que soient les services offerts par cette machine, le serveur ssh est caché et nous souhaitons pouvoir nous connecter à partir d'un poste nomade dont nous ne connaissons pas l'adresse IP a priori.

L'installation de knockd est simple. Il suffit d'installer le paquet **knockd**. Celui-ci contient les deux exécutables : **knockd**, le serveur et **knock**, le client. Nous allons commencer par le serveur. Sur une Debian Jessie, celui-ci utilise deux fichiers de configuration : **/etc/default/knockd** et **/etc/knockd.conf**. Le premier fichier à modifier est le fichier **/etc/knockd.conf**. En effet, celui-ci contient la séquence, supposée secrète. Conserver la séquence par défaut limite beaucoup la pertinence de l'installation. Une fois la séquence modifiée, il faudra autoriser le lancement de **knockd** dans le fichier **/etc/default/knockd** en positionnant la variable **START_KNOCKD** à **1**. Nous allons concentrer les explications sur le fichier **/etc/knockd.conf** qui permet de spécifier le comportement de knockd. Le fichier d'origine propose le déverrouillage du serveur ssh avec la séquence de ports **7000**, **8000** puis **9000** en moins de cinq secondes :

```
[options]
        UseSyslog

[openSSH]
        sequence    = 7000,8000,9000
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp
--dport 22 -j ACCEPT
        tcpflags    = syn

[closeSSH]
        sequence    = 9000,8000,7000
        seq_timeout = 5
        command     = /sbin/iptables -D INPUT -s %IP% -p tcp
--dport 22 -j ACCEPT
        tcpflags    = syn
```

Le fichier est organisé en sections. Chaque section commence par un titre entre crochets, et termine quand la section suivante commence ou à la fin du fichier. On y trouve :

- **sequence** : la suite de ports à écouter, dans le premier exemple, des ports TCP ;
- **seq_timeout** : la limite de temps pour dérouler toute la séquence, en secondes ;

- **command** : la commande à exécuter après réception de la séquence. En général, c'est l'ouverture d'un port ssh par la commande **iptables** ;
- **tcpflags** : les paquets TCP peuvent être de plusieurs types (début de session, fin de session, etc.). Le type est identifié par le **tcpflags**. Le serveur **knockd** peut attendre des paquets qui ne correspondent pas à une communication réelle ;
- **%IP%** : l'adresse IP apparente du client, elle sera différente si le nomade est derrière un NAT.

Dans l'exemple initial, le serveur **knockd** (ici sur la machine **william**) attend trois débuts de connexions TCP sur les ports **7000**, **8000** puis **9000**. Ces connexions peuvent être initialisées par les trois instructions shell suivantes :

```
$ nc william 7000
$ nc william 8000
$ nc william 9000
```

Ou par la commande **knock**, si le paquet **knockd** est installé sur le poste nomade :

```
$ knock william 7000 8000 9000
```

Quelle que soit la méthode choisie, le fichier **syslog** va contenir la trace de la commande :

```
Aug 30 15:36:45 william knockd: 10.33.102.3: openSSH: Stage 1
Aug 30 15:36:45 william knockd: 10.33.102.3: openSSH: Stage 2
Aug 30 15:36:45 william knockd: 10.33.102.3: openSSH: Stage 3
Aug 30 15:36:45 william knockd: 10.33.102.3: openSSH: OPEN SESAME
Aug 30 15:36:45 william knockd: openSSH: running command: /sbin/
iptables -A INPUT -s 10.33.102.3 -p tcp --dport 22 -j ACCEPT
Aug 30 15:36:45 william kernel: [12577.267801] ip_tables: (C)
2000-2006 Netfilter Core Team
```

Nous voyons la séquence d'ouverture se dérouler, puis le sésame et la commande lancée. Le port ssh est ouvert, la connexion devient possible depuis la machine ayant lancé la séquence. L'interrogation des règles du *firewall* présente la même information.

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source       destination
ACCEPT     tcp  --  averell      anywhere            tcp dpt:ssh
```

Nous avons autorisé la connexion ssh depuis notre machine nomade. Il est possible qu'un opportuniste utilise cette ouverture pour tenter de craquer le serveur ssh. Il faut pour cela qu'il utilise la même adresse IP. Ceci est possible s'il utilise le même ordinateur ou si les deux ordinateurs sont derrière le même *firewall* qui masque les adresses privées. Nous pouvons maintenant refermer le port, en utilisant l'autre séquence (closeSSH) :

```
$ knock william 9000 8000 7000
```

La règle a disparu du firewall, le syslog contient la trace de la fermeture :

```
Aug 30 15:48:20 william knockd: 10.33.102.3: closeSSH: Stage 1
Aug 30 15:48:20 william knockd: 10.33.102.3: closeSSH: Stage 2
Aug 30 15:48:20 william knockd: 10.33.102.3: closeSSH: Stage 3
Aug 30 15:48:20 william knockd: 10.33.102.3: closeSSH: OPEN SESAME
Aug 30 15:48:20 william knockd: closeSSH: running command: /sbin/
iptables -D INPUT -s 10.33.102.3 -p tcp --dport 22 -j ACCEPT
```

Lors de la phase initiale, il est plus pratique de fonctionner en ayant laissé l'accès ouvert. Ouvrir un port déjà ouvert ne présentant guère d'intérêt, Nous allons nous intéresser de plus près au *firewall*.

3 Firewall

Après ces premiers tests, il convient de bloquer l'accès initial. Par défaut, toutes les connexions sont autorisées par le *firewall*. Dès qu'on installe un *firewall*, le premier conseil consiste à tout interdire, puis autoriser ce qui doit l'être. Avec **iptables**, il faut activer la politique **DROP** :

```
# iptables -P INPUT DROP
```

Pour le debug, c'est mieux d'utiliser une règle **REJECT** (**iptables -A INPUT -j REJECT**) au lieu de la politique **DROP**. **REJECT** rejette la connexion, la notification est immédiate. **DROP** laisse tomber silencieusement les paquets. Un éventuel attaquant ne peut pas faire la différence entre un paquet « droppé » et un problème réseau. Les scans de ports consomment beaucoup plus de ressources chez l'attaquant, car chaque scan doit attendre l'expiration du délai.

Après avoir activé la politique **DROP**, si la règle **closeSSH** est activée, alors les communications en cours sont bloquées. C'est pourquoi il faut autoriser les communications en cours, c'est la règle :

```
# iptables -I INPUT -m state --state ESTABLISHED -j ACCEPT
```

ACTUELLEMENT DISPONIBLE

MISC N°88 !

WEB : QUELLES ÉVOLUTIONS POUR LA SÉCURITÉ ?

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Cette règle autorise les paquets après le paquet initial, mais pas le paquet initial. Ainsi, les communications en cours continuent, mais les nouvelles ne sont pas autorisées. En résumé :

1. Activer la politique **DROP** ;
2. Autoriser les communications établies ;
3. Autoriser ou refuser les nouvelles connexions par le serveur knockd.

Pour l'instant, la fermeture reste manuelle. Nous allons voir comment automatiser la fermeture en changeant le fichier **knockd.conf**.

4 Attention à la fermeture automatique

Il est toujours plus sûr de demander à l'ordinateur de verrouiller l'accès automatiquement. Pour cela, nous allons utiliser les commandes de début (*start*) et de fin (*stop*) séparées par un délai (**cmd_timeout**). L'étiquette de la règle devient **openclose**, il n'y a plus de règle **close**.

```
[opencloseSSH]
        sequence    = 7000,8000,9000
        seq_timeout = 5
        tcpflags    = syn
                    start_command = /sbin/iptables -A INPUT -s
%IP% -p tcp --syn -j ACCEPT
                    cmd_timeout   = 5
                    stop_command  = /sbin/iptables -D INPUT -s
%IP% -p tcp --syn -j ACCEPT
```

Après avoir reçu la séquence magique, le *firewall* ouvre l'accès ssh depuis la machine émettrice puis referme cet accès cinq secondes plus tard. Le **syslog** montre la réception de la séquence, l'ouverture et la fermeture :

```
Sep  1 11:57:19 william knockd: 10.33.102.3: opencloseSSH: Stage 1
Sep  1 11:57:19 william knockd: 10.33.102.3: opencloseSSH: Stage 2
Sep  1 11:57:19 william knockd: 10.33.102.3: opencloseSSH: Stage 3
Sep  1 11:57:19 william knockd: 10.33.102.3: opencloseSSH: OPEN
SESAME
Sep  1 11:57:19 william knockd: opencloseSSH: running command:
/sbin/iptables -A INPUT -s 10.33.102.3 -p tcp --syn -j ACCEPT
Sep  1 11:57:24 william knockd: 10.33.102.3: opencloseSSH: command
timeout
Sep  1 11:57:24 william knockd: opencloseSSH: running command:
/sbin/iptables -D INPUT -s 10.33.102.3 -p tcp --syn -j ACCEPT
```

Nous avons donc protégé notre serveur ssh en utilisant une séquence secrète. Si un malfaisant est capable d'écouter les communications entre le nomade et le serveur, alors cette séquence peut être détectée et rejouée. Knockd propose une méthode pour nous protéger du rejeu.

5 Séquences uniques

Nous allons utiliser un fichier contenant des séquences secrètes. Une par ligne. Chaque séquence ne sera valable qu'une fois. **knockd** n'offre pas de méthode pour générer un tel fichier aléatoire. Nous allons créer et utiliser le fichier **/etc/knockd/ssh_sequence** :

```
#1000:udp,1200:tcp,1400:tcp
 2000:udp,2200:tcp,2400:tcp
 3000:udp,3200:tcp,3400:tcp
```

Le caractère **#** n'est pas présent dans le fichier créé par l'utilisateur. Il est ajouté par knockd après la première ouverture. Ce fichier d'exemple est évidemment simplissime. Il ne contient que trois lignes et donc limite le nombre de connexions. Après avoir épuisé toutes les lignes, le serveur indiquera la fin dans le log et terminera (s'il n'y a qu'une seule règle).

```
knockd: no more sequences left in the one time sequences file for
door opencloseSSH --> disabling the door
```

À chaque connexion réussie, knockd modifiera le fichier en mettant la ligne en commentaire. Ainsi, même après un reboot du serveur, les séquences déjà utilisées ne pourront pas être rejouées. Ici, il y a déjà eu une connexion réussie, c'est la deuxième ligne qui doit être utilisée :

```
$ knock william  2000:udp 2200:tcp 2400:tcp
```

knock est un très bon utilitaire. Il augmente la sécurité d'un serveur ssh en masquant l'ouverture du port. Néanmoins, knockd souffre de nombreux problèmes.

6 Limitations de knockd

L'utilisation de knockd masque le serveur ssh. Les malfaisants ne pourront pas tenter les attaques de mots de passe ou les failles du serveur ssh. Avec des séquences uniques, même si la communication initiale était espionnée, le rejeu

est impossible. Néanmoins, si le nomade ne se souvient plus du numéro de séquence, il va falloir qu'il essaie de les rejouer toutes. Ceci risque de fournir à un malfaisant qui pourrait intercepter le trafic la liste des séquences à utiliser.

De plus, si plusieurs utilisateurs souhaitent utiliser le serveur ssh, la gestion des séquences devient rapidement difficile.

Enfin, si le nomade est hébergé par un administrateur ayant une vision très limitée d'Internet (il ne sait pas qu'il existe **65535** ports TCP et autant de ports UDP) il pourrait limiter dramatiquement les ports utilisables en sortie de son réseau. Limiter knockd à seulement les ports http, https, dns (par exemple) réduit considérablement ses bénéfices.

Il est possible aussi d'utiliser des drapeaux TCP différents du drapeau initial (syn). Néanmoins, cette méthode risque d'alerter les détecteurs d'intrusion qui verraient passer des paquets étranges, comme un paquet final sans le paquet initial. De même, l'utilisation fréquente de knockd avec des ports aléatoires pourrait aussi générer une alerte. Pour ces raisons, je me suis tourné vers **fwknop** qui n'a pas les mêmes inconvénients. Ceci fera l'objet d'un autre article qui est actuellement en cours de rédaction.

La grande question consiste, comme chez les médecins, à savoir si knockd augmente ou diminue la sécurité. Si un malfaisant est capable d'espionner les communications et comprend que les connexions ssh sont précédées d'une séquence de paquets pour knockd, alors il pourra la rejouer. Knockd ne masquera plus le port ssh, mais celui conservera sa sécurité propre. Même avec une séquence fixe et faible, la sécurité du serveur n'est pas diminuée et seuls quelques malfaisants pourront avoir accès à la séquence clef. Donc knockd permet bien d'augmenter la sécurité du serveur.

Conclusion

Le *port knocking* contribue à augmenter la sécurisation des serveurs exposés sur Internet, en particulier les serveurs ssh. Ils ne protègent pas la partie publique d'un serveur, comme les serveurs Internet par exemple, mais permettent de masquer la présence d'un service pour limiter les tentatives d'attaques sur ce service.

Plusieurs logiciels existent. Knockd utilise une séquence de ports qui peut être altérée par un administrateur trop zélé. Néanmoins, avec le partage de connexion de notre téléphone **Cyanogenmod**, il est possible de lancer la séquence complète sans passer par le réseau hébergeant.

La solution fwknop propose elle un paquet unique qui peut donc être envoyé sur un port qui sera ouvert, comme le port DNS ou le port HTTPS par exemple. Il reste possible que certains réseaux bloquent en sortie ce genre de communication en particulier si un détecteur d'intrusion considère que la communication ne ressemble pas à une communication classique et bloque le nomade. Fwknop est nettement plus compliqué à mettre en place que knockd, mais apporte un gain depuis des réseaux limités.

Knockd et fwknop sont deux logiciels à utiliser sans modération ! ■

Références

[1] Le site de knockd : http://www.zeroflux.org/projects/knock

[2] Page Wikipédia de TCP/IP : https://fr.wikipedia.org/wiki/Suite_des_protocoles_Internet

[3] La page de netfilter, le firewall de Linux : http://www.netfilter.org/

VIRTUALISATION AVANCÉE AVEC PROXMOX VE : VOLUMES ZFS ET CLUSTER COROSYNC

Emmanuel KASPER - [Developpeur Proxmox VE et Mainteneur Debian]

Proxmox VE intègre un grand nombre de technologies du libre, de LXC à Ceph, et de KVM à GlusterFS. Aujourd'hui, coup d'œil sur ZFS et le moteur de cluster Corosync.

Mots-clés : Proxmox, ZFS, Corosync, KVM, LXC, virtualisation

Résumé

ZFS et Corosync vous en avez déjà entendu parler comme le meilleur depuis le fil à couper le beurre ! Mais pour quoi exactement ? Coup d'œil sur leur utilisation pratique dans le gestionnaire de machines virtuelles Proxmox VE.

Proxmox VE est un gestionnaire de machines virtuelles **LXC** et **KVM**, basé sur **Debian**, qu'on peut installer directement via CD/clef USB, ou par-dessus un système Debian préexistant. Proxmox VE (dans la suite de cet article PVE) est aussi disponible préinstallé chez certains gros *hosters* français comme **OVH** ou **Online** (groupe **Illiad**)

PVE se situe à mi-chemin entre des solutions complètes, mais complexes comme **Openstack** et la gestion de machines virtuelles « à la mano » via `kvm -hda mondisk.img` ou `lxc-create -n mon_container -t debian -- -r jessie`.

PVE peut se gérer en ligne de commandes ou depuis une interface web conçue comme une *single page application*. Le tout est sous licence AGPLv3, avec une offre de support professionnelle en option.

1 ZFS pour le stockage de vos machines virtuelles

Le choix d'options de stockage proposé par PVE peut être quelque peu déroutant. Pour le néophyte, entre **LVM**, **NFS**, **ISCSI**, **GlusterFS**, ZFS, il est difficile au premier abord de savoir lequel choisir. Dans le doute, on s'en tiendra avant tout aux solutions éprouvées comme LVM et NFS, auquel il convient maintenant d'ajouter ZFS, en production depuis 2006 sur **Solaris**/**Sparc**.

Regardons d'un peu plus près maintenant ZFS, inclus depuis 2015 dans PVE et qui bénéficie d'un intérêt supplémentaire depuis l'annonce en février de son inclusion dans la dernière version LTS d'Ubuntu.

Traditionnellement sur un serveur Linux on stockerait une image de machine virtuelle (VM) avec les composants suivants : disques physiques -> raid *hardware* ou *software* -> volume physique LVM -> groupe de volumes LVM -> image disque contenue dans un volume logique LVM.

ZFS permet de réduire le nombre de composants de cinq à trois : disques physiques -> pool ZFS -> image disque contenue dans un volume ZFS (zvol).

Les *admins* chevronnés se demanderont alors : et la protection BBU du contrôleur RAID ? Et le cache des disques ? La réponse est que ZFS est prévu pour fonctionner directement au-dessus des disques durs et garantit lui-même que les données

finissent sur un support de stockage stable. À cette fin, ZFS envoie les instructions nécessaires pour contrôler le cache des disques SATA et SCSI via le jeu de commandes disque FUA.

Pour démarrer avec ZFS, on créera par exemple un *pool* de deux disques en miroir (équivalent d'un RAID 1) avec la commande :

```
# zpool create mon_pool_perso mirror ata-WDC_WD20NPVZ-00WFZT0_WD-
WXE1A95RNTND ata-WDC_WD2003FYYS-02W0B0_WD-WMAY02196811
```

On utilise la notation **/dev/disk/by-id** qui correspond à l'ID du disque tel que renvoyé par la commande **hdparm**, ou **smartcl**, ce qui permet d'associer rapidement un message d'erreur au numéro de série du disque.

Ensuite, on ajoutera via l'interface web ou en ligne de commandes le *pool* **mon_pool_perso** aux *storages* préexistants dans PVE, en prenant soin de cocher l'option ***Thin provisioning***.

Enfin, lors de la création de machines virtuelles, on sélectionnera le *storage* que nous venons de créer pour que PVE crée automatiquement un volume logique ZVOL dans le *pool* ZFS.

Avec l'option *thin provisioning* de ZFS, seuls les blocs réellement utilisés dans chaque VM sont alloués, ce qui permet de créer plusieurs machines virtuelles avec un disque de 100 GB, sur un disque physique de 128 GB.

Même si ZFS est un système de fichier local, il est possible avec les commandes **zfs send** et **zfs receive** de faire une copie incrémentielle du *pool* via le réseau.

Comme ZFS est la fois système de fichiers et gestionnaire de volumes, il sait quels blocs sont réellement utilisés, une copie du *pool* via **zfs send** est donc bien plus rapide qu'une copie de volume bit à bit avec **dd** et **ssh**.

2 Cluster Corosync pour Proxmox VE

Par défaut, l'interface web de PVE permet de gérer le serveur physique sur lequel a eu lieu l'installation initiale. Mais si vous disposez de plusieurs serveurs avec PVE installé, vous pouvez les regrouper dans un cluster et administrer toutes les machines depuis une seule page web. Un cluster PVE repose sur Corosync, le moteur de cluster standard sous Linux.

Corosync fonctionne globalement de la façon suivante : chaque nœud désirant rejoindre le cluster se connecte à une adresse multicast sur laquelle les nœuds vont s'échanger un jeton, via le protocole *totem* de Corosync. À tour de rôle chaque nœud doit passer le jeton à un voisin dans un temps

défini (une seconde par défaut). Si un nœud ne passe plus le jeton, les autres membres du cluster recréent une configuration de cluster sans le nœud défaillant.

Lors de la création du cluster et suivant le nombre de nœuds, un *quorum* est défini. Si vous venez du monde associatif, vous connaissez peut-être ce terme, le quorum désignant le nombre minimum de machines présentes pour pouvoir prendre une décision.

Si le quorum n'est pas atteint (par exemple, deux nœuds seulement présents dans l'échange de jetons sur un quorum de trois membres dans un cluster de cinq serveurs), le cluster arrête les services auxquels il est lié, ceci afin de protéger l'intégrité des données, et empêcher qu'un nœud unique modifie des paramètres globaux sans en rendre compte aux autres membres du cluster.

Dans la pratique, on préfèrera créer des clusters Corosync avec des membres en quantité impaire afin de faciliter l'obtention d'une majorité et donc la gestion du quorum.

Dans le cas de PVE, le cluster Corosync sert à partager la configuration des machines virtuelles, les statistiques d'utilisation, et certains paramètres globaux comme les droits d'accès ou le plan de sauvegarde.

Le cluster est créé sur un premier serveur avec la commande :

```
# pvecm create mon_premier_cluster
```

On ajoute alors les nœuds suivants avec :

```
# pvecm add 10.0.0.44 # adresse IP du serveur où la commande
'pvecm create' a été saisie
```

Les nœuds suivants récupèrent la configuration Corosync depuis le nœud initial, laquelle configuration est semblable à ceci pour le cas d'un cluster à trois nœuds :

```
nodelist { # membres du cluster
  node {
    name: premier_noeud # hostname du noeud
    nodeid: 2
    quorum_votes: 1 # ce nœud possède 1 vote lors de l'établissement
du quorum
    ring0_addr: premier_noeud # après résolution DNS du hostname, la
connexion multicast aura lieu sur l'adresse réseau de l'IP retournée
  }

  node {
    name: second_noeud
    nodeid: 1
    quorum_votes: 1
    ring0_addr: second_noeud
  }

  node {
    name: troisieme_noeud
    nodeid: 3
    quorum_votes: 1
    ring0_addr: troisieme_noeud
  }
}

quorum {
  provider: corosync_votequorum
}

totem {
  cluster_name: mon_premier_cluster
  config_version: 2 # incrémente lors de chaque changement du
fichier de configuration
  ip_version: ipv4
  secauth: on
  version: 2 # version du protocole totem, toujours 2
  interface {
    bindnetaddr: 10.0.0.0
    ringnumber: 0
  }

}
```

Dans le cas de PVE, Corosync sert à synchroniser une base **SQLite** dans laquelle est stockée la configuration, et le contenu de cette base de données est rendue accessible via un montage FUSE dans le répertoire **`/etc/pve`**.

Ce répertoire est partagé en lecture/écriture sur tous les nœuds membres du cluster PVE. Quand le quorum est manquant, chaque nœud remonte le répertoire **`/etc/pve`** en lecture seule, l'absence de quorum implique l'impossibilité d'une prise de décision.

2.1 Avantages d'un cluster

Une fois le cluster configuré, on peut gérer l'ensemble des serveurs, des utilisateurs et des machines virtuelles depuis une seule page web. En ajoutant un système de fichiers réseau comme un partage **NFS** sur tous les nœuds, on pourra migrer les machines d'un nœud à un autre (KVM supportant la migration *live* sans *downtime*) voire mettre en place un système de haute disponibilité.

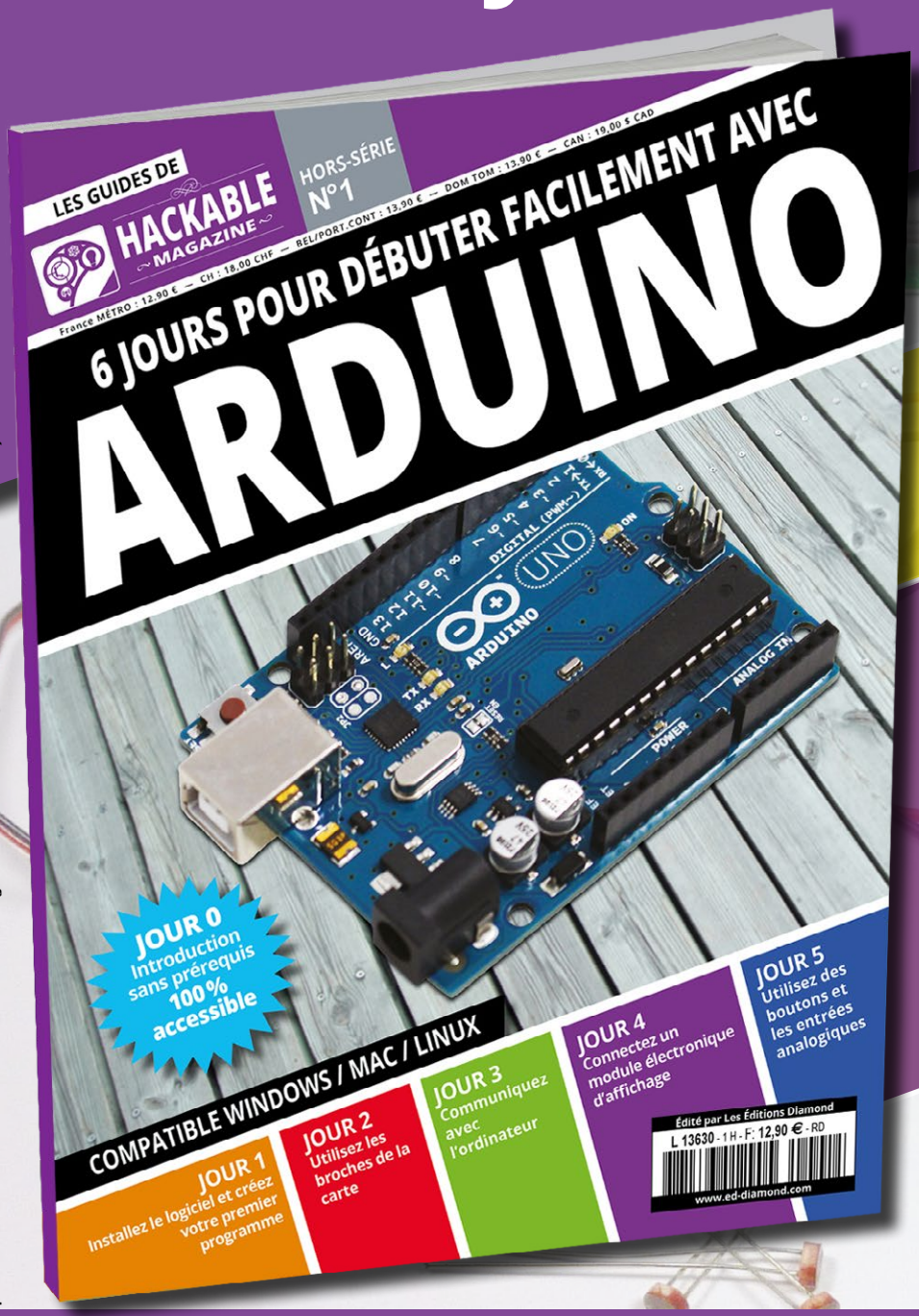
BBU

Battery Backup Unit : mémoire flash incluse dans les contrôleurs RAID de haut niveau, qui garde en mémoire le contenu de la mémoire cache du contrôleur RAID, pour pouvoir écrire ce cache après une coupure de courant.

Conclusion

ZFS semble promis à un bel avenir et son intégration dans PVE montre l'utilité de son architecture pour l'hébergement de machines virtuelles. Quant à Corosync, étant donné son ubiquité dans le monde Linux, il peut être intéressant d'y jeter un œil avant de se lancer dans la construction d'un cluster haute disponibilité. ■

IL EST DE RETOUR
CHEZ VOTRE MARCHAND DE JOURNAUX !



HACKABLE HORS-SÉRIE N°1

VOUS UTILISEZ DÉJÀ LA RASPBERRY PI ? ...OU PAS METTEZ-VOUS À L'ARDUINO !

À PARTIR DU 1ER DÉCEMBRE
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
http://www.ed-diamond.com

MOTEUR DE TEMPLATE TWIG : PRISE EN MAIN

Mariana ANDUJAR - [Ingénieur en informatique à Aix-Marseille Université]
Magali CONTENSIN - [Ingénieur en informatique au CNRS]

Le moteur de template open source Twig facilite le développement, la sécurisation et la maintenance d'applications web PHP. Il est très simple à installer et à prendre en main.

Template Twig PHP XSS HTML Programmation web

L'OBJECTIF

Cet article apprend à utiliser le moteur de *template* **Twig**, qui permet de séparer l'affichage, c'est-à-dire la partie qui génère le code HTML présenté à l'utilisateur, du code métier. L'objectif est de créer une page web présentant les livres d'une bibliothèque depuis **PHP**, en utilisant des *templates* Twig.

LES OUTILS

- un navigateur web
- un éditeur de texte
- un serveur web **Apache** avec PHP

PHASE 1

Installation

Twig est le moteur de *template* intégré dans le *Framework* **Symfony**, mais il peut être utilisé indépendamment de Symfony comme c'est le cas dans ce « How to ». La documentation de Twig est disponible à l'URL suivante : http://twig.sensiolabs.org/.

Pour exécuter le moteur de *template*, il faut un serveur web Apache avec PHP qui ne doit pas être antérieur à la version 5.2.7.

Si le gestionnaire de paquets **Composer** est déjà installé, se placer dans le répertoire de l'application web (dans cet exemple, **`/var/www/html/appli`**), puis exécuter la commande **`composer`** pour télécharger la dernière version de Twig :

```
$ cd /var/www/html/appli
$ composer require "twig/twig:~1.0"
```

Sinon, créer un répertoire **`TWIG`**, et se placer dans ce répertoire. Télécharger et installer composer, puis télécharger Twig. Enfin, placer le répertoire **`vendor`** dans le répertoire **`appli`** à la racine de l'application web, c'est-à-dire **`/var/www/html`** :

```
$ mkdir TWIG
$ cd TWIG
$ curl -s http://getcomposer.org/installer | php
$ php composer.phar require "twig/twig:~1.0"
$ mv vendor /var/www/html/appli
```

Le répertoire du projet web contient à présent un répertoire **`vendor`** comportant la bibliothèque Twig ainsi qu'un script **`autoload.php`**. La version de Twig testée dans ce document est la 1.26.1 du 5 octobre 2016.

Attention aux droits : Twig utilise un répertoire **`cache`** pour stocker les fichiers PHP générés à partir des *templates* et donc le répertoire **`/var/www/html/appli/cache`** doit être accessible en écriture pour le serveur web.

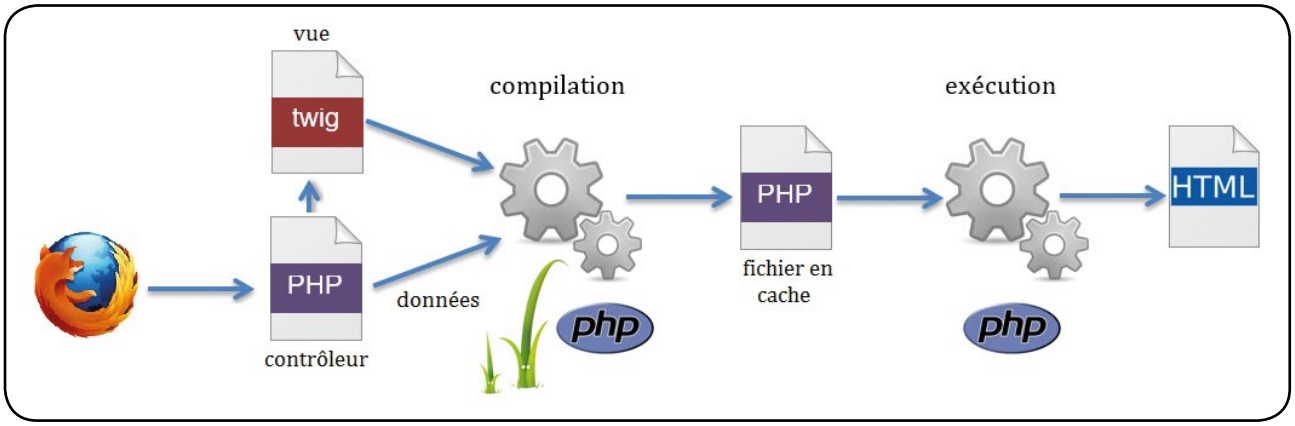


Fig. 1 : Principe de fonctionnement de Twig.

PHASE 2

Création d'un template

Le *template* est utilisé pour générer l'interface utilisateur, car séparer la présentation des traitements facilite le développement du projet et sa maintenance. La page web reçue par le navigateur est générée côté serveur à partir de deux fichiers :

- Le *template* ou gabarit (vue) est un fichier avec l'extension **twig**, qui contient des parties statiques (code HTML) ainsi que des parties dynamiques permettant de sélectionner et intégrer des données dans la page HTML. Il utilise un langage simple et concis pour parcourir les données et extraire l'information à afficher (boucles, conditions, variables, fonctions, filtres).
- Un script PHP (contrôleur) qui définit les données qui seront passées au *template*.

Le script PHP appelé par le navigateur, récupère les données et les met à disposition du *template* (figure 1). Le *template* est analysé et compilé par le moteur de *template* en un fichier PHP. Ce dernier est placé dans le répertoire de cache afin de limiter la consommation de ressources sur le serveur. Une fois ce fichier exécuté, le code HTML est généré et transmis au navigateur.

Nous allons créer un fichier **biblio.twig** qui affichera les données transmises par le script **biblio.php**, qui sera écrit dans la phase suivante. Placer le fichier **biblio.twig** dans le répertoire **vue** de l'application web. Il comportera du code HTML ainsi que des parties dynamiques marquées par deux types de délimiteurs **{{ exp }}** et **{# ... #}**. Le premier affiche le résultat de l'évaluation de l'expression **exp**, ce qui permettra d'afficher les données de manière dynamique. Le second ajoute une ligne de commentaire (voir tableau ci-dessous).

Le fichier **biblio.twig** (figure 2) contient un code HTML statique, ainsi que du code twig surligné en rouge. Les parties dynamiques du code entre délimiteur double accolades seront remplacées lors de l'exécution par les valeurs des variables **titre_doc**, **titre_page** et **date**. Il est possible d'écrire des commentaires dans les *templates* en les plaçant entre accolades et dièses. Le commentaire dans ce fichier indique l'emplacement où les titres et noms de livres seront mis en page dans les phases suivantes.

Délimiteur	Description	Exemple	Équivalent PHP
`{{ exp }}`	Afficher le résultat de l'évaluation de l'expression **exp**	`{{ x }}`	`<?php echo $x ?>` ou `<?= $x ?>`
`{# ... #}`	Commentaire twig (supprimé lors de la compilation)	`{# commentaire #}`	`/* commentaire */`
`{% ... %}`	Exécuter l'instruction (**if**, **for**, ...)	`{% if titre|length > 25 %}`	

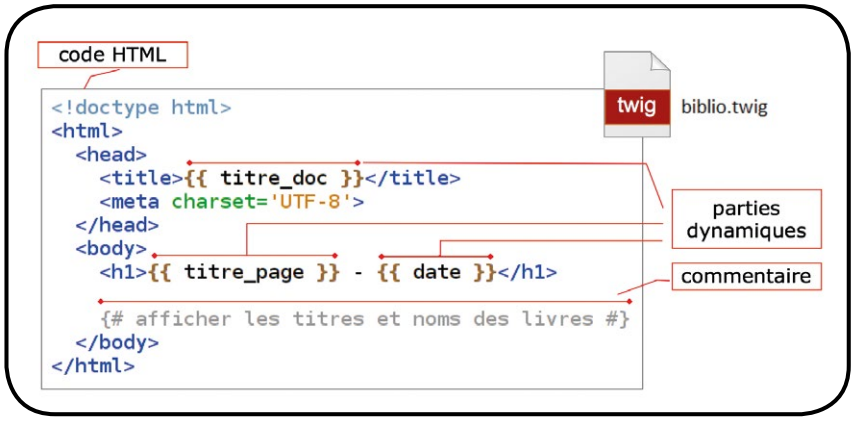


Fig. 2 : Template biblio.twig.

PHASE 3

Création du code PHP

Le fichier **biblio.php**, placé à la racine de l'application web, doit récupérer les données, initialiser Twig, stocker la configuration, charger et compiler le *template* **biblio.twig** et retourner le résultat au navigateur.

Pour simplifier l'exemple, la fonction **getData()** du script **data.php**, retourne des données statiques plutôt que d'interroger une base de données. Ce fichier, placé dans le répertoire **modele**, est inclus dans **biblio.php**. Twig travaille avec des tableaux de données. Dans le *template* Twig **{{ titre_doc }}** fait référence à la valeur de la clé **titre_doc** du tableau associatif des données.

```
<?php
// data.php
function getData(){
    // tableau associatif des données
    $data = array(
        'titre_doc' => 'Bibliotheque',
        'titre_page' => 'Liste des livres',
        'date'  => date("d/m/Y"),
        // pour simplifier l'exemple, les données sont définies
        // statiquement (généralement elles sont extraites d'une BD)
        'biblio' => array(
            array('titre'=>'N ou M', 'nom'=>'Christie', 'prenom'=>'Agatha'),
            array('titre'=>'1984', 'nom'=>'orwell', 'prenom'=>'George'),
            array('titre'=>'Dune', 'nom'=> 'Herbert', 'prenom'=>'Frank')
        )
    );
    return $data;
}
```

Une fois les données récupérées et stockées dans la variable **$donnees** dans **biblio.php**, le script inclut l'autoloader, puis définit le mode de chargement du *template* et son emplacement, en créant une instance de la classe **Twig_Loader_Filesystem**. Le *template* sera chargé depuis le chemin passé en paramètre, c'est-à-dire le répertoire **vue** placé au même niveau de l'arborescence que **biblio.php**.

Nous avons défini deux tableaux d'options pour les phases de développement et de production. Pendant le développement, le système de cache est désactivé en attribuant la valeur **false** à l'option **cache**. Lorsque l'application sera en production, Twig stockera dans le répertoire **cache** les fichiers PHP générés. L'option **autoescape** permet de protéger automatiquement les sorties contre les attaques de type XSS. D'autres options sont disponibles, elles sont décrites dans le tableau d'options ci-après.

Le script **biblio.php** crée une instance de la classe **Twig_Environment**, puis il charge et compile le *template* **biblio.twig** avec les données du tableau **$donnees**. Enfin, l'instruction **echo** retourne la page HTML au navigateur.

```
<?php
// biblio.php
/* récupérer le tableau des données */
require 'modele/data.php';
$donnees = getData();

/* inclure l'autoloader */
require_once 'vendor/autoload.php';
/* templates chargés à partir du système de
fichiers (répertoire vue) */
$loader = new Twig_Loader_
Filesystem('vue');
/* options : prod = cache dans le
répertoire cache, dev = pas de cache */
$options_prod = array('cache' => 'cache',
'autoescape' => true);
$options_dev = array('cache' => false,
'autoescape' => true);
/* stocker la configuration */
$twig = new Twig_Environment($loader,
$options_dev);
/* charger+compiler le template,
exécuter, envoyer le résultat au
navigateur */
echo $twig->render('biblio.twig',
$donnees);
```

PHASE 4

Exécution

Pour tester l'exemple, il suffit d'utiliser l'URL suivante dans le navigateur : http://localhost/appli/biblio.php.

Option	Valeur par défaut	Description
`charset`	`utf-8`	Jeu de caractères des *templates*
`cache`	`false`	Chemin du répertoire de cache
`strict_variables`	`false`	`false` = ignorer les variables ou fonctions inexistantes (remplacé par `null`) ; `true` = lancer une exception
`autoescape`	`true`	`true` = protection automatique des sorties (la stratégie de protection peut être définie : css, url, htm_attr) ; `false` = pas de protection automatique

Lorsque le navigateur demande le script **biblio.php**, les données et la vue **biblio.twig** sont compilées dans un fichier PHP (figure 3) ; c'est ce fichier qui est exécuté et qui génère le code HTML présentant les données.

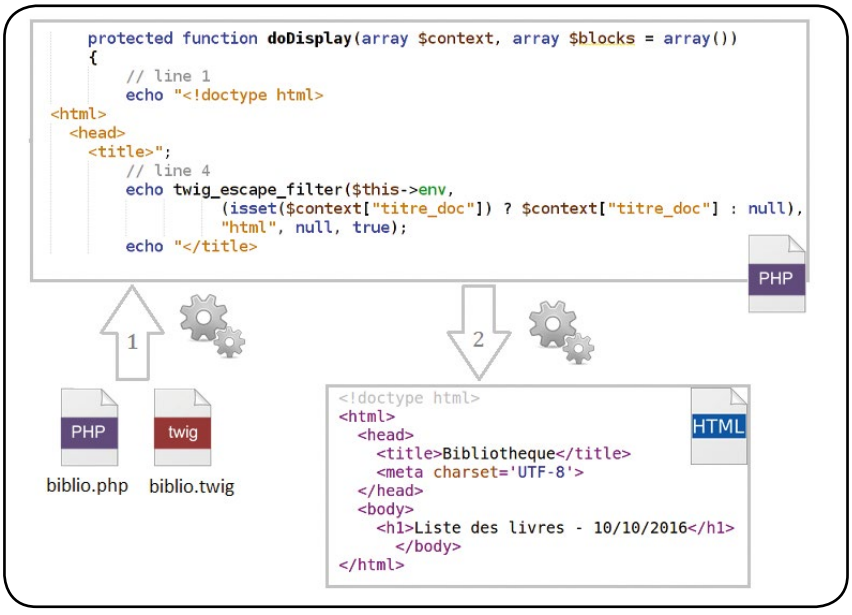


Fig. 3 : Processus d'exécution.

Le code dynamique du *template* a été remplacé par le mot **Bibliotheque** dans le titre du document (onglet dans la figure 4). Le titre de niveau un (**h1**) contient un titre suivi de la date d'exécution du script. Le résultat HTML ne comporte pas encore de liste de livres, car elle sera ajoutée dans une phase ultérieure. Si vous avez fait une erreur de syntaxe dans le *template*, vous obtiendrez une erreur **500** lors de l'exécution, vérifiez les logs d'erreur du serveur Apache pour déterminer la cause du problème.

Fig. 4 : Résultat de l'exécution de biblio.php.

PHASE 5

Ajout d'une condition

Nous allons modifier le *template* **biblio.twig**, pour afficher un message si la variable **biblio** ne contient pas de données. Pour cela, il faut utiliser les structures conditionnelles **if** et **else** entre les délimiteurs d'exécution d'instruction **{% ... %}**.

```
<!doctype html>
<html>
  <head>
    <title>{{ titre_doc }}</title>
    <meta charset='UTF-8'>
  </head>
  <body>
    <h1>{{ titre_page }} - {{ date }}
</h1>
    {% if biblio is not empty %}
  {# afficher les titres et noms des
livres #}
    {% else %}
      Aucun livre dans la bibliothèque.
    {% endif %}
  </body>
</html>
```

Les expressions dans les conditions utilisent des opérateurs et des tests définis dans les tableaux ci-après. L'expression du listing combine l'opérateur **is not** et le test **empty**. Si la variable **biblio** n'est pas vide, les titres et noms de livres seront affichés dans la phase suivante, sinon un message indique qu'il n'y a pas de livre dans la bibliothèque (figure 5).

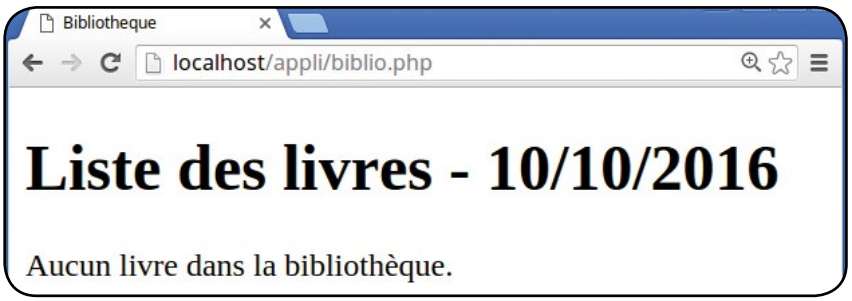


Fig. 5 : Résultat lorsqu'il n'y a pas de données.

Opérateurs	Types d'opérateurs
`+ - * / %`	Mathématiques
`and or not`	Logiques
`== != >= <= > <`	Comparaison
`in`	Booléen (ex : `a in b` est vrai si `a` est dans `b`)
`is is not`	Test
`..`	Création de séquence
`|`	Filtre
`~`	Concaténation
`? :`	Ternaire

Tests	Description
`defined`	Vrai si la variable est définie dans le contexte courant
`empty`	Vrai si la variable est vide (`null`, `false`, `''`, tableau vide)
`null`	Vrai si la variable est à `null`
`even / odd`	Vrai si la variable est paire/impaire
`iterable`	Vrai s'il est possible d'itérer sur la variable (tableau, objet traversable)

Fig. 6 : Parcours du tableau.

PHASE 6

Parcours d'un tableau

Dans cette phase, nous allons parcourir les données du tableau **biblio**, afin d'afficher les titres et les noms d'auteurs. La boucle **for** parcourt chaque item du tableau **biblio**. L'item étant lui-même un tableau, pour accéder au nom de l'auteur du livre, il faudra utiliser le nom du tableau suivi du caractère point et de la clé, c'est-à-dire **item.nom** (figure 6). La figure 7 montre le résultat dans le navigateur. Les filtres utilisés dans le listing et leurs effets sont présentés dans la dernière phase.

```
<!doctype html>
<html>
  <head>
    <title>{{ titre_doc }}</title>
    <meta charset='UTF-8'>
  </head>
  <body>
    <h1>{{ titre_page }} - {{ date }}
</h1>
    {% if biblio is not empty %}
    <table>
     <thead>
      <tr><th>Titre</th><th>Auteur
</th></tr>
     </thead>
     <tbody>
      {# afficher les titres et noms des
livres #}
      {% for item in biblio|sort %}
       <tr>
        <td>{{ item.titre }}</td>
         <td>{{ item.nom|capitalize }}
</td>
       </tr>
      {% endfor %}
     </tbody>
    </table>
    {% else  %}
       Aucun livre dans la
bibliothèque.
    {% endif %}
  </body>
</html>
```

PHASE 7

Ajout de filtres

Twig propose une trentaine de filtres qui peuvent être appliqués à une variable ou à une section de code. Il est possible de chaîner les filtres. Une partie des filtres est listée dans le tableau ci-dessous. Dans le listing, nous utilisons `capitalize` pour mettre en majuscule la première lettre du nom de l'auteur, ainsi que le filtre `sort` pour trier le tableau de données avant son affichage.

LE RÉSULTAT

La figure 7 montre le résultat obtenu une fois toutes les phases réalisées. La page web affiche la liste des livres définie dans le tableau de données (fichier `data.php`). Les variables du *template* `biblio.twig` ont été remplacées dans les éléments `title` et `h1`. Les lignes du tableau HTML ont été générées dynamiquement par la boucle `for`. La liste des livres n'est pas affichée dans l'ordre de déclaration du tableau de données, car un filtre de tri a été appliqué dans le *template*. Le filtre `capitalize` a permis d'uniformiser l'affichage des noms en changeant le premier caractère en majuscule, c'est ce qui explique la majuscule à Orwell qui n'en avait pas dans le tableau de données. Si le fichier `data.php` ne contient pas de clé `biblio` ou que cette clé contient un tableau vide, alors un message est affiché (figure 5). ■

Fig. 7 : Résultat présentant les données.

Filtre	Type	Rôle du filtre
`number_format`	Nombre	Retourne le nombre formaté, argument 1 = nb de décimales, argument 2 = séparateur décimal ex : `{{ nb|number_format(2, ',') }}` format avec 2 chiffres après la virgule
`round`	Nombre	Retourne le nombre arrondi, argument 1 = précision (défaut `0`), argument 2 = type d'arrondi (`ceil`, `floor` ou `common` qui choisit l'arrondi supérieur ou inférieur en fonction de la valeur) ex : `{{ nb|round }}` ou `{{ nb|round(2,'ceil') }}`
`capitalize`	Chaîne	Retourne la chaîne avec le premier caractère en majuscule, les autres en minuscules ex : `{{ titre|capitalize}}`
`lower`	Chaîne	Retourne la chaîne en minuscules
`upper`	Chaîne	Retourne la chaîne en majuscules
`length`	Chaîne, tableau	Retourne la taille de la chaîne ou du tableau
`keys`	Tableau	Retourne les clés d'un tableau
`join`	Tableau	Retourne la concaténation des valeurs des cases d'un tableau, argument 1 = délimiteur ex : `nom_tab|join` ou `nom_tab|join(',')`
`sort`	Tableau	Tri de tableau. ex : `{% for item in tab|sort %} ... {% endfor %}`
`date`	Date	Formate la date. ex : `{{ livre.parution|date("d/m/Y") }}`

ÉTENDEZ PANDOC AVEC LUA

Cyril ROELANDT - [Développeur]

Il est parfois nécessaire de convertir un fichier d'un langage de balisage vers un autre : de Markdown vers du HTML, d'Org-mode vers LaTeX, etc. Pandoc est un outil permettant ce type de conversion, et il est capable de gérer un nombre impressionnant de formats différents. Que faire si l'on souhaite utiliser un format ésotérique inconnu de Pandoc ? Il est possible de l'étendre en Lua !

org-mode extension **Pandoc** lua

L'OBJECTIF

L'architecture de **Pandoc** peut être comparée à celle d'un compilateur : un *lecteur* lit le fichier d'origine et crée une représentation intermédiaire ; cette représentation est ensuite convertie vers le format d'arrivée par un *écrivain*.

Implémenter le support complet d'un nouveau format dans Pandoc consiste donc simplement à écrire un lecteur et un écrivain. Cela semble toutefois fastidieux si le format à implémenter et à maintenir est destiné à rester relativement confidentiel, cantonné à un usage personnel. Il est alors beaucoup plus judicieux d'étendre Pandoc en utilisant le mécanisme prévu à cet effet.

Dans cet article, nous montrerons comment convertir une recette de cuisine écrite en Org-mode en un fichier **LaTeX** utilisant la classe **`recipe`**, et ce afin de générer un joli PDF. Le code complet est disponible à l'adresse suivante : https://framagit.org/Steap/GLMF-articles/tree/master/pandoc-lua-extension.

LES OUTILS

- pandoc (1.17.0.3 dans cet article)
- pdflatex (paquet **`texlive-latex-base`** dans Debian)
- la classe LaTeX **`recipe`** (paquet **`texlive-latex-extra`** dans Debian)
- Lua n'est **pas** nécessaire
- un éditeur de texte

Pourquoi une extension ?

Nous voulons générer notre recette de cuisine depuis le fichier **`pizza.org`** écrit dans le format explicite suivant :

```
# -*- mode: org -*-
#+TITLE: Pizza/bière façon geek
#+AUTHOR: Cyril Roelandt
#+OPTIONS: toc:nil
#+PERSONNES: 6

* Ingrédients
+ 42 bières
+ 1 téléphone

* Préparation
+ Ouvrir une bière (par personne)
+ Se saisir de son téléphone (un seul
pour tout le groupe)
+ Appeler son livreur de pizza préféré
+ Commander poliment ses pizzas
+ Attendre le livreur en buvant une autre
bière (par personne)
+ Ouvrir au livreur, échanger les pizzas
contre des bitcoins
+ Déguster les pizzas en finissant les
bières
```

Voici le fichier **`pizza.tex`** que nous souhaiterions obtenir :

```
\documentclass[a4paper]{recipe}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[french]{babel}
\usepackage{enumitem}
\usepackage{amssymb}

\pagenumbering{gobble} % Disable page
numbering

\newcommand{\bsi}[2]{%
  \fontencoding{T1}\fontfamily{pbs}\
fontseries{xl}\fontshape{n}%
  \fontsize{#1}{#2}\selectfont}
```

```
\renewcommand{\inghead}{\textbf{Ingrédients
(pour 6 personnes)}:\ }
\renewcommand{\rechead}{\centering\
bsi{24pt}{30pt}}

\makeatletter
\renewcommand*\l@subsubsection{\@
dottedtocline{3}{3em}{0em}}
\makeatother

\setlength\parskip{2ex plus 0.5ex}

\begin{document}
\recipe{Pizza/bière façon geek}
\ingred{42 bières ; 1 téléphone ; }
\begin{enumerate}[label=$\
blacktriangleright$]
\item Ouvrir une bière (par personne)
\item Se saisir de son téléphone (un seul
pour tout le groupe)
\item Appeler son livreur de pizza
préféré
\item Commander poliment ses pizzas
\item Attendre le livreur en buvant une
autre bière (par personne)
\item Ouvrir au livreur, échanger les
pizzas contre des bitcoins
\item Déguster les pizzas en finissant
les bières
\end{enumerate}
\begin{flushright}
Une recette de Cyril Roelandt.
\end{flushright}
\end{document}
```

Il existe déjà dans pandoc un *lecteur* capable de transformer le fichier Org-mode en une représentation interne. Pandoc fournit également un *écrivain* capable de générer du LaTeX, mais le code généré n'utilise évidemment pas la classe **recipe** : nous n'aurons donc pas le rendu voulu. Intégrer et maintenir un nouvel écrivain dans pandoc serait pénible et sans doute inutile pour la plupart des utilisateurs; il est donc tout à fait logique de préférer écrire une extension.

PHASE 1

Vue d'ensemble

Notre extension nécessite deux fichiers :

- un **modèle** (**recipe.template**) qui contiendra la structure de notre document LaTeX à générer ;
- un fichier Lua (**recipe.lua**) qui convertira le contenu de notre fichier Org-mode.

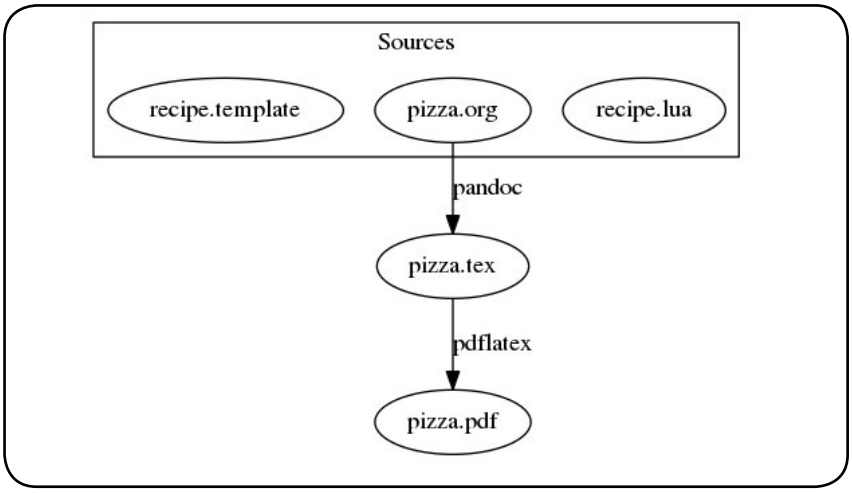


Fig. 1

Comme on peut le voir sur la figure 1, pandoc, grâce à notre extension, convertira notre fichier source **pizza.org** en un fichier LaTeX structuré comme nous le souhaitons (**pizza.tex**) que nous pourrons ensuite convertir en PDF grâce à **pdflatex**.

PHASE 2

Écriture du modèle

Voici le modèle (**recipe.template**) que nous allons utiliser :

```
\documentclass[a4paper]{recipe}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[french]{babel}
\usepackage{enumitem}
\usepackage{amssymb}

% D'autres commandes LaTeX vues plus haut,
% non reprises ici par souci de lisibilité.

\begin{document}
\recipe{$title$}
$body$
$if(author)$
\begin{flushright}
Une recette de $author$.
\end{flushright}
$endif$
\end{document}
```

Il est facilement compréhensible : il contient la structure de notre fichier LaTeX. La partie réellement intéressante est l'utilisation des variables **personnes**, **title** et **author**, qui correspondent aux pragmas utilisés au début de **pizza.org**. Il est possible de spécifier autant de variables que nécessaire. La valeur d'une variable peut même être définie dynamiquement en lançant pandoc avec l'option **-V** :

```
$ pandoc -Vfoo=bar ...
```

PHASE 3

Code minimal de l'extension

La commande suivante nous permet de générer le code Lua d'une extension implémentant un écrivain HTML :

```
pandoc --print-default-data-file sample.
lua > recipe.lua
```

Le fichier **recipe.lua** ainsi généré contient un grand nombre de fonctions Lua permettant de traiter chaque type de nœud (titre, texte, gras, etc.). Chacune de ces fonctions doit retourner une chaîne de caractères.

Nous conservons une implémentation pour les fonctions suivantes :

```
function Doc(body, metadata, variables)
  return body
end

function Str(s)
  return s
end

function Space()
  return " "
end
```

La première nous permettra de retourner le contenu du document (cf. la variable **body** du modèle). Les deux autres nous permettront respectivement de retourner les chaînes de caractères inchangées, et de conserver les espaces.

Nous pouvons « vider » sans risque le contenu des autres fonctions en leur faisant retourner une chaîne de caractères vide ; nous pourrions tout simplement les supprimer, mais il est pratique de les conserver s'il devient nécessaire de les implémenter par la suite :

```
-- Le même corps pour toutes les autres
fonctions.
function Span(s)
  return ""
end
```

PHASE 4
Makefile

Nous devrions désormais être capables de générer un fichier PDF depuis notre fichier source. Utilisons le **Makefile** suivant :

```
%.pdf: %.tex
  pdflatex $<

%.tex: %.org recipe.lua recipe.template
  pandoc -f org -t recipe.lua
--template=recipe.template -o $@ $<

clean:
  rm -f *.log *.aux *.pdf *.tex
```

Il est notamment possible de déverminer notre extension en lisant le fichier **pizza.tex** produit. Générons maintenant notre PDF :

```
$ make pizza.pdf
```

Le fichier produit est valide, mais un peu vide. C'est normal : il nous reste quelques fonctions à réellement implémenter.

PHASE 5
Le code de l'extension

Notre fichier **pizza.org** est composé de deux listes : une liste d'ingrédients, et une liste d'opérations à effectuer. En regardant le fichier **pizza.tex** présenté en début d'article, on voit qu'il faudra convertir la première en une commande **\ingred**, et la deuxième en une liste LaTeX classique. Implémentons donc la fonction Lua **BulletList** dans **recipe.lua** :

```
function BulletList(items)
  if current_section == 'Ingrédients' then
    ret = "\\ingred{"
    for _, item in pairs(items) do
      ret = ret .. item .. " ; "
    end
    ret = ret .. "}\n"
  elseif current_section == 'Préparation'
then
    ret = "\\begin{enumerate}[label=$\\
blacktriangleright$]\n"
    for _, item in pairs(items) do
      ret = ret .. "\\item " .. item .. "\n"
    end
    ret = ret .. "\\end{enumerate}"
  end
  return ret
end
```

Nous retenons dans quelle section nous nous trouvons grâce à la variable **current_section** déclarée en début de fichier et mise à jour à chaque fois que nous rencontrons un titre :

```
local current_section = ""
...
function Header(lev, s, attr)
  current_section = s
  return ""
end
```

Trucs et astuces

Quelle que soit l'extension, la méthodologie reste la même que celle présentée ici :

1. Déterminer la « chaîne de compilation » : suffit-il de lancer pandoc, ou d'autres étapes (comme le lancement d'une commande externe) seront-elles nécessaires ?
2. Construire l'extension minimale permettant de lancer toute cette chaîne sans erreur, même si le résultat ne correspond pas à nos attentes.
3. Implémenter les fonctions Lua nécessaires une par une jusqu'à avoir un résultat satisfaisant.

Il est également utile de pouvoir visualiser l'arbre syntaxique généré par le lecteur afin de déverminer une extension. Pandoc fournit cette fonctionnalité et peut afficher l'arbre en JSON :

```
$ pandoc -f org -t json pizza.org |
json_pp
```

Conclusion

Nous avons vu comment développer une extension Lua implémentant un écrivain simple pour pandoc. La méthode reste la même pour des écrivains plus complexes : cet article a été rédigé en Org-mode puis converti vers le format ODT grâce à une extension pandoc au code un peu plus volumineux. ■

PYTHON ET LE CAS DU SWITCH (OU THE SWITCH CASE EN ANGLAIS)

Tristan COLOMBO

Un développeur débarquant du C (ou de la plupart des langages de programmation) en Python aura toujours la même expression horrifiée en découvrant l'absence de la structure switch/case. Mais pourquoi diable cette structure est-elle absente de Python ?

Mots-clés : Python, switch, case, conditions, variable

Résumé

Les traitements de type `switch/case` posent beaucoup de questions à nombre de développeurs Python. Dans cet article, nous étudions les différentes façons d'obtenir un traitement similaire à cette structure manquante.

La structure **`switch`**/**`case`** est une structure particulièrement utile pour traiter de nombreuses conditions pouvant apparaître sur la valeur d'une variable. Ainsi, pour une variable **`choice`** nous pouvons définir en **C** un certain nombre de traitements de la manière suivante :

```
switch (choice)
{
    case 0:  // traitement
             break;
    case 1:  // traitement
             break;
    case 2:  // traitement
    case 3:  // traitement
             break;
    default: // traitement
}
```

La valeur de **`choice`** présentée ici est un entier, mais il pourrait s'agir bien entendu de n'importe quel autre type. Les traitements pour chaque cas sont exécutés jusqu'à rencontrer une instruction **`break`**. Cela signifie que dans le cas où **`choice`** vaut **`2`**, les blocs de traitement correspondant aux valeurs **`2`** et **`3`** seront exécutés. Pour **`choice`** valant **`3`**, seul le bloc correspondant à la valeur **`3`** sera exécuté. Enfin, pour les cas où **`choice`** possède une valeur différente de **`0`**, **`1`**, **`2`** ou **`3`**, c'est le traitement correspondant au cas **`default`** qui sera exécuté.

Voyons maintenant comment obtenir une structure similaire en Python.

1 La méthode officielle

Les demandes d'ajout d'une structure **`switch`**/**`case`** ont toujours été rejetées (voir PEP 275 **[1]** et PEP 3103 **[2]**). La réponse officielle, présente dans la documentation **[3]** propose trois alternatives :

1. Utiliser des **`if`**/**`elif`** en cascade et achever le traitement par un **`else`**. Le traitement précédent pourrait alors être écrit de la manière suivante :

```
if choice == 0:
    # traitement
elif choice == 1:
    # traitement
elif choice == 2:
    # traitement
elif choice == 3:
    # traitement
else:
    # traitement
```

On voit immédiatement qu'il manque quelque chose : nous n'avons pas intégré le fait que si **choice** vaut **2**, il faut exécuter les blocs de code correspondant à **2** et à **3** ! La solution sera de « briser » la cascade de **if**/**elif** :

```
if choice == 0:
    # traitement
elif choice == 1:
    # traitement
elif choice == 2:
    # traitement
if choice == 2 or choice == 3:
    # traitement
else:
    # traitement
```

Cette réponse est donc satisfaisante pour de petits traitements où la variable est testée sur trois ou quatre valeurs avec un traitement unique pour chaque valeur. Dans les autres cas, le code n'est plus suffisamment lisible.

2. Pour les cas présentant de nombreuses options de choix, la FAQ préconise l'emploi de dictionnaires en utilisant la structure suivante :

```
def traitement_0():
    # code

...

def traitement_3():
    # code

functions = {0: traitement_0,
             1: traitement_1,
             2: traitement_2,
             3: traitement_3}

func = functions[choice]
func()
```

Ici, en dehors de la lisibilité qui peut encode se discuter, on peut constater l'absence de traitement par défaut ainsi que le fait que la valeur **2** ne lance que **traitement_2** alors qu'elle devrait également exécuter **traitement_3**. Voici une proposition d'implémentation permettant d'ajouter ces deux contraintes :

```
01: def traitement_0():
02:     print('0')
03:
04: def traitement_1():
05:     print('1')
06:
07: def traitement_2():
08:     print('2')
09:     traitement_3()
10:
11: def traitement_3():
12:     print('3')
13:
14: def traitement_default():
15:     print('Default')
16:
17: functions = {0: traitement_0,
18:              1: traitement_1,
19:              2: traitement_2,
20:              3: traitement_3,
21:              'default': traitement_default}
22:
23: choice = int(input('Votre choix : '))
24: func = functions[choice] if choice in functions.keys() else
functions['default']
25: func()
```

La lisibilité du code n'est vraiment pas améliorée ! J'ai introduit un appel à **traitement_3()** dans **traitement_2()** (ligne 9) de manière à ce que les deux fonctions soient exécutées dans le cas où **choice** vaut **2** (ligne 19). Il faut souhaiter n'avoir jamais à appeler uniquement le code du **traitement_2()** précédent dans le reste du programme. Sinon, il faudrait alourdir encore le code :

```
...
07: def traitement_2():
08:     print('2')
09:
10: def traitement_2_switch():
11:     traitement_2()
12:     traitement_3()
```

J'ai ajouté le cas d'un traitement par défaut avec la valeur **'default'** en ligne 21. La partie « critique » du traitement se trouve en ligne 24 : si la valeur contenue dans **choice** fait partie des clés du dictionnaire **functions** (**if choice in functions.keys()**) alors **func** prend pour valeur le pointeur vers la fonction **functions[choice]**. Sinon, **func** aura pour valeur **functions['default']**. Bien entendu, si l'utilisateur saisit **default** à la question de la ligne 23, le branchement s'effectuera sur **functions['default']**, mais cela n'a rien de gênant puisque **default** ne fait pas partie de la liste des valeurs autorisées.

Note

La ligne 25 peut également s'écrire de la manière suivante :

```
func = functions.get(choice, functions['default'])
```

La méthode **get()** appliquée à un dictionnaire renvoie la valeur associée à la clé passée en premier paramètre et, si cette clé n'existe pas, elle retourne la valeur du second paramètre.

3. Dans une architecture orientée objet, la FAQ préconise l'emploi d'un *dispatcher* :

```
01: class Switch:
02:     def visit_0(self):
03:         print('0')
04:
05:     def visit_1(self):
06:         print('1')
07:
08:     def visit_2(self):
09:         print('2')
10:
11:     def visit_3(self):
12:         print('3')
13:
14:     def dispatch(self, value):
15:         method_name = 'visit_' + str(value)
16:         method = getattr(self, method_name)
17:         method()
18:
19: if __name__ == '__main__':
20:     switch = Switch()
21:     choice = int(input('Votre choix : '))
22:     switch.dispatch(choice)
```

Je ne vois pas pourquoi intégrer dans une architecture orientée objet une structure fondamentale... Ici on sort vraiment « l'artillerie lourde » pour pas grand-chose. Il faut créer une classe (ici **Switch**) contenant une méthode pour chaque traitement (**visit_0()** lignes 2 et 3, **visit_1()** lignes 5 et 6, **visit_2()** lignes 8 et 9, et **visit_3()** lignes 11 et 12). La méthode **dispatch()** des lignes 14 à 17 permet de créer le nom de la méthode à appeler en fonction du paramètre **value** (ligne 15), de créer un pointeur sur cette dernière (ligne 16) et enfin de l'exécuter (ligne 17). Malgré une architecture complexe (pour l'objectif fixé), là encore on constate l'absence de traitement par défaut et d'exécution de blocs consécutifs simulant l'absence d'un **break**. Voici une implémentation possible corrigeant ces oublis :

```
01: class Switch:
02:     def visit_0(self):
03:         print('0')
04:
05:     def visit_1(self):
06:         print('1')
07:
08:     def visit_2(self):
09:         print('2')
10:         self.visit_3()
11:
12:     def visit_3(self):
13:         print('3')
14:
15:     def visit_default(self):
16:         print('Default')
17:
18:     def dispatch(self, value):
19:         method_name = 'visit_' + str(value)
20:         try:
21:             method = getattr(self, method_name)
22:         except AttributeError:
23:             method_name = 'visit_default'
24:             method = getattr(self, method_name)
25:         method()
26:
27: if __name__ == '__main__':
28:     switch = Switch()
29:     choice = int(input('Votre choix : '))
30:     switch.dispatch(choice)
```

La solution retenue pour simuler la présence ou l'absence de **break** est la même que précédemment en ajoutant un appel à **visit(3)** dans **visit_2()** (ligne 10). La gestion du cas par défaut se fait en interceptant l'exception **AttributeError** qui est levée si le nom de la méthode spécifiée dans **method_name** lors de l'appel à **getattr()** n'est pas valide (ligne 21). Encore une fois la solution n'est pas vraiment satisfaisante.

2 Une bidouille avec les exceptions

On peut trouver sur Internet de nombreuses tentatives de reproduction de structure **switch/case** en Python. Généralement ces « bidouilles » (restons sérieux, personne n'oserait utiliser cela dans un vrai programme) sont basées sur un usage de **for** (qui dispose déjà du **break**) ou des exceptions. Nous n'allons pas analyser toutes les solutions proposées, mais en étudier seulement une pour illustrer les problèmes qui apparaissent avec ce genre de solutions :

```
01: import sys
02:
03: class Case_Select(Exception):
04:     def __init__(self, value):
05:         Exception.__init__(self, value)
06:
07: def switch(variable):
08:     raise Case_Select(variable)
09:
10: def case(*values):
11:    exc_class, exc_obj, _ = sys.exc_info()
12:    if exc_class is Case_Select and exc_obj.args[0] in values:
13:       return exc_class
```

```
14:     return Warning
15:
16: if __name__ == '__main__':
17:     choice = int(input('Votre choix : '))
18:     try:
19:         switch(choice)
20:     except case(0):
21:         print('0')
22:     except case(1):
23:         print('1')
24:     except case(2):
25:         print('2')
26:     except case(3, 4):
27:         print('3 ou 4')
28:     except case(5):
29:         print('5')
30:     except:
31:         print('Default')
```

Ici il faut imaginer les lignes 1 à 15 dans un autre fichier qui serait importé lors de l'utilisation du **switch**/**case**. On commence par définir une classe d'exception (donc héritant de **Exception**) acceptant un paramètre (ce sera notre valeur de test). Cette définition a lieu dans les lignes 3 à 5. Nous définissons ensuite une fonction **switch()** qui lève une exception **Case_Select** créée précédemment et une fonction **case()** qui sera chargée de maquiller le nom de l'exception : si l'exception est bien de type **Case_Select** et que la valeur transmise est correcte alors on retourne l'exception, sinon on renvoie une exception de basse priorité (**Warning** en l'occurrence) pour que le cas soit ignoré (lignes 12 à 14). Les tests sont ensuite basés sur une analyse de **switch(choice)** (lignes 18 et 19) et un traitement des exceptions levées (lignes 20 à 31). Vous constaterez qu'ici encore l'exécution d'un bloc et d'un bloc suivant (absence de **break**) n'est pas traitée et qu'il faudrait, à nouveau, réaliser des appels de fonctions.

D'un point de vue structurel cette solution paraît plus claire, mais elle n'est, elle non plus, pas du tout satisfaisante (détournement du mécanisme des exceptions et impossibilité de simuler simplement le fonctionnement du **break**). C'est cet aspect de détournement d'un mécanisme qui n'est pas prévu pour cela qui en fait une bidouille, comme les propositions basées sur **for** (contrairement aux techniques basées sur **if**/**elif** et les dictionnaires).

Conclusion

Alors que dire du cas du **switch** en Python ? Comme vous avez pu le constater, aucune solution n'est réellement satisfaisante lorsque l'on cherche à obtenir exactement le même fonctionnement qu'un **switch**/**case**. Même Guido van Rossum, à l'origine du PEP 3103 **[2]**, après un sondage lors de PyCon 2007 a constaté qu'une majorité de développeurs (i.e. des développeurs Python présents à PyCon 2007...) n'avaient pas besoin du **switch**/**case**. Nous en revenons donc à la préconisation initiale de la FAQ : utiliser des **if**/**elif** ou des dictionnaires (pour la version orientée objet il faudrait être un peu maso). Et, finalement, en changeant d'optique dans l'écriture du code, il est vrai que ça ne fonctionne pas si mal... même si je fais partie des développeurs Python absents du PyCon 2007 et qui auraient bien aimé un **switch**/**case**, juste pour clarifier certains bouts de codes. Mais soyons francs, même sans **switch**/**case** Python reste le meill^W^W l'un des meill^W^W^W un bon langage ! ■

Références

[1] PEP 275 - *Switching on multiples values* : https://www.python.org/dev/peps/pep-0275/

[2] PEP 3103 - *A switch/Case statement* : https://www.python.org/dev/peps/pep-3103/

[3] FAQ Python - *Why isn't there a switch or case statement in Python?* : https://docs.python.org/3/faq/design.html#why-isn-t-there-a-switch-or-case-statement-in-python

ANALYSER LE WEB À L'AIDE D'UN WEB CRAWLER

Tristan COLOMBO

Êtes-vous prêts à concurrencer Google ? Sans aller jusqu'à obtenir un robot d'indexation rivalisant avec le géant du Web, je vous propose dans cet article de découvrir comment arpenter le Web à l'aide d'un Web Crawler, collecter des données et les analyser.

Mots-clés : Python, Web crawler, robot d'indexation, Google

Résumé

Pour effectuer des recherches sur le Web, il faut avoir réalisé auparavant une indexation de manière à fournir rapidement un résultat à une requête donnée. C'est le travail effectué en amont par tous les moteurs de recherche comme Google, Qwant, etc. Pour comprendre comment fonctionnent ces robots qui parcourent inlassablement le Web pour collecter des données et les indexer, nous développons notre propre *web crawler* et son moteur d'indexation associé.

Vous vous demandez comment fonctionne un « robot d'indexation » (on peut trouver *web crawler* comme mauvaise traduction en anglais) ? C'est justement le sujet que nous allons aborder dans cet article en commençant par cerner les actions affectées à ce robot, ce qui nous permettra de développer notre propre *web crawler* (qui n'est pas un robot d'indexation, mais un robot de collecte d'informations) et ainsi de comprendre un peu mieux le fonctionnement d'un moteur de recherche (même si bien entendu nous ne pourrons reproduire l'ensemble des traitements effectués par des moteurs tels que Google tant par souci de simplification que par absence d'informations sur les algorithmes employés).

1 Le web crawler

Un *web crawler* est un programme qui va parcourir automatiquement le Web en suivant récursivement les URLs présentes sur les pages qu'il visite. Au cours de sa « lecture » de chaque page web, il va collecter des informations : contenu textuel, images, vidéos, fichiers, adresses mail, etc. La partie « indexation » n'intervient qu'après la récupération des données. Un *web crawler* peut être utilisé pour effectuer du *web scraping*, c'est-à-dire simplement extraire des données cibles du Web. On voit donc que la tâche du *web crawler* ne concerne pas l'indexation : on collecte certaines données puis la partie analyse de ces données est transférée à un autre programme. Dans un premier temps, nous allons donc nous intéresser à la récolte des données avant d'envisager différents traitements possibles (notez que la collecte d'adresses mails servant ensuite à envoyer de vastes campagnes de spam utilise exactement la même technique).

1.1 Récolter les données

Le *web crawler* va utiliser une liste de pages sources qui seront ses pages de départ, appelées *seeds*. Le *web crawler* va ainsi maintenir une liste de pages à visiter qui ne contiendra au départ que les *seeds* puis dont le contenu augmentera au

fur et à mesure que des hyperliens seront détectés : les URLs seront ajoutées à la liste, appelée *crawl frontier*. Le principe du robot est alors simplement de remplir cette liste et de la mettre à jour avec les nouveaux liens découverts (on pourra limiter la profondeur de recherche par un paramètre dans notre implémentation) tout en récupérant le contenu ciblé (ou non).

Il faut également noter que dans un but d'indexation le *web crawler* devra parcourir inlassablement le Web à la recherche de mises à jour des pages. Il faudra bien entendu configurer ce paramètre avec des sites offrant des mises à jour plus fréquentes que d'autres.

Enfin, le protocole d'exclusion des robots, plus connu sous le simple nom de son fichier de directives **robots.txt** indique aux *web crawlers* quels sont les robots autorisés ou non à parcourir le site et à quelles pages ils ont accès. Prenons l'exemple d'un fichier **robots.txt** qui indique à tous les robots d'ignorer le répertoire **private** et au robot de Google d'ignorer en plus le répertoire **not_google** :

```
User-agent: *
Disallow: /private/

User-agent: googlebot
Disallow: /not_google/
```

Ceci n'est bien entendu que purement informatif, rien n'obligera notre robot à respecter ces indications à moins que nous ne le décidions.

1.2 Traitement des données

Les pages sont parcourues par le *web crawler*, mais que faire des données ?

Si l'objectif est de créer un moteur de recherche, il va falloir indexer les résultats au sein d'une base de données (on ne va pas reparcourir le Web à chaque recherche...).

L'objectif peut être beaucoup plus ciblé et dans ce cas le *web crawler* ne va récupérer sur chaque page que des informations pertinentes qu'il suffira de stocker. Ce peut être le cas d'un comparateur de prix par exemple ou encore une fois des robots de spammeurs.

2 Notre web crawler en Python

Vous l'avez compris, pour moi le *web crawler* parcourt le Web et le moteur d'indexation indexe. Du coup, mon implémentation va forcément traduire cet état d'esprit avec un objet **WebCrawler** chargé de lire les pages en *seeds* et de collecter les URLs issues des hyperliens des pages visitées. On pourra alors associer à cet objet un moteur permettant de traiter les données comme bon nous semble. Ce moteur se présentera sous la forme d'un objet et devra fournir un certain nombre de méthodes (nous verrons cela dans la suite).

2.1 Analyser du html

Pour commencer, pour analyser les pages html à la recherche des hyperliens (puis plus tard éventuellement d'autres tags), nous allons employer le module **html.parse** qui doit être utilisé d'une manière un peu particulière ; il faut créer une classe qui hérite de **HTMLParser** et cette classe doit contenir (là encore) certaines méthodes :

- **handle_starttag(self, tag, attrs)** : méthode déclenchée sur un tag ouvrant et permettant d'accéder à ses attributs ;
- **handle_endtag(self, tag)** : méthode déclenchée sur un tag fermant ;
- **handle_data(self, data)** : méthode déclenchée à l'intérieur d'un tag (entre deux balises ouvrante/fermante).

Il n'est bien sûr pas obligatoire d'écrire toutes ces méthodes si l'on n'en a pas besoin et d'autres méthodes sont disponibles **[1]**. Lorsque les données de la page Web seront récupérées sous la forme d'une chaîne de caractères, il faudra appeler la méthode **feed()** de **HTMLParser** en lui donnant la chaîne en paramètre, ce qui déclenchera son analyse et l'exécution des fonctions indiquées ci-dessus. Voici un exemple très simple d'utilisation permettant de lire une page et d'indiquer le contenu de l'ensemble des tags de titre **<hn>** où **n** sera passé en paramètre. Notre fichier s'appellera **TestHtmlParser.py** puisqu'il définit l'objet du même nom :

```
01: from html.parser import HTMLParser
02: from urllib.request import urlopen
03: from urllib import parse
04:
05: class TestHtmlParser(HTMLParser):
06:
07:     def __init__(self, header_n, *args, **kwargs):
08:         super().__init__(*args, **kwargs)
09:         self.__currentTag = ''
10:         self.__search = 'h' + str(header_n)
11:
12:     def handle_starttag(self, tag, attrs):
13:         if tag == self.__search:
14:             self.__currentTag = self.__search
15:
16:     def handle_endtag(self, tag):
17:         self.__currentTag = ''
```

```
18:
19:     def handle_data(self, data):
20:         if self.__currentTag == self.__search:
21:             print('{} => {}'.format(self.__search, data))
22:
23:
24: if __name__ == '__main__':
25:     parser = TestHtmlParser(1)
26:     url = 'http://www.gnulinuxmag.com'
27:     print('Ouverture de', url)
28:     response = urlopen(url)
29:     htmlBytes = response.read()
30:     htmlString = htmlBytes.decode("utf-8")
31:     parser.feed(htmlString)
32:     print('*** End ***')
```

Les imports sont réalisés dans les lignes 1 à 3. En ligne 5, nous indiquons que la classe **TestHtmlParser** hérite de **HTMLParser**. Il ne faut donc pas oublier dans le constructeur de faire appel au constructeur de la classe mère (voir ligne 8). Nous ajoutons deux attributs dans ce constructeur : **__currentTag** (ligne 9) qui indique quel est le tag ouvert courant et **__search** qui contient le nom du tag recherché, élaboré en concaténant la lettre **h** avec la valeur **header_n** fournie en paramètre lors de l'appel du constructeur. La méthode **handle_starttag()** des lignes 12 à 14 vérifie si le tag lu est le tag recherché et si c'est le cas elle met à jour la valeur de **__currentTag**. La méthode **handle_endtag()** des lignes 16 et 17 remet l'attribut **__currentTag** à vide lorsque l'on rencontre un tag fermant. Enfin, la méthode **handle_data()** des lignes 19 à 21 vérifie si le tag courant est bien le tag recherché et, si c'est le cas, affiche les données lues sur la page web pour ce tag.

Dans le programme principal, nous créons une instance de **TestHtmlParser** en passant en paramètre la valeur **1** pour indiquer que nous recherchons les tags **<h1>** (ligne 25). En ligne 26, nous créons une variable **url** contenant l'adresse de la page que nous souhaitons analyser. Nous ouvrons ensuite cette page (ligne 28), nous la lisons sous forme de bytes en ligne 29 et en ligne 30 nous la convertissons en chaîne de caractères (la conversion peut également être effectuée par **str(htmlBytes, 'utf-8')**). Enfin, en ligne 31 nous appelons la méthode **feed()** chargée de lancer l'analyse.

Voici ce que l'on obtient à l'exécution :

```
$ python3 TestHtmlParser.py
Ouverture de http://www.gnulinuxmag.com
h1 => Menu Ð
h1 => À nouveau disponible en kiosque : notre guide spécial
surveillance !
h1 => L'édito de GNU/Linux Magazine n°197 !
h1 => Faites vos premiers pas en réalité augmentée avec GNU/Linux
Magazine !
h1 => Découvrez la préface du hors-série Mémo Python !
h1 => Accélérez vos développements Python !
h1 => L'édito de GNU/Linux Magazine n°196 !
h1 => Créez votre première intelligence artificielle !
h1 => À nouveau disponible en kiosque : le guide pour débuter en
C++ !
h1 => Parcourir les articles
h1 => Accélérez vos développements Python !
h1 => Rechercher
h1 => Connexion
h1 => En kiosque !
h1 => Mots-clés
   *** End ***
```

Note

Nous considérons ici comme naturel que l'URL pointe vers une page html... mais dans un programme automatisé ce ne sera pas toujours le cas ! Il faudra alors ajouter un test sur le type de données pointé par l'URL :

```
...
28:     response = urlopen(url)
29:     maintype = response.info().get_content_maintype()
30:     subtype = response.info().get_content_subtype()
31:     if maintype == 'text' and subtype == 'html':
32:         htmlBytes = response.read()
33:         htmlString = htmlBytes.decode("utf-8")
34:         parser.feed(htmlString)
35:     print('*** End ***')
```

La méthode **info()** permet d'obtenir des informations sur le contenu de la page avec notamment les type et sous-type de données (méthodes **get_content_maintype()**et **get_content_subtype()**). Nous récupérons en fait ici le *Content-type* sous la forme de deux valeurs. Ainsi, pour une page html où le *Content-type* serait **text/html**, nous obtenons **maintype = 'text'** et **subtype ='html'**.

2.2 Le web crawler proprement dit

Nous avons maintenant toutes les briques pour créer notre *web crawler* dans **WebCrawler.py**. Je commenterai le code au fil de celui-ci :

```
01: from html.parser import HTMLParser
02: from urllib.request import urlopen
03: from urllib import parse
04:
05: class WebCrawler(HTMLParser):
06:
07:     def __init__(self, maxPages, engine=None, *args, **kwargs):
08:         super().__init__(*args, **kwargs)
```

```
09:         self.__crawlFrontier = []
10:         self.__seeds = []
11:         self.__maxPages = maxPages
12:         self.__currentUrl = None
13:         self.__visited = []
14:         self.__engine = engine
```

On retrouve dans ces lignes la structure de notre test précédent. Le constructeur de **WebCrawler** va prendre comme paramètres **maxPages** qui indique le nombre maximal de pages à visiter et **engine** qui associe un moteur de traitement des données au *web crawler*. Pour l'instant nous laisserons la valeur d'**engine** à **None**, c'est-à-dire que nous n'utiliserons pas de moteur. Dans les lignes 9 à 14 viennent ensuite les déclarations d'attributs : **__crawlFrontier** va contenir la liste des URLs à visiter, **__seeds** la liste des URLs de départ, **__maxPages** le nombre de pages maximal à visiter, **__currentUrl** l'URL en cours de traitement, **__visited** sera une liste dans laquelle nous stockerons les URLs des sites déjà visités, et enfin **__engine** contiendra l'instance du moteur de traitement des données.

```
16:     def addSeeds(self, *args):
17:         for url in args:
18:             self.__seeds.append(url)
```

La méthode **addSeeds()** permet d'ajouter des URLs (notez les arguments de taille non fixée ***args**) dans la liste **__seeds**.

```
20:     def __initCrawler(self):
21:         self.__crawlFrontier = self.__seeds.copy()
```

Lors de l'initialisation du *web crawler*, nous copions les URLs de **__seeds** dans **__crawlFrontier** : **__crawlFrontier** contient la liste des URLs à visiter et elle est mise à jour après chaque chargement d'URL.

```
23:     def __addCrawlFrontier(self, url):
24:         url = parse.urljoin(self.__currentUrl, url)
25:         if url not in self.__visited:
26:             self.__crawlFrontier.append(url)
```

__addCrawlFrontier() permet d'ajouter une URL à **__crawlFrontier** après vérification que cette dernière n'a pas déjà été visitée (ligne 25). Vous noterez l'utilisation de **parse.urljoin()**en ligne 24 : cette fonction permet de construire une URL « absolue » depuis une URL de base et une autre URL **[2]**.

```
28:     def __currentVisited(self):
29:         return self.__currentUrl in self.__visited
30:
31:     def __addCurrentVisited(self):
32:         self.__visited.append(self.__currentUrl)
```

Les deux méthodes ci-dessus permettent respectivement de savoir si un site a été visité et d'indiquer que le site présent à l'URL **__currentUrl** a été visité. Notez que la fonction **__currentVisited()** retourne un booléen : **True** si **__currentUrl** est présent dans **__visited** et **False** sinon (ligne 29).

```
34:     def handle_starttag(self, tag, attrs):
35:         if tag == 'a':
36:             for attr, value in attrs:
37:                 if attr == 'href':
38:                     self.__addCrawlFrontier(value)
39:                     break
40:         if self.__engine is not None:
41:             self.__engine.handle_starttag(tag, attrs)
42:
43:     def handle_endtag(self, tag):
44:         if self.__engine is not None:
45:             self.__engine.handle_endtag(tag)
46:
47:     def handle_data(self, data):
48:         if self.__engine is not None:
49:             self.__engine.handle_data(data)
```

Les trois fonctions débutant par **handle_** sont les fonctions utilisées par **HTMLParser** et la méthode **feed()** comme nous l'avons vu précédemment. Ici, dans **handle_starttag()**, si le tag **<a>** est détecté alors on recherche l'attribut **href** et on récupère son contenu pour ajouter l'URL à la liste des sites à visiter (ligne 38). De plus, si un moteur a été défini alors on exécute sa fonction **handle_starttag()**. Pour la méthode **handle_endtag()** des lignes 43 à 45, nous n'avons aucune action à effectuer ici, mais un moteur externe peut avoir besoin de déclencher une action lors de la détection d'un tag fermant. C'est pour cette raison que l'on ne fait que tester la présence d'un moteur et l'appel de sa méthode **handle_endtag()**. La méthode **handle_data()**des lignes 47 à 49 suit exactement le même principe.

```
51:     def start(self):
52:         pages = 0
53:         self.__initCrawler()
54:         while pages < self.__maxPages:
55:             try:
56:                 if self.__crawlFrontier == []:
57:                     self.__stop()
58:                 self.__currentUrl = self.__crawlFrontier.pop(0)
59:                 if not self.__currentVisited():
60:                     pages += 1
61:                     self.__addCurrentVisited()
62:                     print('Step {} : visiting : {}'.format(pages,
self.__currentUrl), end='')
63:                     self.__engine.addCurrentUrl(self.__currentUrl)
64:
65:                     # Read the page
66:                     response = urlopen(self.__currentUrl)
67:                     maintype = response.info().get_content_
maintype()
68:                     subtype = response.info().get_content_subtype()
69:                     if maintype == 'text' and subtype == 'html':
70:                         htmlBytes = response.read()
71:                         htmlString = htmlBytes.decode("utf-8")
72:                         self.feed(htmlString)
73:                     if self.__engine is not None:
74:                         self.__engine.analyze(maintype, subtype,
htmlString)
75:                     print(' [OK]')
76:             except Exception:
77:                 print(' [FAILED]')
78:         self.__stop()
```

La méthode **start()** va lancer le parcours du Web à la recherche des URLs. On commence naturellement par réaliser des initialisations avec la mise à **0** de **pages** (le nombre de pages visitées) et l'appel de **__initCrawler()** dans les lignes 52 et 53. Ensuite, tant que le nombre de pages visitées est inférieur à **__maxPages**, on extrait une URL de **__crawlFrontier** (ligne 58) et si cette URL n'a pas déjà été visitée on l'ajoute à la liste des sites visités (ligne 61) et on lit la page en s'assurant qu'il s'agisse bien d'une page html comme nous l'avons vu dans notre test de **HTMLParser** (lignes 65 à 71). Si un moteur est présent, peut-être voudra-t-il traiter d'autres types de fichier donc il est appelé en ligne 74 via la méthode **__engine.analyze()**.

```
80:     def __stop(self, callback=None):
81:         print('*** End ***')
82:         exit(0)
```

La méthode **__stop()** permet de quitter proprement le programme après affichage d'un message.

Le script permettant d'appeler notre *web crawler* sans moteur est le suivant :

```
01: from WebCrawler import WebCrawler
02:
03: if __name__ == '__main__':
04:     webCrawler = WebCrawler(10)
05:     webCrawler.addSeeds('nimportequoi', 'http://www.
gnulinuxmag.com', 'http://www.linux-pratique.com')
06:     webCrawler.start()
```

Et nous voyons bien à l'exécution que des hyperliens ont été découverts :

```
$ python3 crawler.py
['nimportequoi', 'http://www.gnulinuxmag.com', 'http://www.linux-
pratique.com']
Step 1 : visiting : nimportequoi [FAILED]
Step 2 : visiting : http://www.gnulinuxmag.com [OK]
Step 3 : visiting : http://www.linux-pratique.com [OK]
Step 4 : visiting : http://www.gnulinuxmag.com/ [OK]
Step 5 : visiting : http://www.gnulinuxmag.com#content [OK]
```

```
Step 6 : visiting : http://boutique.ed-diamond.com/7-gnulinux-magazine
[OK]
Step 7 : visiting : https://boutique.ed-diamond.com/abonnements/3-gnu-
linux-magazine [OK]
Step 8 : visiting : https://github.com/GLMF [OK]
Step 9 : visiting : http://www.editions-diamond.fr/devenir-auteur/ [OK]
Step 10 : visiting : https://boutique.ed-diamond.com/content/25-en-
savoir-plus-sur-linux-pratique [OK]
*** End ***
```

Remarquez que dans l'étape 2 l'URL http://www.gnulinuxmag.com est visitée et dans l'étape 4 c'est l'URL http://www.gnulinuxmag.com/, soit la même page... La norme veut que l'URL d'un répertoire se termine par un slash et pas le nom d'un fichier. Toutefois rien ne nous assure que l'utilisateur intégrera le slash final dans les URLs données pour les *seeds* ni que le nom des pages sera aisément identifiable (par exemple https://boutique.ed-diamond.com/abonnements/3-gnu-linux-magazine est une page). Il n'y a donc que deux solutions : ne rien faire de particulier et tant pis si on relit quelques pages (solution retenue) ou ajouter dans les *crawl frontier* pour chaque URL la version avec et sans slash final.

Dans cet objet **`WebCrawler`**, nous avons autorisé l'ajout d'un moteur permettant de traiter les données. Comme vous l'avez vu dans les lignes 41, 45, 49 et 73, l'instance d'un moteur doit disposer de méthodes spécifiques. Pour faire les choses proprement, il faudrait définir une interface pour indiquer au développeur comment construire la classe qui servira de moteur. Problème : en Python les interfaces n'existent pas... Nous allons donc créer une classe qui contiendra les en-têtes des méthodes et dont le code sera un simple **`pass`**. Cela permettra de créer un moteur en héritant de cette classe générique et de savoir quelles méthodes définir (même si l'utilisation de ce mécanisme sera facultative). Voici donc le code de **`Engine.py`** définissant la classe **`Engine`** sur laquelle nous allons nous baser pour définir nos futurs moteurs :

```
01: class Engine:
02:
03:     def __init__(self):
04:         self._currentTag = ''
05:         self._currentUrl = ''
06:
07:     def addCurrentUrl(self, url):
08:         self._currentUrl = url
09:
10:     def handle_starttag(self, tag, attrs):
11:         pass
12:
13:     def handle_endtag(self, tag):
14:         self._currentTag = ''
15:
16:     def handle_data(self, data):
17:         pass
18:
19:     def analyze(self, maintype, subtype, data):
20:         pass
```

Ici les seules véritables instructions sont les déclarations des attributs **`_currentTag`** et **`_currentUrl`** en ligne 4 et les méthodes **`addCurrentUrl()`** lignes 7 et 8 et **`handle_endtag ()`** lignes 13 et 14. L'attribut **`_currentTag`** permettra d'enregistrer le tag courant en vue de tests dans **`handle_data()`**. **`_currentUrl`** permettra d'accéder à l'URL de la page et la méthode **`addCurrentUrl()`** permettra de le mettre à jour.

Notez que les attributs **`_currentTag`** et **`_currentUrl`** ne sont « protégés » que par un simple *underscore* : avec un double *underscore* ils n'auraient pas été hérités.

3 Ajout d'un moteur d'Indexation

En guise d'exemple, nous allons ajouter un moteur d'indexation simple qui va pondérer les termes rencontrés dans les pages web de la manière suivante :

- on ignore les déterminants, les prépositions, les pronoms personnels, etc. ;
- suivant dans quel tag un terme est présent, son score sera calculé d'après le tableau ci-dessous :

Tag	Score
<title>	150
<h1>	100
<h2>	50
autre	10

- un terme présent plusieurs fois sur une page verra ses scores additionnés.

Les données calculées seront ensuite stockées dans une base NoSQL **Redis [3]** et nous aurons donc besoin du module **`redis`** :

```
$ sudo pip3 install redis
```

Voici le code du moteur d'indexation :

```
01: from Engine import Engine
02: import redis
03:
04: class IndexEngine(Engine):
05:
06:     IGNORE = ['', 'de', 'du', 'des', 'dans', 'le', 'la',
'les', 'a', 'à', 'au', 'aux',
07:               'sur', 'et', 'je', 'tu', 'il', 'elle', 'on',
```

```
'nous', 'vous', 'ils', 'elles',
08:              'par', 'si', 'que', 'qui', 'quoi', 'ou', 'où',
'chez', '|', '!', 'pour']
09:
10:     def __init__(self, host='localhost', port=6379):
11:         super().__init__()
12:         self.__redis = redis.StrictRedis(host=host, port=port)
13:         self.__redis.flushall()
14:
15:     def __addToIndex(self, data, score):
16:         words = data.lower().split(' ')
17:         for word in words:
18:             word = word.strip()
19:             if word not in IndexEngine.IGNORE:
20:                 if self.__redis.exists(word):
21:                     dbData = self.__redis.hgetall(word)
22:                     url = bytes(self._currentUrl, 'utf-8')
23:                     if url in dbData:
24:                         dbData[url] =
bytes(str(int(dbData[url]) + score), 'utf-8')
25:                     else:
26:                         dbData[self._currentUrl] = score
27:                     self.__redis.hmset(word, dbData)
28:                 else:
29:                     self.__redis.hmset(word, {self._currentUrl :
score})
30:
31:     def handle_starttag(self, tag, attrs):
32:         self._currentTag = tag
33:
34:     def handle_data(self, data):
35:         if self._currentTag == 'title':
36:             self.__addToIndex(data, 150)
37:         elif self._currentTag == 'h1':
38:             self.__addToIndex(data, 100)
39:         elif self._currentTag == 'h2':
40:             self.__addToIndex(data, 50)
41:         elif self._currentTag != 'script' and self._currentTag
!= 'meta' and \
42:              self._currentTag != 'style' and self._currentTag
!= 'a':
43:             self.__addToIndex(data, 10)
```

Première constatation : nous n'avons pas besoin de la méthode **analyze()** et donc nous ne l'avons pas redéfinie. L'attribut de classe **IGNORE** permet d'indiquer les termes qui ne devront pas être indexés (lignes 6 à 8). Le constructeur des lignes 10 à 13 fait appel au constructeur de la classe mère **Engine** (ligne 11), se connecte à la base Redis en ligne 12 (il est possible de modifier les valeurs par défaut pour **host** et **port**), et enfin il efface le contenu de la base en ligne 13 (l'indexation commencera toujours d'une base vierge dans notre programme par souci de simplification). La méthode **__addToIndex()** des lignes 15 à 29 va prendre en paramètres une chaîne de caractères (**data**) et un **score**. La chaîne va être segmentée en mots (ligne 16) : on passe tous les caractères en minuscules avec **lower()** et on découpe suivant le caractère espace avec **split(' ')**. Ensuite, pour chacun des mots **word** de la liste (lignes 19 à 29), on nettoie le mot en enlevant les caractères invisibles inutiles (espace, saut de ligne, etc.) à l'aide de **strip()** en ligne 18. Puis, si le mot ne fait pas partie de la liste de mots à ignorer (ligne 19) alors, s'il n'existe pas dans la base, on l'ajoute (ligne 29) sous la forme d'un dictionnaire **{url: score}**. Sinon, on récupère les données de la base en ligne 21. Il s'agit d'un dictionnaire dans lequel les données sont stockées sous forme de bytes. Pour effectuer des comparaisons, il faut donc convertir nos chaînes en bytes, ce qui est fait en ligne 22 avec **self._currentUrl**. Nous cherchons ensuite si cette URL est présente dans le dictionnaire. Si c'est le cas, alors le mot est déjà présent sur la page et il faut augmenter son score (ligne 24), toujours en pensant au problème de conversion des bytes. Sinon, on ajoute un nouvel index en ligne 26. La méthode **handle_starttag()** des lignes 31 et 32 permet simplement de savoir dans quel tag on se trouve lors de l'analyse de la page et la méthode **handle_data()** des lignes 34 à 43 effectue un filtre sur les tags pour affecter un score aux mots présents dans ceux-ci. En ligne 41 à 42 on décide de ne pas indexer les termes présents dans les balises **<script>**, **<meta>**, **<style>** et **<a>**.

Pour lancer l'indexation avec ce moteur, il va falloir changer l'appel de notre *web crawler* dans **crawler.py** :

```
01: from WebCrawler import WebCrawler
02: from IndexEngine import IndexEngine
03:
04: if __name__ == '__main__':
05:     webCrawler = WebCrawler(10, engine=IndexEngine())
06:     webCrawler.addSeeds('http://www.gnulinuxmag.com',
'http://www.linux-pratique.com')
07:     webCrawler.start()
```

Lançons le code pour obtenir une base d'indexation :

```
$ python3 crawler.py
Step 1 : visiting : http://www.gnulinuxmag.com [OK]
Step 2 : visiting : http://www.linux-pratique.com [OK]
Step 3 : visiting : http://www.gnulinuxmag.com/ [OK]
Step 4 : visiting : http://www.gnulinuxmag.com#content [OK]
Step 5 : visiting : http://boutique.ed-diamond.com/7-gnulinux-
magazine [OK]
Step 6 : visiting : https://boutique.ed-diamond.com/abonnements/3-
gnu-linux-magazine [OK]
Step 7 : visiting : https://github.com/GLMF [OK]
Step 8 : visiting : http://www.editions-diamond.fr/devenir-auteur/
[OK]
Step 9 : visiting : https://boutique.ed-diamond.com/content/25-en-
savoir-plus-sur-linux-pratique [OK]
Step 10 : visiting : https://boutique.ed-diamond.com/content/27-
en-savoir-plus-sur-open-silicium [OK]
*** End ***
```

Il ne reste plus qu'à exploiter les données récoltées dans la base...

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : http://**boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Offre	SUPPORT / ABONNEMENT (Prix en Euros / France Métropolitaine)	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
LM	11n° GLMF	LM1	65,-	LM12	95,-	LM13	149,-	LM123	174,-
LM+	11n° GLMF + 6n° HS	LM+1	118,-	LM+12	177,-	LM+13	197,-	LM+123	256,-
LES COUPLAGES « LINUX »									
A	11n° GLMF + 6n° LP	A1	95,-	A12	140,-	A13	218,-	A123	263,-
A+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS	A+1	182,-	A+12	263,-	A+13	300,-	A+123	386,-
B	11n° GLMF + 6n° MISC	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	11n° GLMF + 6n° HS + 6n° MISC + 2n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	11n° GLMF + 6n° LP + 6n° MISC	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
F	11n° GLMF + 6n° HK* + 4n° OS	F1	125,-	F12	188,-	F13	229,-*	F123	292,-*
F+	11n° GLMF + 6n° HS + 6n° HK* + 4n° OS	F+1	183,-	F+12	275,-	F+13	287,-*	F+123	379,-*
LES COUPLAGES « GÉNÉRAUX »									
H	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR : http://www.ed-diamond.com pour consulter toutes les offres !

Rappels Redis

Redis est une base de données NoSQL. Son installation sur les systèmes basés sur Debian se fait de la manière suivante :

```
$ sudo apt install redis-server
```

Et pour démarrer le serveur :

```
$ sudo service redis-server start
```

Pendant le développement, vous pouvez utiliser l'interface en ligne de commandes pour vérifier les données et éventuellement réinitialiser la base :

```
$ redis-cli
127.0.0.1:6379>
```

- Afficher la liste des clés :

```
127.0.0.1:6379> keys *
...
1102) "utiliser"
1103) "ceux"
```

- Vider la base :

```
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> keys *
(empty list or set)
```

4 Et pour développer un moteur de recherche ?

Le rôle du moteur de recherche va être d'interroger la base constituée par l'indexation. Il s'agit alors simplement de demander un terme à l'utilisateur et d'afficher les résultats en les ordonnant par scores descendants :

```
01: import redis
02:
03: if __name__ == '__main__':
04:     redis = redis.StrictRedis()
05:     while True:
06:         results = []
07:         search = input('Votre recherche : ').strip()
08:         if search == 'quit':
09:             break
10:         dbData = redis.hgetall(search)
11:         for url, score in dbData.items():
12:             results.append((str(url, 'utf-8'), int(score)))
13:         results.sort(key = lambda elt : elt[1], reverse=True)
14:         for url, score in results:
15:             print('- {} ({})'.format(url, score))
16:         print()
```

En ligne 4, on se connecte à la base Redis avec les *host* et *port* par défaut (à ajouter dans les paramètres si vous les avez modifiés). On demande à l'utilisateur de saisir un terme en ligne 7 (que l'on nettoie avec **`strip()`**). Les données sont ensuite récupérées (ligne 10) puis transformées en liste de tuples de la forme **`(url, score)`** dans les lignes 11 et 12. Ceci permet alors de les trier par score à l'aide d'une fonction de tri personnalisée (ligne 13). Enfin, dans les lignes 14 et 15 nous affichons les résultats. Le code étant placé dans un **`while`** sans fin (ligne 5), il faudra saisir le mot **`quit`** pour quitter le programme (lignes 8 et 9).

Voici un exemple de recherche :

```
$ python3 myGoogle.py
Votre recherche : linux
- http://www.linux-pratique.com (370)
- https://boutique.ed-diamond.com/content/25-en-savoir-plus-sur-
linux-pratique (270)
- https://boutique.ed-diamond.com/abonnements/3-gnu-linux-magazine (100)
- http://boutique.ed-diamond.com/7-gnulinux-magazine (20)
- http://www.editions-diamond.fr/devenir-auteur/ (20)
- https://boutique.ed-diamond.com/content/27-en-savoir-plus-sur-
open-silicium (10)

Votre recherche : quit
```

Conclusion

Nous voici arrivés au terme du développement de notre moteur de recherche. Il y a bien entendu encore beaucoup de travail si nous voulons le rendre efficace : optimisation de l'indexation, calcul d'un score plus pertinent, tri des données, suppression des caractères parasites, etc. Mais nous sommes en possession de l'architecture de base, qui plus est évolutive puisqu'il est possible d'associer divers moteurs à notre *web crawler*. Il ne reste donc plus qu'à développer des moteurs qui traiteront de manière spécifique les données glanées par notre robot. Et si vous nous voulez pas démarrer votre projet de zéro, vous pouvez aller jeter un œil au module **Scrapy [4]** qui devrait bien vous aider... ■

Références

[1] Documentation sur **`html.parser`** : https://docs.python.org/3.5/library/html.parser.html

[2] Documentation pour **`parse.urljoin()`** : https://docs.python.org/3.5/library/urllib.parse.html

[3] API du module redis pour Python : https://redis-py.readthedocs.io/en/latest/

[4] Site officiel de Scrapy : https://scrapy.org/

WEB CRYPTOGRAPHY API

Sylvain NAYROLLES - [Développeur logiciel de sécurité]

La WebCryptoAPI est une spécification W3C qui a pour principal objectif de fournir, aux programmeurs d'applications web côté client, les principaux outils cryptographiques, et ceci à des fins de sécurité et de confidentialité.

Mots-clés : web cryptography api, client, javascript, sécurité, cryptographie

Résumé

Nous allons redonner le contrôle des données au client en nous appuyant sur ce dernier pour fournir l'ensemble des outils cryptographiques via la WebCryptoAPI. Cette dernière offre l'ensemble des outils et algorithmes suffisant pour assurer un niveau de confidentialité et de sécurité semblable à ce que l'on trouve côté serveur.

La WebCryptoAPI permet d'utiliser un ensemble standardisé d'outils cryptographiques les plus utilisés dans le monde du réseau, mais cette fois-ci côté client. Il est donc maintenant possible, via **JavaScript**, de réaliser du chiffrement, de calculer des signatures, etc. Le but ici est bien d'augmenter le niveau de confidentialité des données qui sont stockées côté serveur. Il existe **TLS** me direz-vous ? TLS est certes un protocole d'échange fiable qui assure un haut niveau de confidentialité, que nous ne nous permettrons pas de remettre en cause ; il est plutôt question ici de remettre en cause une architecture, qui jusqu'ici privilégiait la manipulation des données côté serveur, et qui maintenant peut être entièrement manipulée côté client. La machine cliente est maintenant le moteur cryptographique de l'application.

Cette spécification est maintenant implémentée (partiellement, mais suffisamment) dans la plupart des navigateurs internet.

1 Familiarisation avec l'API

La WebCryptoAPI a été pensée, dans sa conception, de manière à pouvoir s'insérer très facilement dans le monde JavaScript moderne. Elle est donc disponible via l'objet global `window` :

```
window.crypto
```

Elle exploite massivement le concept de *Promise* (promesse) en JavaScript, et utilise la représentation de données via **asm.js**, pour des raisons de performances.

Note

Le concept de *promise* est un contrat de réalisation asynchrone d'une tâche coûteuse. Nous allons demander la réalisation de la tâche en lui fournissant une *callback* en cas de succès et une *callback* en cas d'échec. De nombreuses API sur le sujet avaient fleuri avant une normalisation via la norme **ECMAScript 6**. C'est cette dernière que la WebCryptoAPI utilise pour ses fonctions `importKey`, `deriveKey`, `digest`, `encrypt`, `decrypt`, etc. Elle consiste en une action listée précédemment avec un appel chaîné à `then` pour la clause de succès ou `catch` pour la clause d'échec. Nous pouvons constater l'avancée de l'implémentation de la norme ECMAScript 6 via le tableau **[1]**.

Note

Asm.js est un sous-ensemble de JavaScript permettant un gain de performances conséquent pour la manipulation de structures de données fortement typées. Il fut introduit avec l'avènement du **WebGL**, mais c'est avec le *backend* **LLVM emscripten** qu'il a pris toute son importance. La WebCryptoAPI se sert énormément de cet ensemble à des fins de performances. Il se base sur un type concret `ArrayBuffer` et des vues typées : `Uint8Array`, `Uint16Array`, et `Uint32Array`.

Les principales primitives cryptographiques sont accessibles via la propriété, la seule d'ailleurs, **subtle** de l'objet **Crypto** :

```
window.crypto.subtle
```

L'objet **SubtleCrypto** ainsi obtenu fournit un ensemble de méthodes cryptographiques que nous allons explorer dans la suite de cet article. Ceci représente donc une interface commune pour tous les algorithmes cryptographiques. C'est pour cette raison que les paramètres de ces fonctions sont contenus dans un dictionnaire et sont donc très dépendants de l'algorithme sous-jacent. Ces paramètres ainsi que l'ensemble des algorithmes disponibles font aussi partie de la normalisation **W3C** associée à la WebCryptoAPI.

2 Hasher des données

Les fonctions de *hashage* sont des fonctions mathématiques dites injectives ; c'est-à-dire qu'à un ensemble de départ elle n'associe qu'une seule et unique valeur dans l'ensemble d'arrivée, et que l'opération inverse n'existe pas. En informatique, il est très courant de se servir de fonctions de *hashage* pour identifier de façon unique une donnée, comme c'est le cas dans une table de *hashage*. Mais il est aussi courant de se servir de telles fonctions pour stocker les mots de passe utilisateur en les associant avec un *salt* (souvent le nom d'utilisateur).

Note

Le principe de *salt* est très important dès lors que nous stockons nos mots de passe sous forme de *hash* dans une base de données. Le *salt* permet d'apporter une information supplémentaire au mot de passe afin d'éviter l'identification du mot de passe dans le cadre d'une compromission de la base de données. Il existe de nombreuses bases sur Internet de *hash* (**MD5**, **SHA1** principalement) de mots de passe couramment rencontrés, permettant de remonter facilement au mot de passe originel. Mais si ce dernier contient en plus une valeur propre à l'utilisateur, cette opération est donc rendue quasiment impossible.

Côté client, les fonctions de *hashage* peuvent nous servir à réaliser des dérivations de clefs, afin de convertir des mots de passe textuels sous une forme bien plus exploitable pour les algorithmes de chiffrement.

Enfin, on peut utiliser ces dernières pour, par exemple, calculer l'empreinte d'une donnée souvent transférée, mais qui ne change pas souvent. On peut donc se servir d'une fonction de *hashage* afin de conserver l'empreinte de la donnée en local, puis de transférer cette dernière entre le client et le serveur.

La WebCryptoAPI propose quatre fonctions de *hashage*, toutes dérivées du standard SHA :

- **SHA-1** ;
- **SHA-256** ;
- **SHA-384** ;
- **SHA-512**.

La fonction SHA-1 étant fortement déconseillée, car on soupçonne l'existence de collision, il est conseillé d'utiliser l'une des trois dernières, comme nous l'indique ce bulletin d'actualité du CERT-FR **[2]**.

Calculer le SHA-256 d'une donnée se fait donc de la manière suivante :

```
window.crypto.subtle.digest(
    {
        name: "SHA-256",
    },
    new Uint8Array([1,2,3,4]) // La donnée type source
)
.then(function(hash){
    // hash contient le résultat sous forme de ArrayBuffer
    console.log(new Uint8Array(hash));
})
.catch(function(err){
    console.error(err);
});
```

3 Gérer des clés

La gestion des clefs est un problème souvent rencontré quand on fait de la cryptographie. Il est donc tout à fait naturel que la WebCryptoAPI propose une interface complète pour la gestion de clefs cryptographiques.

3.1 Génération

Il est quelques fois nécessaire de générer une clef de manière aléatoire :

```
window.crypto.subtle.generateKey(
   algo, // Paramètre de l'algorithme
cible de la clef tel que la taille
   true,  // true si l'on veut extraire
la clef via exportKey
   ["encrypt", "decrypt", "sign",
"verify", "wrapKey", "unwrapKey"]
  // ensemble des fonctions pour
lesquelles la clef est disponible
  // ces valeurs dépendent du type
d'algorithme
)
.then(function(key){
   // key est de type CryptoKey
})
.catch(function(err){
});
```

La génération des clefs se fait donc très facilement. Il fut défini, dès le début, une partie spécifique pour l'algorithme pour lequel cette clef est générée. Cette partie spécifique fournit la souplesse nécessaire pour une API multi-algorithmes.

Cette fonction est disponible pour l'ensemble des algorithmes cryptographiques que propose l'API. Si l'on veut générer une clef pour un algorithme de type **AES-CBC-256**, on utilisera la configuration suivante :

```
window.crypto.subtle.generateKey(
    {
      name: "AES-CBC",
      length: 256
    },
    false,
    ["encrypt", "decrypt"]
)
.then(function(key){
})
.catch(function(err){
});
```

Un cas d'usage de la génération de clef peut être, par exemple, l'initialisation d'un nouvel utilisateur.

3.2 Importation et Exportation

Il est donc possible d'importer une clef cryptographique stockée sous différents formats. Les formats supportés sont donc très adhérents à l'algorithme cryptographique ciblé. Dans le cadre d'algorithmes symétriques, tels que **AES**, le format **raw** est privilégié, car simple. Les algorithmes asymétriques, tels que **RSA**, ont une gestion des clefs bien plus complexe et offrent des formats bien plus structurés, dus essentiellement à l'avènement des certificats, tels que **pkcs8**. Comme la génération, une clef importée peut être limitée à une surface fonctionnelle, telle que **`encrypt`** ou **`decrypt`**.

En plus du format raw, il existe un format très utilisé au sein de la WebCryptoAPI : le **jwk**, pour *JSON Web Key,* formalisé au sein de la RFC 7517 **[3]**. Ce format fut formalisé au sein du sigle **JOSE** pour *Javascript Object Signing and Encryption*. Étant au format Json, il existe des facilités pour les stocker et les manipuler côté client.

Il est donc tout à fait possible d'exporter des clefs, dans la mesure où cette action fut autorisée lors de la génération ou de l'importation. La gestion avancée des clefs, telle que la propose la WebCryptoAPI, répond à un besoin avancé en cryptographie de l'application développée. Il est important de savoir que cela existe, mais dans cet article, et dans l'exemple qui suit, nous nous contenterons de dériver la clef à partir d'un mot de passe utilisateur.

```
window.crypto.subtle.importKey(
    "raw",                              // format de la clef
    ArrayBuffer([1, 2, 3, 4 ..., 256]),  // clef au bon format
    {
       name: "AES-CBC",                // Algorithme cible
       length: 256
    },
    false,
    ["encrypt", "decrypt"]
)
.then(function(key){
  // retourne l'objet clef sous forme de KeyObject
})
.catch(function(err){
});
```

4 Chiffrer et déchiffrer des données

Enfin, nous arrivons au cœur de notre sujet : pouvoir manipuler des données chiffrées dans un client. L'API de chiffrement ou de déchiffrement est donc semblable à ce que nous venons de voir. Ces opérations sont donc accessibles via les fonctions **`encrypt`** et **`decrypt`**, dont le prototype est équivalent aux autres fonctions déjà rencontrées :

```
window.crypto.subtle.encrypt(
    algo, // Paramètres d'identification de l'algorithme cible
    key, // clef au format interne générée par generateKey ou importKey
    data // Donnée claire au format ArrayBuffer
)
.then(function(encrypted){
  // encrypted sont les données chiffrées sous forme d'ArrayBuffer
})
.catch(function(err){
});
```

```
window.crypto.subtle.decrypt(
    algo, // Paramètres d'identification de l'algorithme cible
    key, // clef au format interne générée par generateKey ou importKey
    data // Données chiffrées au format ArrayBuffer
)
.then(function(decrypted){
  // decrypted sont les données claires sous forme d'ArrayBuffer
})
.catch(function(err){
});
```

5 Authentification

En cryptographie, cohabitent deux objectifs différents : le chiffrement et l'authentification. Dans les parties précédentes, nous nous sommes focalisés sur le chiffrement, mais la WebCryptoAPI fournit bien évidemment des outils pour assurer l'authentification des données, via deux fonctions principales : **sign** et **verify**. Ces interfaces respectent l'esprit initié par l'API depuis le début :

```
window.crypto.subtle.sign(
    algo, // paramètres de l'algorithme
    key,  // keyobject issue de generateKey ou de importKey
    data  // ArrayBuffer des données à chiffrer
)
.then(function(signature){
})
.catch(function(err){
});
```

Et la fonction de vérification :

```
window.crypto.subtle.verify(
    algo, // paramètres de l'algorithme
    key,  // KeyObject issue de generateKey ou de importKey
    signature, // ArrayBuffer de la signature à vérifier
    data  // ArrayBuffer des données à chiffrer
)
.then(function(signature){
})
.catch(function(err){
});
```

6 Un peu de pratique

Nous allons développer une application cliente pouvant manipuler des données chiffrées. Notre serveur pourra enregistrer nos données chiffrées via une interface GET minimaliste. Ici, nous allons nous attarder sur la partie cliente qui pourra, soit afficher la donnée chiffrée, soit chiffrer cette dernière avant de l'envoyer au serveur de stockage.

Comme dans la plupart des bibliothèques JavaScript, nous allons initier notre API au travers d'une fonction anonyme. Cette fonction définira un objet **Cypher** représentant l'interface de notre API :

```
(function() {
  Cypher = function() {
    this.key = null;
  }
  Cypher.prototype = {
  }
})();
this.Cypher = new Cypher();
```

L'objet ne contient qu'un seul membre qui est notre clef symétrique de chiffrement. Nous allons définir un ensemble de fonctions nous permettant de manipuler les types de données lors des opérations cryptographiques :

- **str2ab16** : cette fonction permet de convertir le type **string** JavaScript en **UintArray16**. Les strings JavaScript sont en utf16 ;
- **ab162str** : qui est la fonction inverse ;
- **str2ab8** : cette fonction permet de convertir le type **string** JavaScript en **UintArray8**. Cette fonction nous est utile quand la chaîne de caractères contient des données non unicode ;
- **ab82str** : qui est la fonction inverse.

Le code de ces fonctions ne sera pas détaillé ici, mais ces dernières, comme le reste de cette API, sera publié sur le GitHub de *GNU/Linux Magazine*.

Dans le prototype de l'objet **Cypher**, nous allons exposer trois principales fonctions :

- **importKey** ;
- **encrypt** ;
- **decrypt**.

importKey sera appelée, si nécessaire, lors des appels à **encrypt** et **decrypt**, selon l'état du membre **key** :

```
importKey : function (success, err) {
  // On demande le mot de passe
  var passwordText = window.prompt("Enter your password");

  // on convertit le mot de passe en ArrayBuffer
  var password = str2ab16(passwordText);

  var self = this;
 // on calcule le sha-256
  window.crypto.subtle.digest(
    {
      name: "SHA-256"
    },
    password
  )
  .then(function(passwordHash){
    // on importe la clef
    window.crypto.subtle.importKey(
      "raw",
      passwordHash,
      {
        name: "AES-CBC"
      },
      false,
      ["encrypt", "decrypt"]
    )
    .then(function(key) {
      // on enregistre la clef
      self.key = key;
      success();
    })
    .catch(err);
  })
  .catch(err);
}
```

Au travers de la fonction **window.prompt**, somme toute un peu basique, on demande de rentrer un mot de passe sous forme d'un texte. Nous réalisons ensuite la dérivation de cette dernière en appliquant un SHA-256. Le SHA-256 permet de générer une donnée dont la longueur est de 32 octets. Ceci est exactement le format de clef attendu par l'algorithme AES-256-CBC. Lors de l'opération d'import de la clef, nous précisons l'algorithme cible, ainsi que la surface fonctionnelle désirée.

Cette fonction sera donc appelée en premier lieu dans les fonctions **encrypt** et **decrypt** :

```
encrypt : function (text, success, err) {
  var self = this;
  // on vérifie l'état de la clef
  if (this.key == null) {
    this.importKey(function() {
      self.encrypt(text, success, err);
    }, err);
    return;
  }

  // on génère le vecteur d'initialisation
  var iv = window.crypto.getRandomValues(new Uint8Array(16));

  // on chiffre
  window.crypto.subtle.encrypt(
    {
      name: "AES-CBC",
      iv: iv,
    },
    this.key,
    // on convertit la chaîne utf-16 en ArrayBuffer
    str2ab16(text)
  )
  .then(function(encrypted) {
    // on contacte l'iv et le résultat
    var result = new Uint8Array(iv.buffer.byteLength + encrypted.
byteLength);
    result.set(iv, 0);
    result.set(new Uint8Array(encrypted), iv.buffer.byteLength);
    // on fournit le résultat en base64
    success(window.btoa(ab82str(result.buffer)));
  })
  .catch(err);
}
```

Le début de la fonction vérifie l'état de la clef, et appelle la fonction **this.importKey** si besoin. Ensuite, nous faisons appel à une nouvelle fonction de la WebCryptoAPI : **getRandomValues**. Cette dernière permet d'initialiser de manière aléatoire un vecteur de données. Ceci représente l'*IV* de notre algorithme ; c'est-à-dire une partie aléatoire, de même taille que la clef, mais qui n'a pas vocation à rester secrète (*Initialization Vector* ou Vecteur d'Initialisation). Ensuite, nous appelons la fonction de chiffrement à proprement parler. Le résultat ainsi obtenu, sera donc transmis à l'appelant sous la forme suivante : les 16 premiers octets représentants l'*IV* lors du chiffrement, et la suite étant la donnée chiffrée, le tout encodé en base64 via l'appel à la fonction **window.btoa**.

La fonction de déchiffrement est donc symétriquement la même :

```
decrypt : function (data, success, err) {
  var self = this;
  // on vérifie l'état de la clef
  if (this.key == null) {
    this.importKey(function() {
      self.decrypt(text, success, err);
    }, err);
    return;
  }

  // on décode la base64
  var dataAb = str2ab8(window.atob(data));

  // on extrait l'iv
  var iv = new Uint8Array(dataAb.slice(0, 16));

  // on extrait la donnée chiffrée
  var cipher = dataAb.slice(16);
```

```
  // on déchiffre
  window.crypto.subtle.decrypt(
    {
      name: "AES-CBC",
      iv: iv
    },
    self.key,
    cipher
  )
  .then(function(decrypted) {
    // on décode l'utf-16
    success(ab162str(decrypted));
  })
  .catch(err);
}
```

On vérifie toujours l'état de la clé ; on décode la base64 via l'appel à la fonction **`window.atob`**. Ensuite, on extrait l'*IV* qui est dans les 16 premiers octets ainsi que la donnée chiffrée et on applique l'algorithme.

La principale difficulté réside dans le format des données.

Conclusion

La WebCryptographyAPI est à mon sens une réelle avancée dans l'apport en termes de confidentialité que peut proposer une application web. Elle est intuitive, rapide, et standardisée. Le code présenté ici n'a pas été testé sur l'ensemble des navigateurs, mais déjà entre **Firefox** et **Chrome**, nous couvrons une bonne population d'utilisateurs linuxiens. Cet article avait pour vocation de vous exposer les possibilités que vous offrent les derniers standards inclus dans les navigateurs modernes, qui répondent à un besoin de confidentialité toujours croissant des utilisateurs connectés. ■

Références

[1] Tablau d'avancement de l'implémentation de la norme *ECMAScript 6* : http://kangax.github.io/compat-table/es6/

[2] Bulletin d'actualité du *CERT-FR* : http://www.cert.ssi.gouv.fr/site/CERTFR-2016-ACT-005/CERTFR-2016-ACT-005.html

[3] RFC 7517 sur *JSON Web Key* : https://tools.ietf.org/html/rfc7517

Pour aller plus loin

Le site https://github.com/diafygi/webcrypto-examples permet d'avoir un exemple de code pour chaque algorithme présent dans l'API.

TYPESCRIPT : DEVENEZ SÉRIEUX AVEC JAVASCRIPT

Stéphane MOUREY - [Mousse sur le Seeraiwer]

Depuis son apparition, JavaScript a fait bien du chemin. D'un langage dont les seuls intérêts semblaient être de valider les formulaires et de créer des effets visuels sur les pages web, on est passé à un instrument permettant la réalisation d'applications web de qualité professionnelle avec un confort d'utilisation bien difficile à obtenir avec d'autres technologies. Cette mutation n'a été possible que par le développement continuel d'outils venant corriger ses défauts d'origine. Microsoft apporte sa contribution avec TypeScript, un méta-langage permettant d'appliquer des concepts salvateurs que l'on trouve couramment dans d'autres langages, mais qui font cruellement défaut à JavaScript.

Mots-clés : JavaScript, compilation, optimisation, programmation objet, classes

Résumé

TypeScript est un méta-langage destiné à produire du JavaScript. Il offre aux développeurs des outils pour contourner les écueils de JavaScript tout en proposant des éléments méthodologiques venus d'autres langages, en particulier objet, permettant de travailler plus facilement sur des projets de grande dimension. Dans cet article, nous allons examiner quelques aspects fondamentaux de TypeScript, sans être exhaustifs toutefois, vu l'ampleur du sujet.

Commençons par dire que tout code JavaScript valide sera également un code TypeScript valide. Vous partez donc sur des bases connues, et vous n'aurez pas à jeter tous vos développements antérieurs sous prétexte qu' « il faut tout refaire », phrase clé de tout bon développeur qui a le don d'agacer tout bon chef de projet. Avec TypeScript, vous pourrez faire la migration en douceur, vos nouveaux développements s'ajoutant harmonieusement aux anciens, en attendant le jour où vous aurez gagné suffisamment d'assurance avec cette méthodologie pour reprendre progressivement l'ensemble.

Méthodologie ? Oui, car si TypeScript se présente comme un langage qu'il faudra compiler pour obtenir du JavaScript, son intérêt tient surtout à ce qu'il permette au développeur d'appliquer avec souplesse des paradigmes venus d'autres langages, en particulier objet. Si TypeScript peut faire *mieux* que JavaScript, il ne peut faire *plus*.

Un dernier point avant de rentrer dans le vif du sujet : TypeScript est un outil *open source*, mais il s'agit tout de même d'une production Microsoft. Il s'intègre donc bien avec des logiciels privateurs proposés par cet éditeur.

Par conviction, j'ai décidé de ne pas évoquer ces facilités. Vous verrez qu'il n'y a aucune nécessité d'y recourir. Si, toutefois, la question vous intéresse, vous trouverez toutes les informations utiles sur le site officiel **[1]**.

1 Préliminaires

1.1 Installation

Il est nécessaire de remplir un prérequis pour pouvoir utiliser TypeScript : il faut avoir d'abord installé **Node.js** et son gestionnaire de dépendances, **npm**. Il vous sera facile de le faire à l'aide du gestionnaire de paquets de votre distribution, ou encore en le téléchargeant depuis le site officiel **[2]**.

Sous Debian, cela donnera :

```
# apt-get install node.js npm
# ln -s "$(which nodejs)" /usr/bin/node
```

La dernière commande peut se révéler nécessaire comme vous le constaterez quelques lignes plus bas.

À partir de là, TypeScript s'installe à l'aide de la commande **npm** :

```
# npm install -g typescript
```

Vous pouvez maintenant tester la commande **tsc** :

```
$ tsc --help
/usr/bin/env: node: No such file or directory
```

Si vous rencontrez cette dernière erreur, provoquée par l'absence d'un lien symbolique, vous l'aurez facilement résolue à l'aide de la commande présentée précédemment, à savoir :

```
# ln -s "$(which nodejs)" /usr/bin/node
```

Si, par contre, vous n'avez *aucun* affichage, c'est que vous avez précédemment installé un paquet **node**, qui n'a rien avoir avec Node.js. Il ne s'agit plus que d'un paquet de transition, supprimé sur les versions récentes de Debian, aujourd'hui rebaptisé en **ax25-node**, destiné aux radio-amateurs.

1.2 Prise en main

La commande **tsc --help** devrait maintenant vous donner la liste des options qu'accepte la commande. Testons-la tout de suite, avec un simple fichier **hello.ts**. Notez que, par convention, les fichiers TypeScript ont l'extension **.ts**.

```
01: function hello(person) {
02:  return "Hello, " + person;
03: }
04:
05: document.body.innerHTML = greeter("world");
```

Lançons la compilation :

```
$ tsc hello.ts
hello.ts(5,27): error TS2304: Cannot find name 'greeter'.
$ ls
hello.js hello.ts
```

La commande s'achève sur un message d'erreur. Le fait est que je me suis inspiré d'un code trouvé dans la documentation officielle, et que j'ai renommé une fonction, mais que j'ai oublié de modifier l'appel en conséquence. L'erreur se produit lors de cet appel à la ligne 5, au caractère 27, le nom **greeter** n'est pas trouvé. J'aurais dû écrire **hello** à la place. L'intérêt de TypeScript apparaît déjà à cette étape : il permet une détection anticipée des erreurs lors de la compilation. Naturellement, cela fera sourire les développeurs d'à peu près n'importe quel autre langage, mais en ce qui concerne JavaScript, il s'agit d'un progrès considérable. Dans son fonctionnement ordinaire, JavaScript ne fournit aucun outil pour détecter ce genre de problèmes, et ce n'est que lorsque l'exécution du code aura effectivement lieu que celle-ci apparaîtra.

Remarquez que cela n'a pas empêché TypeScript de produire un fichier **hello.js**. Ainsi, les erreurs sont de simples avertissements, elles n'empêchent pas d'aller au bout du traitement demandé. Examinons le contenu du fichier produit :

```
function hello(person) {
  return "Hello, " + person;
}
document.body.innerHTML = greeter("world");
```

Le contenu est sensiblement identique à **hello.ts**, or une ligne blanche a été supprimée. Et cela est normal, car

nous n'avons pas encore fait appel aux améliorations syntaxiques apportées par TypeScript. Nous avons simplement placé du JavaScript classique dans un fichier **.ts**, ce qui est tout à fait acceptable, étant donné qu'un fichier JavaScript valide est aussi un fichier TypeScript valide.

Je ne m'étendrais pas ici sur les différentes options offertes par **tsc**. J'en relèverai seulement deux qui feront le bonheur des développeurs web. La première **-w** ou **--watch** permet de faire fonctionner **tsc** en mode surveillance. Il détectera automatiquement toute modification de votre fichier source et produira le fichier de destination sans attendre. Après avoir jeté un œil sur la fenêtre correspondante pour vous assurer qu'aucune erreur ne s'est produite, vous pourrez recharger la page dans votre navigateur web pour des tests plus approfondis. Ainsi, l'étape de compilation n'en est plus vraiment une. La seconde option est **--pretty** qui permet de générer un affichage plus coloré et plus explicite, rendant les erreurs bien plus évidentes à voir et à comprendre d'emblée. Comparez l'affichage de la compilation précédente avec celle-ci :

```
$ tsc --pretty hello.ts

5 document.body.innerHTML = greeter("world");
'~~~~~~~'

hello.ts(5,27): 'error' TS2304: Cannot find name 'greeter'.
```

--pretty est donc l'option à utiliser conjointement avec **--watch** pour éviter qu'une erreur passe inaperçue.

2 Le langage

Nous n'avons pas l'espace nécessaire ici pour explorer TypeScript de manière exhaustive. Comme pour tout langage qui se respecte, un livre entier n'y suffirait pas. Par ailleurs, comme TypeScript étend JavaScript, il serait nécessaire d'explorer d'abord JavaScript pour ne laisser aucune zone d'ombre, ce qui nous amène à deux livres... Bref, nous allons nous contenter d'examiner les apports qui nous ont paru les plus significatifs de ce langage.

2.1 Classes

2.1.1 Définition

L'un des aspects les plus troublants pour les développeurs venus d'autres langages objets découvrant JavaScript est l'absence de classes pour définir les objets. JavaScript est construit selon un autre paradigme, à savoir celui du *prototype*. Ce principe définit un mécanisme d'héritage, mais nettement moins lisible et contraignant que les classes.

TypeScript permet de retrouver nos habitudes en nous autorisant l'écriture de classes et d'interface. Considérons le code suivant :

```
01: class User {
02:  fullName;
03:  constructor(firstName, lastName) {
04:    this.fullName = firstName + " " + lastName;
05:  }
06: }
07:
08: var test = new User('John','Doe');
09:
10: document.body.innerHTML = test.fullName;
```

Nous avons défini ici la classe **User**. Celle-ci dispose de certaines propriétés, dont la première est **fullName**. Les propriétés déclarées ainsi (leurs noms sur une ligne) sont publiques par défaut. Puis nous avons écrit la fonction **constructor**, couramment nommée le *constructeur*, qui est appelée à la création de l'objet et permet son initialisation. Ligne 8, nous créons un nouvel objet **User**, dont nous utilisons la propriété **fullName** comme seul contenu de notre document. Tout cela est pour le moins classique en programmation objet, mais voyons ce que cela donne compilé en JavaScript :

```
var User = (function () {
  function User(firstName, lastName) {
    this.fullName = firstName + " " + lastName;
  }
  return User;
}());
var test = new User('John', 'Doe');
document.body.innerHTML = test.fullName;
```

La structure est bien différente et nettement moins lisible. Le nombre de parenthèses, d'accolades et de fonctions imbriquées parait déjà bien complexe pour un code aussi simple...

Notons que les propriétés peuvent être optionnelles.

2.1.2 Visibilité des propriétés

Un des aspects les plus classiques de la programmation objet est la possibilité de limiter la visibilité des propriétés.

Pour cela, nous retrouvons les mots-clés habituels de **private**, **protected** et **public**.

Notons au passage que TypeScript propose un raccourci : les arguments passés au constructeur peuvent devenir des propriétés, à condition de préciser leur portée. Modifions le code précédent pour illustrer cela :

```
class User {
public fullName;
constructor(protected firstName, protected lastName) {
this.fullName = firstName + " " + lastName;
}
}

var test = new User('John','Doe');

document.body.innerHTML = test.firstName;
```

Ici, nous avons défini la propriété **fullname** comme étant publique, mais aussi **firstName** et **lastName** comme étant protégées, et ce dans le constructeur. À la dernière ligne, j'ai volontairement effectué un accès non autorisé à une propriété protégée. Pourtant la compilation n'échoue pas : un simple avertissement est levé.

```
hello.ts(10,27): error TS2445: Property 'firstName' is protected
and only accessible within class 'User' and its subclasses.
```

Le code JavaScript s'exécute quant à lui sans problème, la propriété protégée est lue sans difficulté. Vous pourriez trouver cela surprenant, mais cela correspond bien à la philosophie adoptée par TypeScript : les contraintes apportées par le langage sont faibles, dans le sens où elles n'empêchent pas la compilation du code ni son exécution, mais elles provoqueront des messages d'erreur lors de la compilation. Charge à vous de bien lire ces messages et de corriger les erreurs correspondantes de façon à obtenir un code propre, plus facile à maintenir et à faire évoluer.

Voyons ce que donne la compilation de notre nouveau code, mais uniquement le constructeur, seul élément modifié :

```
function User(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.fullName = firstName + " " + lastName;
}
```

Le code n'est pas très différent du précédent. Nous voyons simplement que deux lignes sont apparues, qui permettent aux arguments reçus d'être utilisés comme propriétés. Rien dans le JavaScript ne donne d'indication quant à la lisibilité du code et que cette question relève uniquement de la compilation.

2.1.3 Héritage

Nous l'avons dit plus haut, JavaScript propose un mécanisme d'héritage s'appuyant sur le concept de prototype. TypeScript nous permet de reprendre nos habitudes en définissant un mécanisme plus classique d'extension de classe. Nous utiliserons pour cela le mot-clé **extends**. Modifions légèrement les dernières lignes de notre code précédant de façon à simplement définir une classe **Moderator** qui étend **User** :

```
class User {
  public fullName;
  constructor(protected firstName, protected lastName) {
    this.fullName = firstName + " " + lastName;
  }
}

class Moderator extends User {}

var test = new Moderator('John','Doe');

document.body.innerHTML = test.fullName;
```

Pour la dernière fois, je vous indique le code JavaScript correspondant, pour que vous preniez pleinement conscience du point auquel TypeScript simplifie le travail du développeur :

```
var __extends = (this && this.__extends) || function (d, b) {
  for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
  function __() { this.constructor = d; }
  d.prototype = b === null ? Object.create(b) : (__.prototype =
b.prototype, new __());
};
var User = (function () {
  function User(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = firstName + " " + lastName;
  }
  return User;
}());
var Moderator = (function (_super) {
  __extends(Moderator, _super);
  function Moderator() {
    _super.apply(this, arguments);
  }
  return Moderator;
}(User));
var test = new Moderator('John', 'Doe');
document.body.innerHTML = test.fullName;
```

Cela est assez convaincant sur ces quelques lignes de code triviales. Imaginez ce qu'il peut en être pour un projet complexe : cela finit par ressembler à de la magie noire...

Il est naturellement possible d'ajouter de nouvelles méthodes à la classe enfant ou d'écraser les méthodes existantes. Il peut dans ce cas être utile de pouvoir appeler les méthodes de la classe parente : cela sera réalisé à l'aide du mot-clé **super** qui peut être utilisé comme une simple fonction pour appeler le constructeur du parent, ou comme un objet porteur pour accéder à une méthode. Nous rendons pour le coup la classe **User** plus complexe en lui ajoutant une méthode **hello** qui prendra un titre comme paramètre, méthode que nous allons écraser et appeler dans la classe enfant.

```
01: class User {
02:  public fullName;
03:  constructor(protected firstName, protected lastName) {
04:    this.fullName = firstName + " " + lastName;
05 : }
06:  hello(title) {
07:    return "Hello " + title + " " + this.fullName;
08:  }
09: }
10:
11: class Moderator extends User {
12:  constructor(firstName, lastName) {
13:    super(firstName,lastName);
14:  }
15:  hello(){
16:    return super.hello("Moderator")`;
17:  }
18: }
```

Remarquez aux lignes 13 et 16 l'emploi de l'instruction **super**.

2.2 Les indications de type

Les indications de type ne sont pas spécifiques aux classes ou aux fonctions, mais elles se révèlent particulièrement utiles dans ce contexte. TypeScript vous permet de définir des types attendus pour les arguments de vos fonctions, leur valeur de retour, ou même pour n'importe quelle variable. Cette possibilité est particulièrement pertinente afin de vérifier la cohérence de votre code.

Rappelons que JavaScript utilise des types pour ses variables, mais il ne nécessite pas de les déclarer au préalable et effectue des conversions automatiques selon leur contexte d'utilisation. Cela se révèle parfois source de problèmes difficiles à démêler.

L'indication des types permet à TypeScript d'en éviter un grand nombre en amont. Pour le type d'une variable, il suffit de faire suivre son nom lors de sa déclaration par un double-point **:** suivi du type de variable attendue, **string** par exemple :

```
class User {
  public fullName : string;
  constructor(protected firstName : string, protected lastName :
string) {
    this.fullName = firstName + " " + lastName;
  }
  hello(title : string) {
    return "Hello " + title + " " + this.fullName;
  }
}

var test : User = new User('John','Doe');

document.body.innerHTML = test.hello("Mister");
```

Si vous examinez le code JavaScript produit avant et après l'indication des types, vous ne constaterez aucune différence. Et même le compilateur restera muet sur le sujet, à moins que vous ne lui ajoutiez l'option **--noImplicitAny**, qui me paraît indispensable. Par ailleurs, les indications de type pourront être utilisées par votre IDE s'il supporte la syntaxe TypeScript **[3]** : infobulles et autres aides contextuelles vous permettront de développer plus vite sans erreur.

Alors, pourquoi s'embêter ? Parce que cette fonctionnalité de documentation est absolument essentielle pour une meilleure compréhension du code, et donc pour le faire évoluer mieux. Notez que vous n'êtes pas limité aux types de base (**boolean**, **number**, **string**, **array**, **tuple**, **enum**, **any** et **void**), mais toute nouvelle classe peut être utilisée comme indication de type, ainsi que nous l'avons fait sur l'avant-dernière ligne du listing précédent.

2.3 Interfaces

Un autre concept de la programmation objet qui manque à JavaScript et que TypeScript va lui apporter est celui d'interface. Pour rappel, une interface permet de définir des obligations que doit remplir une classe qui l'implémente, en terme de propriétés et de méthodes requises. L'interface permet ainsi de définir les moyens de dialoguer avec toutes les classes qui l'implémentent sans faire de présupposé sur leur fonctionnement interne. L'utilité des interfaces étant méthodologique, leur utilisation n'aura pas d'effet par lui-même sur le code JavaScript compilé, mais la qualité du code que vous écrirez en sera grandement améliorée.

GNU LINUX MAGAZINE / FRANCE

BACK TO ROOT !

n° 200

À partir du **23 décembre**, GNU/Linux Magazine revient aux sources. Découvrez sa

Nouvelle formule !

- L'embarqué et la programmation bas niveau sont de retour
- Du dev sans contraintes avec les hacks & bidouilles
- Et toujours ce mélange de système, développement et algo

Les nouvelles rubriques de votre magazine :

- Actualités & Humeur
- IA, Robotique & Sciences
- Système & Réseau
- IoT & Embarqué
- Kernel & Bas niveau
- Hacks & Bidouilles
- Libs & modules
- Mobile & Web
- Sécurité & Vulnérabilité

TOUCHEZ À TOUS LES DOMAINES DE L'INFORMATIQUE SUR SYSTÈMES OPEN SOURCE

Une interface se déclare à l'aide du mot-clé **interface**. Sa syntaxe est proche de celle d'une classe, les propriétés suivant sa déclaration entre accolades. Les méthodes sont plus complexes à indiquer, nous verrons un peu plus loin comment. La classe qui l'implémente le fait à l'aide du mot-clé **implements** :

```
interface UserInt {
  fullName: string;
}

class User implements UserInt {
  public fullName: string;
[...]
}
```

TypeScript offre la possibilité de créer des interfaces pour des fonctions (qui, vous vous en souviendrez, sont également des objets en JavaScript !). La syntaxe est un peu différente, dans la mesure où notre interface ne définira qu'une seule propriété anonyme qui indiquera entre parenthèses la liste des arguments accompagnés de leur type, suivi du type de la valeur de retour :

```
interface helloInt {
  (title: string) : string;
}

function hello(title: string) {
  return "Hello "+title;
}
```

Comme une classe, une interface définit un nouveau type. Dès lors, il est possible d'indiquer qu'une méthode, définie par une autre interface, est requise :

```
interface helloInt {
  (title: string) : string;
}
interface UserInt {
  fullName: string;
  hello: helloInt;
}

class User implements UserInt {
  public fullName: string;
  hello(title: string) {
    return "Hello " + title + " " + this.fullName;
  }
}
```

2.4 Fonctions

Concernant les fonctions également, TypeScript apporte son lot d'améliorations !

2.4.1 Indications de types sur les fonctions anonymes

Nous avons vu plus haut comment ajouter des indications de type aux fonctions et aux méthodes. Cela est également possible avec les fonctions anonymes, mais en suivant une syntaxe un peu différente :

```
let helloFunc: (who: string)=>string = function(who: string):
string { return "Hello "+who;}
```

Pour le moment, laissons de côté **let**, instruction TypeScript utilisée pour une déclaration de variable améliorée. Nous définissons donc **helloFunc**, variable porteuse d'une fonction anonyme. Nous indiquons le type de cette variable, mais comme il s'agit d'une fonction, celui-ci est plus complexe. Entre parenthèses, les arguments désignés par leurs noms suivis de leur type. Puis le signe **=>** qui permet d'indiquer le type de la valeur retournée. Voilà, après le signe égal nous pouvons passer à l'écriture de la fonction proprement dite. Là aussi nous avons indiqué le nom des arguments et leur type entre parenthèses, suivi, mais après deux points cette fois-ci, du type de la valeur de retour. Vous trouvez cela un peu redondant ? Vous avez raison. Aussi les développeurs de TypeScript ont-ils prévu un mécanisme d'inférence permettant au compilateur de déduire le type de la fonction depuis celui de la variable la contenant et inversement. Aussi les deux codes suivants sont-ils équivalents au premier :

```
let helloFunc: (who: string)=>string = function(who) { return
"Hello "+who;}
let helloFunc = function(who: string): string { return "Hello "
+who;}
```

C'est tout de même mieux. À vous de choisir quelle version vous préférez, et de vous y tenir afin d'être consistant.

2.4.2 Valeurs par défaut et paramètres optionnels

Lors de la compilation, TypeScript vérifie que les appels de fonctions s'effectuent avec le nombre exact d'arguments attendus. Il est donc utile de pouvoir signaler certains d'entre

eux comme étant optionnels, ainsi que de leur donner une valeur par défaut.

Un point d'interrogation à la suite de son nom suffit pour qu'il devienne optionnel.

```
function helloFunc2(who: string, title? :string): string {
  return "Hello " + title + who;
}
```

Un paramètre ayant une valeur par défaut étant évidemment optionnel, le point d'interrogation n'est plus nécessaire. Le nom est alors suivi d'un signe égal, puis de la valeur à utiliser si l'argument est manquant :

```
function helloFunc2(who: string, title = "Monsieur"): string {
  return "Hello " + title + who;
}
```

Dans ce cas, nous ne précisons pas le type du dernier argument, celui-ci est déduit par TypeScript de sa valeur par défaut.

2.4.3 Paramètres surnuméraires

Il est parfois utile de pouvoir écrire une fonction acceptant un nombre indéterminé d'arguments. TypeScript permet de les recevoir dans un tableau. Imaginons une fonction de concaténation :

```
function myConcat(...txt: string[]) {
  return txt.join("");
}
```

Les trois points **...** indiquent que la variable **txt** va recevoir tous les arguments suivants. Ici, il n'y en a pas de précédent, ce qui simplifie la tâche. En indiquant le type de **txt**, nous précisons qu'il s'agit de chaînes de caractères, mais stockées dans un tableau à l'aide des crochets **[]**.

2.5 Déclaration de variables

Terminons sur un point moins appétissant de prime abord, car basique, mais qui a son importance : la déclaration de variables. Traditionnellement, JavaScript effectue cette opération grâce à l'instruction **var**. TypeScript propose de lui ajouter **let** pour corriger certaines difficultés posées par celle-ci.

Pour commencer, il est parfaitement autorisé en JavaScript de déclarer plusieurs fois la même variable, sa valeur sera simplement modifiée si une nouvelle attribution a lieu :

```
var maVar = "GNU/Linux Magazine France";
var maVar = "Linux Pratique";
```

Ce comportement n'est pas forcément souhaité et peut être la cause de bogues difficiles à démêler, car il est possible d'écraser involontairement la valeur d'une variable. Avec **let**, une erreur sera lancée :

```
let maVar = "GNU/Linux Magazine France";
let maVar = "Linux Pratique"; // Erreur: "Cannot redeclare block-
scoped variable 'maVar'"
```

let permet également d'intervenir sur les portées des variables. Le fonctionnement par défaut de JavaScript est assez complexe et particulier, voire exotique selon les langages que vous avez fréquentés auparavant. Cela peut être source de comportements inattendus, voire de bogues difficiles à résoudre. Nous n'avons pas l'espace nécessaire pour rentrer dans des explications détaillées, aussi nous contenterons-nous d'un exemple particulièrement frappant :

```
for (var i = 0; i < 10; i++) {
  setTimeout(function() {console.log(i); }, 100 * i);
}
```

Tout le monde s'attend à voir les nombres de **1** à **10** s'afficher progressivement sur la console JavaScript. Mais le résultat n'est pas celui-là : si l'échelonnement dans le temps est correct, la console affichera dix fois le nombre dix. La solution classique en pur JavaScript consiste à encapsuler la déclaration du *Timeout* dans une autre fonction qui recevra la valeur à utiliser en paramètre :

```
for (var i = 0; i < 10; i++) {
  (function(i){
    setTimeout(function() {console.log(i); }, 100 * i);
  })(i)
}
```

La solution en TypeScript est beaucoup plus élégante :

```
for (let i = 0; i < 10 ; i++) {
  setTimeout(function() {console.log(i); }, 100 * i);
}
```

3 Fichier de configuration

Voilà, après ce rapide tour d'horizon, vous êtes prêts à faire vos premiers pas avec TypeScript. Un élément vous manque encore pour pouvoir travailler efficacement, à savoir la possibilité de compiler l'ensemble d'un projet. En effet, au début de cet article, nous n'avons vu que la commande permettant de compiler un unique fichier, ce qui est évidemment insuffisant. TypeScript vous permet heureusement d'écrire un fichier de configuration que le compilateur utilisera à chaque appel, à moins que vous ne lui indiquiez précisément le fichier **.ts** à compiler.

Le fichier doit se trouver dans le dossier racine de votre projet et s'appeler **tsconfig.json**. Sans surprise, le format à utiliser est JSON. Pour l'utiliser, soit vous invoquez la commande **tsc** sans autre précision en vous étant préalablement placé dans ce dossier, soit vous ajoutez le paramètre **-p** suivi du chemin vers ce même dossier.

En voici un bref exemple :

```
{
  "compilerOptions": {
    "module": "commonjs",
    "noImplicitAny": true,
    "removeComments": true,
    "outFile": "../../built/local/tsc.js"
  },
  "files": [
    "core.ts",
    "program.ts",
    "tsc.ts"
  ]
}
```

Remarquons quelques options intéressantes : **noImplicitAny** qui permet de lancer un avertissement dès qu'un type n'est pas précisé, **removeComments** qui permet une première optimisation du code produit en supprimant les commentaires, **outFile** qui permet de définir un fichier unique de destination contenant tout le JavaScript produit (ce qui sera toujours plus propre qu'une longue suite de fichiers JavaScript à télécharger par le navigateur). Notons enfin la directive **files** qui recense l'ensemble des fichiers à compiler. Vous pouvez également utiliser, mais pas en même temps, une directive contraire, **exclude** qui contiendra la liste des fichiers et des dossiers à exclure de la compilation.

Voilà de quoi travailler à grande échelle !

Conclusion

Comme annoncé, nous n'avons pu réaliser ici qu'un survol des principales fonctionnalités apportées par TypeScript. Il reste bien des notions à approfondir ou à découvrir : classes abstraites, membres statiques, espaces de noms, modules, classes génériques, JSX... et bien d'autres choses encore !

Toujours est-il que, si vous avez déjà fréquenté JavaScript, cette première approche vous aura sans aucun doute séduit, et vous aurez à cœur d'ajouter TypeScript à vos outils quotidiens. Si tel est le cas, prenez le temps de réfléchir à l'intégration de cet outil dans votre flux de travail, en particulier en équipe. Sans quoi, il pourrait arriver qu'un de vos collaborateurs intervienne directement dans les fichiers JavaScript comme il avait l'habitude de le faire jusque-là... ce qui ne manquerait pas de poser quelques problèmes. ■

Références

[1] TypeScript : https://www.typescriptlang.org

[2] Node.js : https://nodejs.org

[3] Hors Visual Studio et d'autres IDE privateurs, il existe des extensions pour Emacs (https://github.com/ananthakumaran/tide), Eclipse (https://github.com/palantir/eclipse-typescript), Atom (https://atom.io/packages/atom-typescript) et même Vim (https://github.com/Microsoft/TypeScript/wiki/TypeScript-Editor-Support#vim).

Pour aller plus loin

Cet article axé sur quelques-uns des aspects fondamentaux du langage ne rend pas justice à sa puissance. Pour la voir en action, je vous recommande d'examiner quelques exemples présentés à la page https://www.typescriptlang.org/samples/index.html : vous y trouverez, entre autres, une implémentation d'imagerie 3D par lancé de rayons (un *raytracer*) ou encore des exemples d'applications intégrant TypeScript avec d'autres technologies telles que **Backbone.js**, **jQuery**, **Node.js** ou **MongoDB**.

HANDICAP INTERNATIONAL

Le Sac à Sapin,

une touche de magie pour votre sapin de Noël !

1€50 en faveur des personnes handicapées

Un produit eco-responsable : 100% biodégradable

Pratique et Décoratif

Être libre et indépendant des géants du web

LINAGORA

Les logiciels libres

www.linagora.fr