Developpez Le Mag

Édition de février – mars 2016
Numéro 62
Magazine en ligne gratuit
Diffusion de copies conformes à l'original autorisée
Réalisation : Alexandre Pottiez
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Sommaire

Éditorial

La rédaction se plie une fois encore en quatre pour vous permettre de lire le meilleur des publications dans toutes vos technologies préférées. Profitez-en bien !

La rédaction

Article ALM

Git - Démarrage rapide sous Linux

Ce tutoriel est destiné aussi bien au développeur solitaire qui désire bénéficier rapidement d'un système de gestion de versions simple et efficace, qu'à une équipe voulant mettre en place rapidement un système de gestion de versions collaboratif.
par Loïc Guibert
Page 11

Article Cloud Computing

Découvrir et apprendre la solution de cloud computing : Click & Cloud

Dans ce tutoriel, vous allez découvrir et apprendre cette solution très intéressante et facile à prendre en main.
par Guillaume Sigui
Page 53

C++

Les derniers tutoriels et articles

Préconditions en C++
Partie II

Dans ce billet, je continuerai à partager mes réflexions sur les préconditions. Il traitera un peu de la philosophie qui est derrière le concept des préconditions (et des bugs), et étudiera la possibilité de mettre à profit le compilateur pour vérifier certaines préconditions. Plusieurs lecteurs ont fourni des retours utiles dans mon précédent billet (lien 1), j'essaierai de les incorporer à celui-ci.

1 En quoi les préconditions peuvent-elles (ou pas) nous aider ?

Tout d'abord, je souhaite partager une observation, qui vous semble peut-être évidente, mais qui est récente pour moi : les préconditions (et les autres spécifications des contrats) n'empêchent pas (ni ne traitent) tous les types de bugs. Elles répondent seulement aux problèmes d'incompréhension ou de spécification insuffisante des interfaces.

Lorsque je tente de me rappeler les bugs que j'ai écrits dans mon code ces dernières années, je vois qu'ils consistaient souvent à utiliser une expression booléenne incorrecte :

```
1  if (!isConditionA || isConditionB) {
2    relyOnConditionB();
3  }
```

Alors que je souhaitais en réalité exprimer :

```
1  if (!isConditionA && isConditionB)
```

J'aurais aussi pu spécifier une expression booléenne erronée dans un test de précondition. On croit facilement à tort que les fonctions que nous écrivons peuvent contenir des bugs (que nous devons vérifier), mais que les contrôles que nous écrivons sont toujours corrects. La même chose s'applique, en fait, aux tests unitaires et aux autres types de tests.

Les préconditions comblent les manques dans les spécifications de l'interface. Par exemple, si vous traitez les fonctions du code comme des fonctions mathématiques, les préconditions aident à restreindre le domaine des arguments de fonction. Si la fonction f accepte n'importe quel entier (qui soit suffisamment petit), vous la déclarez ainsi :

```
1  void f(int i);
```

À présent, si une fonction f1 n'est correctement définie que pour des entiers non négatifs, vous la déclarez comme suit :

```
1  void f1(unsigned int i);
```

C'est-à-dire que vous utilisez le système de typage et ses fonctionnalités de sécurité pour garantir, à la compilation, les contraintes sur le domaine de la fonction. Mais si une fonction f2 n'est bien définie que pour des arguments entiers supérieurs à 2, que faites-vous ? Vous spécifiez un nouveau type capable de contenir uniquement des entiers supérieurs à 2 ? Cette solution est possible, mais moins aisée que la simple utilisation d'un type natif. Étant donné les difficultés qu'implique l'implémentation d'un nouveau type restreint, spécifier une précondition est une solution alternative attirante.

Les préconditions présentent des inconvénients évidents, comparées aux contrôles de type : la validité ne peut pas être garantie aussi aisément à la compilation. Les vérifications supplémentaires à l'exécution semblent n'être qu'un piètre contournement. Par conséquent, idéalement, on ne devrait les utiliser que pour remplacer les vérifications de type lorsque ces dernières ne sont pas possibles. Les préconditions offrent peu de garanties concernant la validité, mais elles en offrent quand même ; elles peuvent par exemple tenir lieu d'indications utiles pour les analyseurs statiques. Nous tenterons d'aborder ceci dans la prochaine partie.

Même sans l'assistance d'outils, les préconditions sont présentes et visibles des programmeurs. Elles aident à rédiger et à maintenir le code. Par exemple, supposons que vous soyez en train de corriger un bug dans la fonction suivante :

```
double ratio(int nom, int den)
{
  if (den != 0) {
    return double(nom) / den;
  }
  else {
    return 0;
  }
}
```

Vous constatez que le return 0 est suspect. Vous voulez donc le remplacer par une instruction qui déclenche une exception. Mais vous vous posez une question : « Vais-je casser le code d'autres personnes, s'il s'appuie sur le fait que notre fonction renvoie 0 quand den vaut 0 ? » Si cette fonction spécifiait une précondition qui exige que den != 0, vous avez la réponse : même si certaines personnes s'appuient sur le comportement actuel, elles le font de façon illégale : on ne doit faire aucune présupposition sur le comportement de la fonction dont la précondition n'a pas été respectée — du moins selon la définition de « précondition » que j'essaie de promouvoir ici.

Au vu de la description ci-dessus, spécifier des préconditions et spécifier des axiomes de concepts poursuivent des objectifs très proches : lien 2.

2 La définition d'une précondition

En général, les gens s'accordent sur le fait que « l'appelant *devrait* satisfaire la précondition d'une fonction avant d'appeler cette fonction ». Le point de désaccord et d'incompréhension est ce « devrait ». La fonction peut-elle considérer comme acquis que sa précondition est respectée ? Ou bien devrait-elle faire tout le contraire : supposer que la précondition n'est pas respectée, et tenter de le prouver (avant de faire quoi que ce soit d'autre) ?

Résoudre l'ambiguïté sur le « devrait » est très important. Sans cela, nous revenons au problème par lequel nous démarrions le billet précédent (lien 3) : celui avec la fonction checkIfUserExists. Si un développeur attend que la fonction vérifie toujours sa précondition et que l'autre suppose qu'il peut compter sur l'appelant pour vérifier cette même précondition, nous avons à nouveau le bug — bien que nous spécifiions diligemment les préconditions !

Fait intéressant, cette question n'a pas de bonne réponse. D'une part, imaginez qu'à chaque appel, operator[] de std : :vector vérifie que l'indice est compris dans le bon intervalle. Et dans du code en production ? Cela ruinerait les performances. Et l'une des principales raisons de choisir C++ est la performance. C'est précisément pourquoi l'opérateur spécifie la précondition : pour ne pas avoir à la vérifier lui-même.

D'un autre côté, dans les fonctions où la performance n'est pas critique, on serait mal avisé de ne pas vérifier la violation d'une précondition, surtout en sachant que cela pourrait causer de sérieux dommages. Cela dit, même pour operator[] de std : :vector, la performance n'est pas non plus critique en toutes circonstances, notamment dans des versions de test ou de débuggage.

J'ai déjà apporté ma définition de « devrait » dans le précédent billet, qui tente de traiter les deux attentes contraires. Appeler une fonction dont la précondition n'est pas satisfaite résulte en un comportement non défini(*UB - undefined behavior*). Cela signifie que l'appelant n'est pas autorisé à faire de présuppositions quant au résultat de l'appel de fonction, et que la fonction appelée est autorisée à faire *tout* ce qui lui semble approprié. Ce « tout » permet à l'auteur de la fonction de faire les choses suivantes :

1. Optimiser, en partant du principe que la précondition est systématiquement respectée.
2. Vérifier la précondition et signaler sa violation par tous les moyens (appeler std : :terminate, déclencher une exception, lancer un debugger, etc.).
3. Choisir entre 1 ou 2 en fonction d'autres facteurs (comme la valeur de la macro NDEBUG).

« L'appelant ne peut rien présupposer » — ceci est aussi très important : même si notre fonction déclenche (pour l'instant) une exception en cas d'échec de la précondition, l'appelant ne peut pas s'appuyer sur cela pour choisir son chemin d'exécution, selon qu'il attrape ou non l'exception. Cela signifie que l'implémentation suivante de la fonction AbsoluteValue est invalide :

```
double sqrt(double x)
// précondition: x >= 0
{
  if (x < 0) {
    throw std::domain_error{"nombre négatif passé à sqrt"}; // OK
  }
  else {
    // calculer...
  }
}

double AbsoluteValue(double x)
// précondition: true
try {
  sqrt(x);
  return x;
}
catch(std::domain_error const&) {
  return -x;
}
```

Ceci est dû au fait que AbsoluteValue repose sur un *UB*. Ou, exprimé plus formellement, si la fonction AbsoluteValue ci-dessus est appelée avec un x

négatif, le comportement est non défini (un *UB* peut toutefois aussi se solder par l'obtention du résultat attendu).

3 Prévenir les échecs des préconditions

La seule façon correcte de traiter les échecs de préconditions (ou, en général, les bugs) est de les éliminer. Bien entendu, cela n'est pas toujours possible et il faut prendre des mesures pour répondre aux bugs à l'exécution. Cette démarche n'est pas idéale, mais elle s'avère nécessaire. Mais vous savez comment faire et nous ne discuterons pas ceci dans ce billet. Nous nous concentrerons plutôt sur la prévention des bugs. Nous avons déjà indiqué la méthode en montrant comment unsigned int peut remplacer int. Ce n'était pas le meilleur exemple : la conversion implicite d'un int vers unsigned int risque de réduire nos efforts à néant, nous allons donc chercher des moyens plus robustes.

Dans leurs commentaires sur mon précédent billet, quelques lecteurs ont discuté sur le fait que les exemples auraient été « sécurisés » si l'on y avait utilisé des types plus fortement contraints :

```
1  bool checkIfUserExists(Name userName);
2  // Name ne contient jamais ni espaces ni
       ponctuation
3
4  NonNegative sqrt(NonNegative x);
```

Le seul nom du type donne un message clair aux appelants : « Ne me passez pas n'importe quelle chaîne de caractères ». Le compilateur émettra un avertissement si nous passons simplement un objet std : :string à checkIfUserExists. Ou peut-être pas ? Voyons ce à quoi la définition du type Name pourrait ressembler. En voici une implémentation possible :

```
1  class Name
2  {
3  public:
4    explicit Name(std::string const& r);
5    // précondition: isValidName(r)
6
7    explicit Name(std::string && r);
8    // précondition: isValidName(r)
9
10   std::string const& get() const;
11
12   operator std::string const&() const;
13 };
```

Il y a trois choses à noter. Premièrement, le constructeur est explicit : nous voulons que l'appelant mentionne explicitement le type Name. C'est comme exiger de l'appelant qu'il appose sa signature : « Oui, je m'engage à passer une chaîne convenable pour un Name ». Deuxièmement, la conversion de Name vers std : :string est implicite. Cela est sécurisé, car n'importe quel Name est aussi un std : :string valide. En convertissant de cette façon, les contraintes sont relaxées. Troisièmement, il nous faut spécifier la précondition ! Nous ne sommes pas débarrassés de la précondition initiale — nous l'avons déplacée ailleurs. C'est ainsi que nous pouvons utiliser notre type.

```
1  bool authenticate()
2  {
3    std::string userName = UI::
         readUserInput();
4    return checkIfUserExists(userName); //
         erreur à la compilation
5  }
```

Cela résout-il le problème ? Cela prévient en effet les erreurs où l'appelant croirait naïvement que userName est un nom d'utilisateur valide. L'appelant peut toujours écrire :

```
1  bool authenticate()
2  {
3    std::string userName = UI::
         readUserInput();
4    return checkIfUserExists(Name{userName
         }); // je sais ce que je fais !
5  }
```

Si userName a un contenu nuisible, on obtient encore le même résultat inattendu. Mais, au moins, la nécessité de typer NameuserName a plus de chances de faire réfléchir l'appelant pendant une seconde.

Au cas où l'on aurait choisi que c'était au constructeur de Name de vérifier la précondition à l'exécution, et d'indiquer les violations comme bon lui semble, l'erreur aurait été signalée avant que la fonction checkIfUserExists ne soit exécutée. C'est une caractéristique très avantageuse, car elle indique clairement à tous les outils (comme les debuggers et générateurs de *dumps*) que le bug n'est pas dans la fonction à appeler, mais dans l'appelant.

Le réel bénéfice de cette méthode est visible si nous pouvons aussi faire en sorte que la fonction UI : :readUserInput renvoie un Name :

```
1  bool authenticate()
2  {
3    Name userName = UI::readUserInput();
4    return checkIfUserExists(userName);
5  }
```

On introduit ainsi dans notre programme une abstraction généralement utile : Name, qui représente une séquence (typiquement courte) de chiffres et de lettres, sans espaces ni signes de ponctuation. La précision « généralement utile » est importante, car nous pouvons aussi l'utiliser dans d'autres parties de l'application pour représenter d'autres choses que des noms d'utilisateurs : des identifiants, des codes courts... La précision « typiquement courte » est également utile, car nous pouvons appliquer certaines optimisations à notre classe : par exemple, stocker la chaîne entière à l'intérieur de l'objet, et

ne recourir au *tas* que dans des situations exceptionnelles. C'est ici que les allocateurs de pile peuvent aider : lien 4. De façon similaire, le type NonNegative représente également une abstraction généralement utile, en particulier du fait qu'avec les littéraux définis par l'utilisateur, nous pouvons en plus garantir qu'un littéral négatif ne sera jamais autorisé :

```
1  NonNegative sqrt(NonNegative x);
2
3  NonNegative y = sqrt(2.25_nn);  // "_nn"
       indique un littéral NonNegative
4  NonNegative x = sqrt(-2.25_nn); //
       ERREUR: littéral négatif
```

Pour voir comment mettre en œuvre cette astuce, voir ici : lien 5. Toutefois, toute précondition ne donne pas nécessairement lieu à un type contraint généralement utile. Imaginez le cas suivant :

```
1  int fun(int i);
2  // précondition: i > 3
```

Bien sûr, il y a moyen de construire un type contraint à partir de cette précondition. Il existe même une bibliothèque qui le facilite : Constrained Value (lien 6). Mais à quelle fréquence avez-vous besoin du type IntGreaterThan3 ? Si vous introduisez un tel type pour les besoins de la précondition d'une seule fonction, le coût pourrait dépasser le bénéfice. Introduire un nouveau type a un coût. Définir de nouveaux types n'est pas trivial en C++. Il vous faut prévoir si votre type sera copiable, déplaçable, comment vous permettrez sa construction, comment il gérera les violations de préconditions… En introduisant un nouveau type, vous risquez donc d'introduire de nouveaux bugs. De plus, le type supplémentaire accroît le temps de compilation et les besoins en mémoire de votre EDI : il lui faut désormais reconnaître un nouveau type et fournir des indications à son sujet. On ne réfléchit jamais ainsi, car on ajoute généralement un type pour résoudre un problème, ou réduire la complexité du programme. Cependant, dans notre cas, on risque au contraire d'augmenter la complexité du programme, uniquement pour détecter des situations qui ne se produiront de toute façon jamais.

Plus haut, nous n'avons montré qu'une implémentation possible du type Name. D'autres possibilités existent. Par exemple, considérez celle-ci, qui ne nécessite presque pas de spécification de préconditions :

```
1  class Name
2  {
3  private:
4    explicit Name(std::string const& r);
         // précondition: isValidName(r)
5    explicit Name(std::string && r);
         // précondition: isValidName(r)
6    friend optional<Name> makeName(std::
         string && r);
7
8  public:
9    std::string const& get() const;
10   operator std::string const&() const;
11 };
12
13 boost::optional<Name> makeName(std::
       string && s)
14 {
15   if (isValidName(s))
16     return Name{std::move(s)};
17   else
18     return boost::none
19 }
```

Bien que la précondition soit toujours présente, les constructeurs sont privés, et l'exigence que la précondition devrait faire respecter n'est plus le contrat entre le composant utilisateur et le composant auteur. En effet, le composant utilisateur ne pourra jamais appeler directement le constructeur et, par conséquent, ne pourra jamais non plus enfreindre la précondition. Dès lors, notre utilisateur peut seulement coder :

```
1  bool authenticate()
2  {
3    auto userName = makeName(UI::
         readUserInput());
4    if (userName) {
5      return checkIfUserExists(*userName);
6    }
7    else {
8      // RÉAGIR
9    }
10 }
```

Par optional, je me référais à la bibliothèque Boost.Optional : lien 7. S'il vous faut aussi plus de fonctionnalités C++11 (comme le constructeur par déplacement), vous pouvez essayer cette version : lien 8. Ce n'est que pour des raisons de brièveté que j'ai mis ce « RÉAGIR » à l'apparence anodine. En réalité, lorsque le compilateur identifie le problème, la réaction de l'utilisateur dans le code peut être plus invasive que le simple ajout d'une clause if. Elle peut même exiger de modifier la signature de authenticate, car elle n'est pas claire, si nous voulons signaler de la même façon deux choses différentes : (1) que le nom d'utilisateur saisi n'existe pas et (2) qu'il y a un bug dans le code. Vous pouvez aussi ne pas apprécier cette implémentation de Name, car elle ajoute un surcoût à l'exécution qui peut s'avérer inacceptable. Même si vous prouvez, de quelque façon que ce soit (p. ex. , par une analyse statique), que des chaînes invalides ne seront jamais passées, vous subissez cependant la pénalité de toujours effectuer le contrôle de validité. Dans ce cas particulier (où nous devons accéder à la base de données), cela ne sera pas notre principal problème, mais, en général, vous ignorez si vous pouvez vous le permettre, et même s'il est possible de vérifier la précondition. Cette technique ne peut donc pas être généralisée à toute précondition.

Et remarquez l'autre point notable avec cette solution de l'optional. Nous avons aussi une autre précondition :

```
template <typename T>
T& optional<T>::operator*();
// précondition: bool(*this)
```

Donc, nous avons juste échangé une précondition contre une autre qui, on l'espère, est mieux connue de tous.

Sans parler d'une implémentation en particulier, il y a des problématiques à traiter en général, lorsque l'on utilise des types contraints pour remplacer les préconditions. Tout d'abord, tentons d'imaginer à quoi ressemblerait l'implémentation revue de la fonction checkIfUserExists :

```
bool checkIfUserExists(Name userName)
{
  std::string query = "select count(*)
      from USERS where NAME = \'"
                    + userName.get() + "
                        \';"
  return DB::run_sql<int>(query) > 0;
}
```

Remarquez l'appel à la fonction-membre get. Vous fiez-vous à l'objet userName pour renvoyer une chaîne valide, ou devrions-nous contrôler la précondition, par précaution ?

```
bool checkIfUserExists(Name userName)
// précondition: isValidName(userName.
    get())
```

Ma réponse à ceci est : « Oui, explicitez chaque présupposition ». Mais n'ajoutons-nous pas le type Name pour rien ? La réponse est « non ». Comme on l'a dit plus haut, en utilisant le type Name, nous perturbons la compilation du programme pour les gens qui oublient que les saisies peuvent ne pas être des noms d'utilisateurs valides. Cela donne à penser que certains bugs seront détectés à la compilation et corrigés. Il reste cependant une chance que nous ayons tout de même le bug (c'est pourquoi nous spécifions malgré tout la précondition) ; nous en avons toutefois réduit la probabilité. En outre, introduire de tels types a d'autres bénéfices que seulement garantir le respect des préconditions. Cela explicite mieux nos intentions, rend le code plus lisible et compréhensible, et aide à éviter d'autres sortes de bugs.

```
bool save(Name owner, FilePath path,
    BigText content = {});
```

Le code ci-dessus reflète mieux vos intentions que :

```
bool save(string owner, string path,
    string content = {});
```

Il permet aussi certaines optimisations, et il évite des bugs tels que :

```
save(owner, content);    // arguments
    dans le mauvais ordre

function<bool(string, string)>
    binaryPredicate;
binaryPredicate = save;  // signatures
    identiques, mais par hasard
sort(vec.begin(), vec.end(), comparator)
    ;
```

Jusqu'ici, nous avons étudié des préconditions qui n'« inspectent » qu'un seul argument. La technique ci-dessus pour introduire des types auxiliaires ne fonctionne pas bien si nous souhaitons contraindre simultanément deux arguments ou plus :

```
double atan2(double y, double x);
// précondition: y != 0 || x != 0
```

De même, nous avons parfois une contrainte sur un seul objet, mais il est difficile d'utiliser un type contraint parce que, par exemple, l'objet est *this. Considérez un exemple où, pour une raison quelconque, vous ne pouvez pas totalement initialiser votre objet dans le constructeur et il vous faut donc une initialisation en deux phases. L'utilisation typique de votre classe se ferait donc comme suit :

```
Machine m;                          // premiè
    re phase
Param p = computeParamFor(m);
m.initialize(p);                  // seconde
    phase
m.run();
```

Typiquement, la fonction run aura une précondition :

```
void Machine::run();
// précondition: this->isInitialized();
```

Techniquement, il est possible d'introduire et d'utiliser un type contraint, mais cela pourrait désorienter les utilisateurs :

```
Machine m;
Param p = computeParamFor(m);
InitializedMachine im{m, p};
im.run();
```

Nous voici à la fin de ce billet. Je n'ai toujours pas réussi à mentionner toutes les choses que je voulais partager avec vous, je suppose qu'il me faudra une troisième partie. Ce sujet s'est avéré plus vaste que je ne m'y attendais.

Une question pour la fin : quel type devrait retourner la fonction sqrt, selon vous ? double ou NonNegative ?

*Retrouvez l'article de **Andrzej Krzemieński** traduit par **kurtcpp** en ligne : lien 9*

Qt

Les derniers tutoriels et articles

Intégration de code OpenGL dans des applications Qt Quick 2

Première partie : organisation du moteur de rendu

Pour bon nombre d'applications dans l'industrie, l'intégration de code OpenGL dans des interfaces Qt Quick 2 est un besoin récurrent – par exemple, pour afficher des contrôles Qt Quick (comme des boutons, des curseurs de défilement) par-dessus une visualisation OpenGL existante.
Cette série d'articles est tirée de la présentation du même nom, donnée par Giuseppe D'Angelo lors du Qt World Summit. Elle a reçu un accueil très chaleureux, ce qui a poussé son auteur à la transformer en une série d'articles.

1 Moteur de rendu de Qt Quick 2

Pour comprendre les modes d'interaction entre du code OpenGL et Qt Quick, la première étape est d'analyser la manière dont Qt Quick 2 affiche une scène. En fait, la machinerie responsable du rendu est assez complexe ; elle utilise déjà OpenGL dans un fil d'exécution distinct pour l'affichage des éléments Qt Quick.

Premièrement, **cela implique l'utilisation d'OpenGL pour l'affichage**. Cette décision de conception a un impact très profond sur Qt Quick et s'appuie sur le fait qu'à peu près tout appareil a aujourd'hui un processeur graphique contrôlable par OpenGL. De plus, l'utilisation d'OpenGL est nécessaire pour effectuer des rendus à raison de soixante images par seconde de manière stable, pour ajouter des effets spéciaux (comme des ombres, des particules, des flous) ou encore pour avoir des animations fluides. Ainsi, tous les éléments visibles dans une scène Qt Quick sont affichés à l'aide de points, de lignes, de triangles et de shaders. Cette décision facilite aussi l'intégration de code OpenGL dans l'étape de rendu Qt Quick.

Deuxièmement, **l'affichage est effectué dans un fil d'exécution distinct**, différent du fil principal, c'est-à-dire celui qui exécute la fonction main() en C++. Cette manière de procéder a deux avantages principaux :

1. Elle permet de laisser le rendu continuer et de ne pas bloquer l'interface (et notamment de garder les animations à soixante images par seconde), même si le processeur est occupé dans des calculs ou si le fil principal est arrêté dans l'appel d'une fonction bloquante ;
2. De même, elle permet de laisser le fil principal se dérouler au cas où le processeur graphique serait trop lent pour l'affichage de la scène.

Ce fil d'exécution réservé à l'affichage est utilisé pour à peu près toutes les plateformes avec Qt 5.6, avec la notable exception d'ANGLE sous Windows. Même si Qt n'est pas encore compatible avec une plateforme donnée, il est probable qu'elle le soit dans une version à venir : prendre en compte ces aspects multifils pour l'intégration de code OpenGL dans le moteur de rendu de Qt Quick 2 est donc très important – au risque de voir ses applications planter lamentablement sans raison apparente.

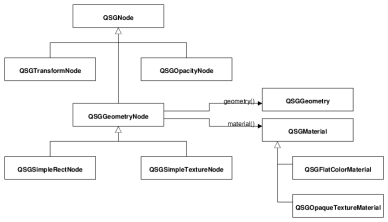
2 Graphe de scène et synchronisation

La vraie question n'est toujours pas posée : comment le moteur effectue-t-il le rendu d'une scène ? En réalité, il utilise une structure de données spéciale, **le graphe de scène**. Ce nom revient assez souvent dans le cas de l'affichage 3D : il fait référence à toute structure de données qui peut être utilisée pour analyser et optimiser certaines tâches (pas seulement l'affichage, mais aussi les simulations

physiques, la recherche de chemins, etc.). Cette série ne s'attardera pas sur la manière d'utiliser ce graphe de scène pour construire de nouveaux éléments Qt Quick, puisque son objectif est de montrer l'intégration de code existant : elle détaillera seulement les parties les plus intéressantes.

Dans le monde de Qt Quick, **le graphe de scène n'est utilisé que pour l'affichage**. Il contient toutes les informations visuelles nécessaires pour dessiner une scène. Ce dernier point est important : toutes les informations non visuelles sont perdues lors du passage au graphe de scène. Par exemple, pour dessiner un bouton rectangulaire, le graphe de scène n'a aucune idée des réactions qu'il peut avoir lors d'un clic ; tout ce qu'il sait, c'est que, pour afficher ce bouton, il doit d'abord dessiner un quadrilatère (en réalité, deux triangles), avec certaines couleurs définies au niveau des arêtes pour simuler un dégradé dans l'arrière-plan ; ensuite, il doit dessiner un quadrilatère plus petit avec une texture (c'est-à-dire une image) qui représente le texte du bouton.

Le graphe de scène Qt Quick est implémenté comme un arbre de nœuds. Chaque type de nœud (une classe qui dérive de QSGNode (lien 10), où SG signifie « graphe de scène », qui se dit *scene graph* en anglais) implémente une fonctionnalité particulière : modifier les transformations des enfants, leur opacité, gérer une forme géométrique (à l'aide de VBO), détenir un shader, etc. La figure suivante montre une série de classes du graphe de scène de Qt Quick.

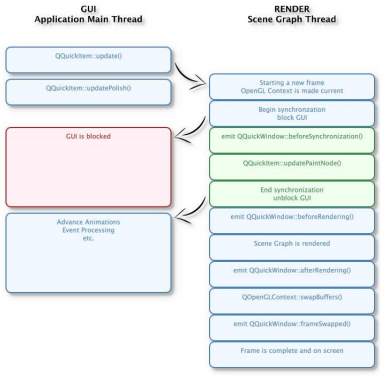


Le graphe de scène est construit en itérant sur les éléments QML. Chaque type d'élément visuel de Qt Quick (Rectangle, Image, Text…) construit un petit arbre de nœuds dans sa version de la fonction QQuickItem : :updatePaintNode et le retourne au moteur de rendu. Ce dernier peut alors prendre tous ces arbres, les analyser et décider de la meilleure manière de les afficher.

Le lecteur attentif aura remarqué une pierre d'achoppement : comment le fil d'exécution du moteur de rendu construit-il, exactement, la structure du graphe de scène à partir des éléments QML, qui appartiennent au fil d'exécution principal ? **Évidemment, avec deux fils d'exécution, la question de la synchronisation s'impose à l'agenda**. Le fil de rendu ne peut pas parcourir l'arbre des éléments QML si le fil principal modifie *en même temps* cet arbre.

À première vue, ajouter des verrous un peu partout devrait résoudre ce problème, mais ils deviennent vite des fardeaux à l'exécution. La solution choisie par Qt Quick est drastique, mais elle enlève ces problèmes de verrouillage de manière explicite : *le fil de rendu et le fil principal se synchronisent*, c'est-à-dire que le moteur de rendu met en pause le fil d'exécution principal, ce qui lui permet d'appeler sans crainte la fonction QQuickItem : :updatePaintNode sur tous les éléments visuels de la scène. Ainsi, cette fonction est bien appelée depuis un autre fil d'exécution. Une fois que le fil de rendu a terminé ses opérations sur les éléments QML, le fil principal est débloqué et peut continuer son exécution. Le fil de rendu a alors récupéré toutes les informations nécessaires pour dessiner la scène à l'écran : il peut maintenant analyser le graphe de scène et l'afficher.

Parfois, une image vaut cent mots : la documentation Qt détaille les interactions entre les deux fils et leur synchronisation avec le diagramme suivant.



Tout ce processus se produit chaque fois qu'une nouvelle image est demandée – par exemple, parce qu'un élément visuel a changé dans la scène ou parce que la fonction QQuickWindow : :update a été appelée.

Cette image révèle toutefois un autre élément : pendant la synchronisation, le fil de rendu émet des signaux. La première manière d'intégrer du code OpenGL avec Qt Quick 2 utilisera justement ces signaux. En effet, il est possible d'y connecter des slots et d'appeler des fonctions OpenGL à l'intérieur de ces slots, ce qui permet de mélanger du code OpenGL arbitraire à l'intérieur du processus de rendu de Qt Quick 2, ce qui est exactement l'objectif poursuivi.

3 Et après ?

Avec ces quelques éléments sur le fonctionnement du moteur de rendu de Qt Quick 2, les prochaines parties de cette série étudieront trois mécanismes

différents pour intégrer du code OpenGL dans une application Qt Quick :

1. Des incrustations dans une scène OpenGL ;
2. Des éléments personnalisés exploitant directement OpenGL ;
3. Un contrôle manuel du rendu de Qt Quick.

4 Article original

Le blog KDAB est rédigé par les ingénieurs de KDAB s'occupant des formations, de la consultance ainsi que du développement (de Qt et de produits additionnels). Vous pouvez trouver les versions originales : lien 11.

Cet article est une traduction de l'article original écrit par Giuseppe D'Angelo paru le 22 octobre 2015 : lien 12.

Cet article est une traduction de l'un des articles en anglais écrits par KDAB. Les éventuels problèmes résultant d'une mauvaise traduction ne sont pas imputables à KDAB.

Retrouvez l'article de ***Giuseppe D'Angelo*** *traduit par* ***Thibaut Cuvelier*** *en ligne : lien 13*

Perl

Les dernières news Perl

Sortie officielle de Perl 6 (Version 1.0 de production)

Les équipes de développement de Perl 6 et Rakudo ont annoncé le 25 décembre 2015 la disponibilité de la version « Noël » (décembre 2015 - 2015.12) de Rakudo Perl 6 #94. Cette version couvre la version v6.c « Christmas » de la spécification du langage.

Pour rappel, **Rakudo Star** est une **distribution de Perl 6** conçue pour une utilisation par les utilisateurs précoces du langage. Elle comprend :

- **Rakudo Perl 6**, un **compilateur** pour le langage de programmation Perl 6 ciblant la machine virtuelle MoarVM ;
- la machine virtuelle MoarVM, qui sert d'interface entre le compilateur et le système d'exploitation ou la machine sous-jacente ;
- de la documentation ;
- une suite de modules que les utilisateurs peuvent trouver utiles.

Cette version apporte également des changements notables. Il faut surtout noter que le compilateur Rakudo Perl 6 livré avec cette release de Rakudo Star est annoncé officiellement comme une version de production.

Tout en étant ravies de mettre à la disposition du public cette première distribution officielle de Perl 6, les équipes de Rakudo soulignent cependant que cela ne sonne pas la fin du développement de Rakudo, bien au contraire. De nouvelles versions continueront à sortir mensuellement pour améliorer les performances et l'expérience utilisateur.

En fait, ce n'est pas tant cette sortie de Rakudo qui constitue la principale nouvelle de ce Noël 2015, mais la finalisation de cette version (6.c version[^3]) de la spécification du langage, connue sous le nom « roast » (Repository Of All Spec Tests - dépôt de l'ensemble des jeux de tests). Cette version maintenant figée des spécifications contient plus de 120 000 tests, qui ont tous été passés avec succès au moins une fois dans certaines circonstances ou sur une architecture donnée.

Cela ne signifie pas que Rakudo lui-même soit figé, il y a encore beaucoup de travail pour améliorer la vitesse, la portabilité et la stabilité. Mais le fait que la spécification de détail du langage soit maintenant figée et stable signifie que l'on peut maintenant réellement développer des projets en Perl 6 sans risquer de voir cette spécification évoluer lors de nouvelles distributions.

La gestation aura été longue, puisque les premières discussions publiques sur Perl 6 ont eu lieu il y a 15 ans, mais le bébé est vraiment prometteur. Parmi les nouveautés particulièrement dignes d'intérêt, on peut citer :

- les regex et surtout les grammaires de Perl 6, en particulier la possibilité d'ajouter dynamiquement des nouveaux éléments syntaxiques à la grammaire Perl 6 existante, rendant le langage intrinsèquement malléable et évolutif ;
- un nouveau système de programmation orientée objet particulièrement flexible, puissant et expressif ;
- les fonctions multiples et la capacité de créer dynamiquement de nouveaux opérateurs ou de surcharger des opérateurs existants ;
- un modèle de programmation fonctionnelle très enrichi avec en particulier le support aux listes paresseuses ;
- un modèle de programmation parallèle et concurrente de haut niveau, fiable, facile à utiliser et extrêmement prometteur.

Le site http ://www.developpez.com/ propose des tutoriels et une documentation en français exceptionnellement riche et abondante sur Perl 6 : lien 14. Aucun autre site en français n'offre des documents s'approchant même de loin de la richesse de cette documentation et, pour certains aspects, celle-ci est même plus approfondie que ce que l'on peut trouver en anglais.

Rakudo Star est téléchargeable à cette adresse : lien 15.

*Commentez la news de **Lolo78** en ligne : lien 16*

ALM

Les derniers tutoriels et articles

Git - Démarrage rapide sous Linux

Je vous propose ici un petit tutoriel de mise en œuvre rapide de Git sous Linux.
Ce tutoriel est destiné aussi bien au développeur solitaire qui désire bénéficier rapidement d'un système de gestion de versions simple et efficace, qu'à une équipe voulant mettre en place rapidement un système de gestion de versions collaboratif.
Ce tutoriel est accompagné d'une Visual cheat-sheet Git : lien 17.

1 Avant-propos

git est un outil de gestion de versions décentralisé.

Je vous propose ici un petit tutoriel de mise en œuvre rapide de ***Git*** sous Linux. Ce tutoriel est destiné aussi bien au développeur solitaire qui désire bénéficier rapidement d'un système de gestion de versions simple et efficace, qu'à une équipe voulant mettre en place rapidement un système de gestion de versions collaboratif.

Nous n'aborderons pas ici :

- les problématiques hiérarchiques (gestion des différents rôles/droits des utilisateurs) ;
- les aspects sécurité (mise en place de clé publique/privée pour les accès SSH, gestion des utilisateurs sur le dépôt...) ;
- la mise en place de façades (Serveur HTTP/HTTPS pour l'accès au dépôt, Application web de navigation dans le dépôt...).

Je décris la procédure d'installation pour un système Linux *Debian*, mais elle peut facilement être adaptée à d'autres distributions.
Git est également disponible sous *Windows* et *Mac* : lien 18.

Ce tutoriel a été écrit pour la version 1.7 de ***Git***, mais il reste valable pour la version 2.x.

Si vous voulez en savoir un peu plus sur ***Git*** avant de commencer :

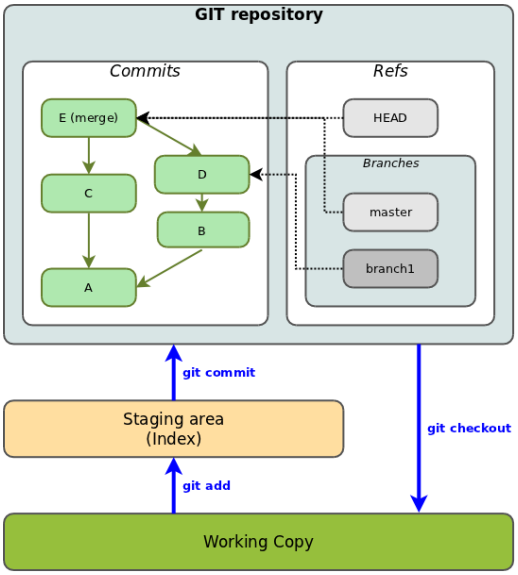
- le site officiel : lien 19 ;
- le livre *ProGit 2de édition* : lien 20 ;
- une traduction en français du *Git Community Book* : lien 21.

Vous pouvez également télécharger ma Visual cheat-sheet Git : lien 22.

Le logo de Git présenté ici est celui disponible sur le site officiel lien 23 de Git et a été réalisé par Jason Long sous licence Creative Commons Attribution 3.0.

2 Quelques notions

Avant de commencer, il est nécessaire d'aborder quelques notions importantes dans **Git**. Voici l'organisation de base d'un dépôt *Git* local :



— Le ***Git repository*** est le dépôt à proprement parler. Il contient tous les *commits*, les *branches*, les *tags*...
  — Un ***commit*** est une sorte de « photo » du répertoire de travail. Il enregistre les fichiers modifiés depuis le *commit* précédent.
  — Une ***branche*** est un pointeur sur un *commit*.
  — Un ***tag*** est une étiquette associée à un *commit*. Elle peut être annotée par son créateur.
— La ***Staging area*** (ou ***Index***) sert d'aide à la création d'un *commit*. C'est une zone particulière, gérée par *Git*, dans laquelle nous ajoutons des « photos » de fichiers. Lorsque cette zone est cohérente, en d'autres termes, quand elle contient les photos de tous les fichiers dont nous voulons enregistrer la version, alors il suffit de faire un *commit* de la *staging area*.
— La ***Working copy*** (ou ***Copie de travail***) est le répertoire de travail contenant les fichiers du projet. *Git* enregistre sur quelle branche elle pointe (la référence ***HEAD***) afin de placer correctement le prochain *commit*.

Je vous conseille de vous référer aux liens fournis dans l'introduction, notamment le livre ProGit (2de édition), afin de bien comprendre les notions que nous venons de voir : lien 24.

3 Configuration du poste de développement

3.1 Installation

Nous allons commencer par installer *Git* ainsi que deux utilitaires graphiques assez pratiques (ils vont nous faciliter la vie par rapport au terminal pour certaines actions, comme la gestion de la *staging area*, la consultation de l'historique...) :

— ***gitk*** : pour visualiser les *branches* et les *commits* ;
— ***git-gui*** : pour gérer le dépôt.

```
1 sudo apt-get install git git-man gitk
    git-gui
```

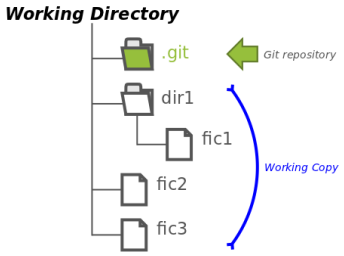
3.2 Configuration

Nous allons configurer le nom d'utilisateur et le mail sous lesquels vont apparaître nos *commits* dans l'historique :

```
1 git config --global user.name "<user>"
2 git config --global user.email "<email>"
```

4 Pour le développeur solitaire...

Nous allons ici créer simplement un dépôt directement dans le répertoire de travail du projet, afin de permettre à *Git* de gérer les versions des fichiers. Nous aurons donc une arborescence comme ceci :



Pour ce faire, nous initialisons un nouveau dépôt :

```
1  git init <workingDir>
```

Nous allons créer notre *commit* initial :

```
1  cd <workingDir>
2  echo "Répertoire contenant les sources
       du projet XXX" > README
3  git add README
4  git commit -m "commit initial"
```

Revenons, plus en détail sur les quatre commandes précédentes :

1. Entrée dans le dossier de la *working copy*.
2. Création d'un fichier README.
3. Ajout du fichier README à la *staging area*.
4. *Commit* de la *staging area* avec le commentaire « commit initial ». Comme il s'agit du premier *commit*, ceci a pour effet de créer la *branche* « master » et d'y ajouter le *commit*.

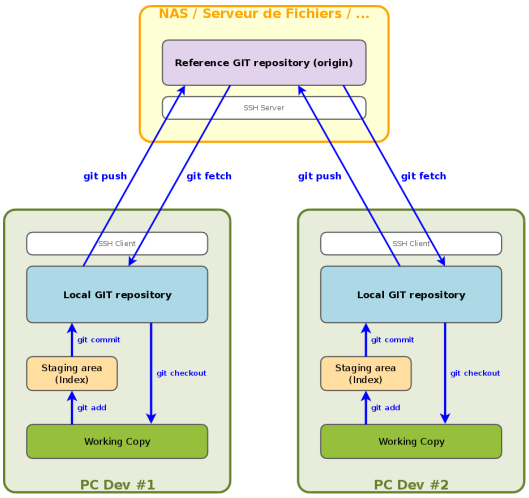
Ensuite, au cours du développement, il suffit d'appliquer le même principe afin de faire évoluer le dépôt local. Je vous conseille de vous référer aux liens fournis dans l'introduction, notamment le livre ProGit (2de édition) (lien 25), et à la Visual cheat-sheet Git (http ://loic-guibert.developpez.com/tutoriels/git/get-started/fichiers/git-cheatsheet.pdf) afin d'aborder le travail au quotidien avec *Git* (création de branches, fusion, retour en arrière...).

Exécuter la commande *git gui* depuis le répertoire de travail permet de lancer un utilitaire graphique qui va nous faciliter la vie pour les *commits* : ajout/retrait de fichiers dans la *staging area*, *commit* avec commentaire...

5 Travail en équipe...

Afin de pouvoir commencer rapidement à travailler, je vous propose ici une organisation très simple avec une mise en place rapide. Nous mettons tout le monde au même niveau, pas de gestion de rôles ni de droits sur les dépôts.

Voici, l'organisation proposée :

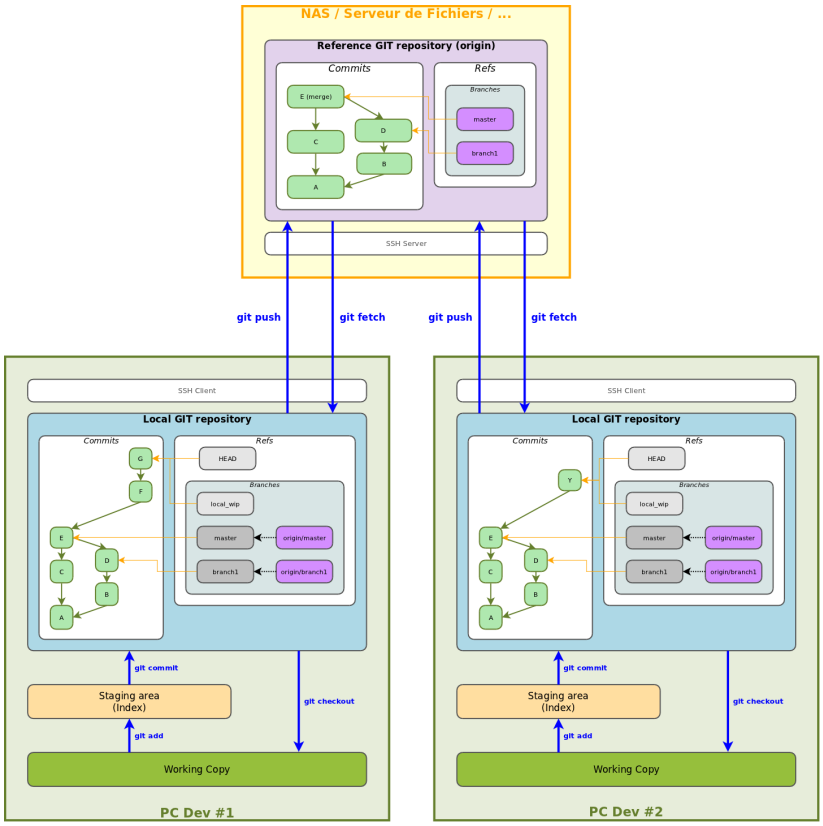


Nous mettons en place un dépôt « central » de référence. C'est le dépôt « officiel » ou « public » du projet. C'est lui qui va contenir tout le travail d'équipe.

Chaque développeur possède en local une copie liée au dépôt de référence (voir man git remote (lien 26), livre ProGit (lien 27)). Ces dépôts contiennent (au moins) une branche locale de travail (« local_wip » dans le schéma suivant). Ainsi, chaque développeur travaille sur sa branche locale et rapatrie régulièrement les modifications du dépôt de référence afin de maintenir sa copie à jour.

Il fait l'intégration de son travail (avec des *rebase/merge*...) sur la branche « publique » concernée (« master » dans le schéma suivant) et pousse les modifications de la branche « publique » sur le dépôt de référence (« origin/master » dans le schéma suivant).

Voici un schéma plus détaillé de l'organisation et du contenu des dépôts :



5.1 Mise en place du dépôt de référence

Afin de mettre en place rapidement le dépôt de référence, nous allons utiliser *Git* avec le protocole *SSH*. C'est, en général, la solution la plus simple et rapide à mettre en œuvre. Nous supposons ici que le serveur SSH est installé et opérationnel sur la machine qui va héberger le dépôt.

Intallation de *Git* :

```
# Installation de git
sudo apt-get install git
```

Création d'un utilisateur système dédié à *Git*, que nous utiliserons pour nous connecter au dépôt via le serveur SSH. Créons, par exemple, l'utilisateur « git ». :

```
sudo adduser --no-create-home --home /var/git --shell /bin/bash git
sudo passwd git
```

Création du répertoire de base des dépôts *Git* :

```
sudo mkdir /var/git
sudo chown git:git /var/git
sudo chmod 2775 /var/git
```

Suivant votre configuration du serveur SSH, faites le nécessaire afin qu'il soit possible de se connecter avec l'utilisateur « git » en SSH sur cette machine.

Création du dépôt sans *working copy* :

```
git init --bare /var/git/<monDepot>.git
# l'option --bare pour ne pas avoir de working copy
```

Test et Initialisation du dépôt :

```
# Récupération du dépôt
cd /tmp
git clone ssh://git@127.0.0.1[:portSSH]/~/<monDepot>.git monDepot

# Commit initial
cd /tmp/monDepot
echo "Répertoire contenant le sources du projet XXX" > README
git add README
git commit -m "commit initial"
git push origin master:master

# Nettoyage
cd ..
rm -rf monDepot
```

5.2 Pour chaque développeur

5.2.a Récupération du dépôt de référence

```
git clone ssh://git@<IP_Machine_Ref>[:portSSH]/~/<monDepot>.git <cheminRepTravail>
cd <cheminRepTravail>
```

Création d'une branche locale de travail « local_wip » :

```
1 git checkout -b local_wip
```

Vous êtes prêt à commencer les développements !

5.2.b J'ai bien bossé, je veux diffuser mon travail au reste de l'équipe

Comme vous pouvez le constater sur le schéma de présentation, chaque développeur possède un dépôt local lié au dépôt de référence. Or la « synchronisation » n'est pas automatique. C'est à chaque développeur de la réaliser manuellement (mais ceci est outillé bien sûr !).

Ainsi, il ne faut pas oublier que **chaque *commit* fait par un développeur n'est pas envoyé automatiquement au dépôt de référence**. Il faut garder à l'esprit que c'est le développeur qui envoie, quand il le souhaite, ses modifications après avoir fait l'intégration en local de son travail.

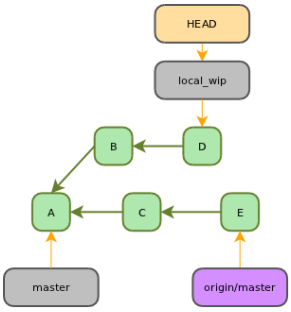
Voici une manière de procéder :

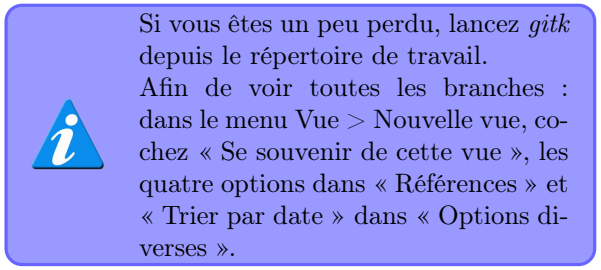
— j'ai réalisé plusieurs *commits* sur ma branche locale de travail (« local_wip ») ;

— entre-temps, un coéquipier a poussé quelques *commits* sur la branche « master » du dépôt de référence.

Je rapatrie les modifications depuis le dépôt de référence :

```
1 git fetch
```

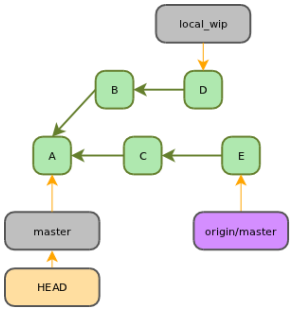




Si vous êtes un peu perdu, lancez *gitk* depuis le répertoire de travail.
Afin de voir toutes les branches : dans le menu Vue > Nouvelle vue, cochez « Se souvenir de cette vue », les quatre options dans « Références » et « Trier par date » dans « Options diverses ».

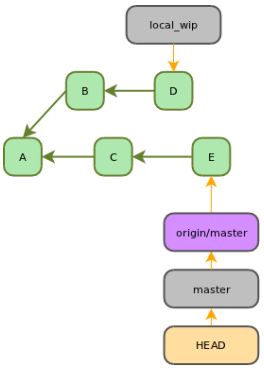
Je bascule sur la branche « master » :

```
1 git checkout master
```



Comme je n'ai rien fait sur « master » depuis le dernier *fetch*, *Git* m'indique que je peux faire un *fast-forward*. En clair, pour que ma branche soit à jour, il n'y a qu'à la faire pointer sur le dernier *commit* de la branche « origin/master » :

```
1 git merge origin/master
```



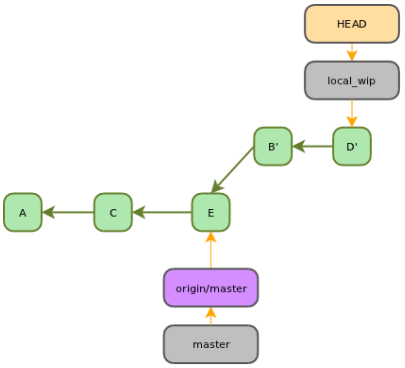
Ensuite, deux solutions :

— soit je fusionne (*merge*) ma branche « local_wip » dans la branche « master » ;

— soit je la *rebase* sur « master ».

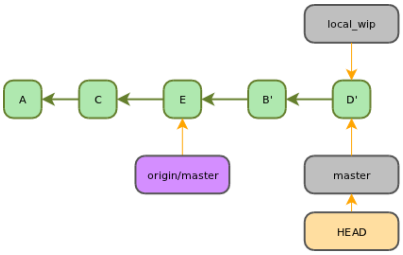
Je vais faire un *rebase* afin de garder un historique linéaire sur la branche « master » (on n'aura pas de *commit* de *merge*) :

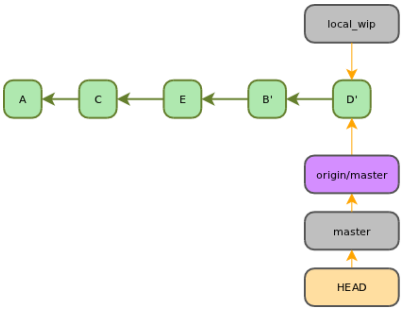
```
1 git checkout local_wip
2 git rebase master
```



Ensuite, je vais appliquer tous les *commits* de ma branche sur la branche « master » :

```
1 git checkout master
2 git merge local_wip
```





Comme nous venons de faire un *rebase*, on a changé l'origine de la branche « local_wip » pour la faire pointer sur le dernier *commit* de la branche « master ». Ainsi, le *merge* de « local_wip » dans « master » est un *fast-forward* et on a donc bien un historique linéaire !

Je n'ai plus qu'à envoyer mes modifications au dépôt de référence :

```
1 git push
```

Je n'oublie pas de repasser sur « local_wip » pour reprendre le développement :

```
1 git checkout local_wip
```

Ensuite, au cours du développement, il suffit d'appliquer le même principe afin de faire évoluer le dépôt local. Je vous conseille de vous référer aux liens fournis dans l'introduction et à la Visual cheat-sheet Git (lien 28), afin d'aborder le travail au quotidien avec *Git* (création de branches, fusion, retour en arrière...).

6 J'ai commencé tout seul, mais maintenant, je veux partager mon travail...

Nous allons voir ici qu'il est possible, à tout moment, de créer un dépôt local et de le partager. Ceci se fait en trois étapes :

1. Création du dépôt local.
2. Création d'une copie *bare* du dépôt local et envoi sur un « serveur ».
3. Référencement du dépôt distant en tant que *remote* dans le dépôt local.

6.1 Création du dépôt local

Si vous n'avez pas encore créé de dépôt local, il faut le faire maintenant.

Si vous avez déjà créé le dépôt local et que vous avez fait tous les *commits* nécessaires pour qu'il soit à jour, vous pouvez passer directement à la section suivante.

Création du dépôt local :

```
1 cd <workingDir>
2 git init .
```

Commit initial :

```
1 echo "Dépôt du projet XXX" > README
2 git add README
3 git commit -m "Initialisation du dépôt"
```

Ajout des fichiers existants :

```
1 # Soit via git gui :
2 git gui
3 # Soit à la main :
4 git add <lesFichiersDuProjet>
5 git commit -m "Initialisation du projet"
```

6.2 Création du dépôt distant

Pour créer le dépôt distant, nous allons faire un clone « *bare* » du dépôt local que nous enverrons directement sur le serveur :

```
1 cd ..
2 git clone --bare -l <workingDir> <
    nomDepotDistant>.git
3 scp -r <nomDepotDistant>.git git@<server
    >:~/
```

6.3 Référencement du dépôt distant

Maintenant que le dépôt distant est publié, il faut le référencer dans le dépôt local afin de pouvoir le mettre à jour :

```
1 cd <workingDir>
2 git remote add origin <urlDepotDistant>
3
4 # On spécifie que la branche master
    locale suit la branche origin/master
5 echo "" >> .git/config
6 echo "[branch \"master\"]" >> .git/
    config
7 echo "  remote = origin" >> .git/config
8 echo "  merge = refs/heads/master" >> .
    git/config
```

À partir de là, nous nous retrouvons dans la même configuration que dans la section « Travail en équipe... » après avoir fait un *git clone*

7 Conclusion

Ce tutoriel n'est qu'un guide de démarrage rapide. ***Git*** est un outil puissant, fiable et efficace. Prenez le temps de l'essayer grandeur nature. N'hésitez pas à approfondir le sujet avec le livre ProGit (2de édition) dont je parlais dans l'avant-propos, il est assez facile d'accès : lien 29.

Une fois familiarisé avec *Git*, vous aurez du mal à vous en passer !

Retrouvez l'article de ***Loïc Guibert*** *en ligne : lien 30*

Algorithmique

Les dernières news Algorithmique

Un algorithme quasipolynomial pour l'isomorphisme de graphes, la contribution de László Babai fait grand bruit en informatique théorique

La théorie de la complexité, en informatique, classifie les problèmes selon leur difficulté intrinsèque, c'est-à-dire selon la complexité en temps de l'algorithme le plus efficace pour les résoudre. Par exemple, pour trier un tableau, il existe une multitude d'algorithmes (lien 31) : le tri par insertion, le tri par fusion, le tri par tas ou encore, le plus célèbre, le tri rapide. Les plus efficaces ont une complexité pseudolinéaire : pour trier n éléments, il leur faudra $\Theta\left(n \log n\right)$ opérations ; d'autres algorithmes, comme le tri par insertion, ont une complexité quadratique : le nombre d'opérations évolue comme $\mathcal{O}\left(n^2\right)$. Les premiers algorithmes pourront trier des tableaux beaucoup plus grands que les seconds, peu importe le langage de programmation utilisé ou la minutie d'implémentation.

Classes de complexité

En réalité, la théorie de la complexité distingue deux catégories principales de problèmes : ceux qui acceptent une solution en un temps polynomial (des problèmes « faciles »), comme le tri, la recherche de plus court chemin, la multiplication matricielle (ils forment l'ensemble $\mathcal{P}$), puis ceux pour lesquels aucun algorithme polynomial n'est connu, comme le voyageur de commerce (lien 32) ou l'analyse de formules en logique ($\mathcal{NP}$). Une contrainte supplémentaire pour être dans $\mathcal{NP}$ est qu'il soit possible de vérifier si une réponse est acceptable en un temps polynomial : pour le voyageur de commerce, il suffit de vérifier que toutes les villes sont visitées. Par contre, pour trouver la solution optimale, il n'y a pas de technique qui soit vraiment meilleure que d'explorer toutes les possibilités (du moins, d'un point de vue théorique : il reste possible de résoudre de grandes instances en des temps raisonnables).

En informatique théorique, un problème récurrent est de savoir si $\mathcal{P} = \mathcal{NP}$. Aucune preuve n'a pu être avancée jusqu'à ce jour, malgré des dizaines d'années d'essais. Si cette égalité était vérifiée, il deviendrait possible de résoudre tous ces problèmes « difficiles » en un temps polynomial — ce qui pourrait changer la face du monde. Rares sont ceux, cependant, qui croient que ces deux classes coïncident : si tel était le cas, on aurait déjà trouvé un algorithme polynomial pour ces problèmes.

Isomorphisme de graphe

Isomorphisme de graphe $\mathcal{P}$ ou $\mathcal{NP}$, comme l'isomorphisme de graphes. Il s'agit de déterminer si deux graphes ont la même structure (ils sont alors dits *isomorphes*), c'est-à-dire si leurs points peuvent correspondre entre eux tout en gardant les liens entre ces points. (Par exemple, les relations d'« amitié » dans un réseau social comme Facebook peuvent être représentées avec des points pour les individus et des liens pour ceux qui se sont déclarés « amis ».)

Les applications de ces isomorphismes sont nombreuses, telle la reconnaissance de l'unicité d'un visiteur sur un site Web de par son comportement ou l'identification de régions pour implanter des infrastructures lors d'une réflexion urbanistique. Un bon nombre de ces applications est cependant de haut vol, comme pour l'optimisation du code dans un compilateur ou, en informatique théorique, la vérification de programmes, notamment dans des contextes parallèles, ou l'égalité de langages reconnus par des automates.

L'avancée de László Babai

Récemment, László Babai a prouvé qu'il existait un algorithme de complexité quasipolynomiale pour résoudre ce problème d'isomorphisme, c'est-à-dire qu'il nécessite un nombre d'opérations $\mathcal{O}\left(\exp\left(\log^c n\right)\right)$ (le cas où $c = 1$ étant polynomial), par rapport à $\mathcal{O}\left(\exp\sqrt{n \log n}\right)$ précédemment. Même si l'avancée semble faible, ce résultat montre que l'isomorphisme a une classe de complexité inférieure à $\mathcal{NP}$ (mais n'est pas forcément dans $\mathcal{P}$).

Toutefois, cet algorithme et les détails de la preuve ne sont pas très accessibles, exploitant la

théorie des groupes et les automorphismes de mots. De plus, ce résultat n'aura pas beaucoup d'implications pratiques : ceux qui en ont besoin peuvent résoudre des isomorphismes suffisamment rapidement pour leurs besoins. Certes, ce nouvel algorithme a des garanties théoriques, mais il n'a pas encore fait ses preuves en pratique pour les cas les plus utilisés.

Voir la vidéo : lien 33.

Et alors ?

László Babai a reçu le prix Knuth cette année (lien 34), décerné par l'ACM, pour ses « contributions fondamentales dans les domaines de la conception d'algorithmes et d'analyse de la complexité ». Le résultat obtenu pour les isomorphismes de graphes est important pour plusieurs raisons, principalement théoriques. Notamment, le lien entre propriétés des groupes et des graphes (les principes sous-jacents pourraient mener rapidement à d'autres résultats du même genre en théorie de la complexité, peut-être même pour donner un algorithme polynomial pour les isomorphismes de graphes). Il faut aussi remarquer que la preuve n'a pas encore été publiée dans une revue avec comité de relecture, mais sur arXiv.

Ce résultat a donc principalement des vertus théoriques. Il pourrait aussi, en pratique, éliminer les parties aléatoires des heuristiques utilisées pour résoudre ces isomorphismes pour n'utiliser que des algorithmes plus classiques dans leur manière de fonctionner ; l'avantage serait alors une bien meilleure prédictibilité des résultats. Il pourrait aussi questionner les pratiques actuelles en cryptographie, en abaissant la classe de complexité du problème de factorisation des nombres entiers.

Commentez la news de ***Thibaut Cuvelier*** *en ligne : lien 35*

AMD lance l'initiative Boltzmann pour se relancer dans le calcul sur GPU

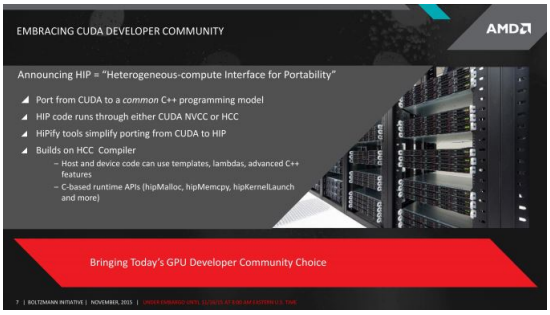
Avec une couche de compatibilité avec NVIDIA CUDA

AMD s'est récemment lancé, comme Microsoft, dans une série d'actions d'ouverture de ses codes sources sous des licences libres, notamment au niveau de ses pilotes pour Linux (AMDGPU : lien 36). À un tout autre niveau, pour le calcul sur GPU, ils espèrent que leur architecture hétérogène (dite HSA) sera compatible avec les instructions de délégation de calcul d'OpenMP dans GCC 6 (qui devrait sortir au début de l'année 2016) (lien 37), pour se mettre au même niveau qu'Intel (leur accélérateur Xeon Phi est déjà accessible par ce biais depuis GCC 5, c'est-à-dire le début de l'année 2015). De même, ils explorent le côté LLVM des compilateurs libres, avec l'ouverture prévue du code de HCC, leur compilateur hétérogène pour leur plateforme HSA.

L'initiative Boltzmann prend le nom d'un physicien autrichien (ce qui n'est pas sans rappeler les noms de code des GPU NVIDIA), à l'origine de l'approche statistique en physique (ses travaux sont fondamentaux dans certaines utilisations actuelles des GPU). Cette initiative correspond à une revalorisation des GPU à destination des serveurs dans le marché du calcul de haute performance, avec notamment un pilote Linux prévu exclusivement pour le calcul sur ces GPU (sans aucune implémentation d'OpenGL). Leur compilateur HCC permettra d'y exécuter du code C ou C++ en utilisant OpenMP, un mécanisme de parallélisation assez général (pas initialement prévu pour les GPU), c'est-à-dire avec un seul et même langage et un seul compilateur pour une série de processeurs (de manière similaire au C++ AMP, proposé par Microsoft : lien 38).

Une partie de cette initiative Boltzmann est prévue pour le portage des applications CUDA vers un « modèle de programmation C++ commun » aux différents types de processeurs disponibles, un modèle connu sous le doux nom de HIP (*heterogeneous compute interface for portability*). Il s'agit notamment d'effectuer une transpilation partielle du code CUDA, qui devrait être automatique pour nonante pour cent des cas courants — les dix pour cent restants devant être traduits à la main, ce décompte ne tenant pas compte de l'utilisation d'assembleur

(PTX) ou de l'appel direct au pilote, ce code transpilé sera toujours compilable pour les GPU NVIDIA. Au contraire, des applications CUDA compilées ne pourront pas directement être lancées sur un GPU AMD, ce qui nécessiterait l'implémentation complète dans le pilote d'une pile CUDA (et, accessoirement, une licence de la part de NVIDIA pour ce faire, déjà proposée dans le passé : lien 39). Ce mouvement est absolument requis pour qu'AMD se relance dans la course du calcul scientifique de haute performance avec ses solutions GPGPU (et pas simplement des APU : lien 40), au vu de la quantité de code CUDA existant.



Globalement, cette initiative Boltzmann matérialise un véritable retour en grande pompe dans le domaine du HPC, une niche très lucrative : le matériel existe déjà, mais l'environnement logiciel était encore défaillant pour reprendre des parts de marché, à Intel et NVIDIA. Les premiers résultats devraient arriver au premier trimestre 2016, avec des préversions. Restera à voir l'impact sur la performance.

Commentez la news de ***Thibaut Cuvelier*** *en ligne : lien 41*

Pourquoi l'apprentissage profond et les réseaux neuronaux sont-ils si prometteurs ?

L'apprentissage profond et les réseaux neuronaux sont à la mode pour le moment dans le domaine de l'apprentissage automatique : Google, NVIDIA et plus récemment Microsoft proposent des bibliothèques (lien 42), plus ou moins ouvertes, pour faciliter leur utilisation.

De fait, l'apprentissage profond accumule les succès ces derniers temps, y compris pour battre des humains au jeu de go (lien 43) — même si le meilleur joueur au monde, selon les classements actuels, Lee Sedol, estime encore pouvoir battre ce système d'intelligence artificielle. L'intérêt du jeu de go est sa complexité, malgré des règles relativement simples : il existe approximativement 10^{761} parties de go, contre « à peine » 10^{120} d'échecs (un nombre bien plus abordable actuellement).

Apprentissage d'un réseau

Cependant, de manière théorique, rien ne pouvait justifier les succès des réseaux neuronaux, qui sont l'outil principal derrière l'apprentissage profond. Depuis la première vague d'intérêt de la part du monde académique, dans les années 1990, leur étude avait montré la présence de nombreux minima locaux de l'erreur totale. L'apprentissage d'un réseau neuronal se fait en définissant la pondération des entrées de chaque neurone : changer un peu ces poids peut avoir un grand impact sur la prédiction du réseau.

Pour choisir cette pondération, tous les algorithmes testent le réseau sur des données pour lesquelles le résultat est connu : par exemple, un son et les mots auxquels il correspond ; la différence correspond à l'*erreur* commise par le réseau. La présence de ces *minima locaux* signifie que, une fois l'exécution de l'algorithme terminée, la pondération n'est pas forcément idéale : en changeant quelques valeurs, il peut être possible de diminuer drastiquement l'erreur totale. L'objectif des algorithmes d'apprentissage est d'atteindre le *minimum global* d'erreur.

Premières analyses et verre de spin

Jusqu'à présent, l'analyse théorique des réseaux neuronaux s'était portée sur des réseaux de quelques neurones : ces minima locaux sont alors présents en grand nombre et sont assez éloignés les uns des autres. Cette caractéristique menace alors la performance des réseaux, puisque le minimum local après apprentissage peut être très éloigné du minimum global.

Ce comportement correspond, en physique, à celui des verres de spin, « des alliages métalliques comportant un petit nombre d'impuretés magnétiques disposées au hasard dans l'alliage » (lien 44) : l'éner-

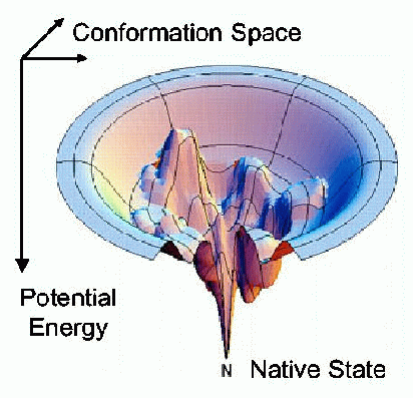
gie du matériau dépend fortement de la configuration des impuretés, qui présente un grand nombre de minima locaux éloignés du minimum global. Ce verre de spin est alors coincé dans une configuration dite métastable : en réorganisant très légèrement les impuretés, l'énergie globale pourrait baisser assez fortement.

Nouvelles analyses

Le seul résultat théorique dont on disposait jusque l'année dernière était que certains réseaux neuronaux correspondent exactement aux verres de spin. Cependant, le résultat obtenu par l'équipe de Yann LeCun (directeur du laboratoire d'intelligence artificielle de Facebook) montre, au contraire, que, pour un très grand nombre de neurones, la fonction d'erreur a plutôt la forme d'un entonnoir : les minima locaux sont très rapprochés du minimum global. Plus le réseau est grand, plus ces points sont rassemblés autour du minimum global. Or, justement, l'apprentissage profond propose d'utiliser un très grand nombre de ces neurones, plusieurs millions : le résultat d'un apprentissage n'est donc jamais loin du minimum global.

Plus précisément, les algorithmes d'apprentissage convergent vers des points critiques. Les chercheurs ont montré que la majorité de ces points critiques sont en réalité des points de selle et non des minima : ils correspondent à une zone plate, avec des directions montantes et descendantes. Il est donc relativement facile de s'en échapper, en suivant la direction descendante (en termes d'erreur). Globalement, les vrais minima (qui correspondent à des cuvettes : seulement des directions qui augmentent l'erreur) sont assez rares — et proches de la meilleure valeur possible.

Physiquement, les réseaux neuronaux correspondent donc plus à des « entonnoirs de spin », avec des formes plus sympathiques (lien 45) : l'énergie de la configuration varie de manière abrupte, sans véritablement offrir de minimum local. Ces matériaux trouvent bien plus facilement leur configuration native (avec une énergie minimale).



Ces résultats confirment donc que des techniques comme la descente de gradient stochastique (SGD) peuvent fonctionner : la fonction d'erreur d'un réseau neuronal est à peu près convexe. Cependant, les réseaux modernes sont souvent plus complexes que ceux étudiés, afin d'éviter le surapprentissage (correspondre trop bien aux données pour l'apprentissage, mais avoir du mal à reconnaître des données qui n'en font pas partie).

Néanmoins, la chimie théorique et la physique de la matière condensée proposent d'ores et déjà un panel d'outils mathématiques pour comprendre la structure de ces entonnoirs de spin et des variations plus complexes, notamment dans le cas du pliage de protéines (elles prennent une forme qui minimise cette énergie). Cette étude propose ainsi de nouveaux mécanismes d'étude des réseaux neuronaux, mais peut-être aussi de nouveaux algorithmes d'apprentissage ou techniques pour éviter le surapprentissage.

Commentez la news de ***Thibaut Cuvelier*** *en ligne : lien 46*

Faut-il mettre zlib à la retraite ?

zlib (lien 47) est une bibliothèque de compression très largement utilisée dans bon nombre d'applications : les consoles de jeu les plus récentes (PlayStation 3 et 4, Xbox 360 et One, notamment), mais aussi le noyau Linux, les systèmes de gestion des versions comme SVN ou Git ou le format d'image PNG. Sa première version publique a été publiée en 1995 par Jean-Loup Gailly et Mark Adler en implémentant la technique DEFLATE. Son succès est notamment dû à l'absence de brevets logiciels (ce qui a principalement un intérêt aux États-Unis) sur ses algorithmes, mais aussi à des débits en compression et décompression relativement élevés pour une utilisation en ressources assez faible et un bon taux de compression.

Fonctionnement de zlib

Au niveau algorithmique, DEFLATE utilise des techniques éprouvées des années 1990, principalement un dictionnaire (selon l'algorithme LZ77) (lien 48) et un codage de Huffman. Le principe d'un dictionnaire est de trouver des séquences de mots répétées dans le fichier à compresser et de les remplacer par un index dans un dictionnaire. Le codage de Huffman (lien 49) fonctionne avec un arbre pour

associer des codes courts à des séquences de bits très fréquentes. Ces deux techniques s'associent pour former la technique de compression standard actuelle : lien 50.

Concurrents modernes

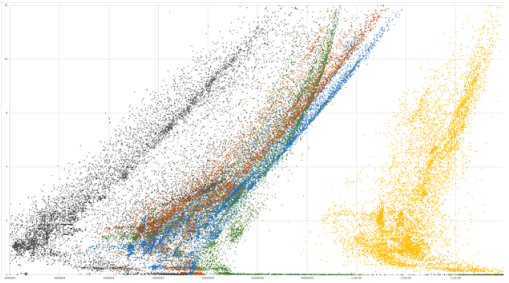
Cependant, depuis le développement de ces techniques, la recherche au niveau de la compression sans perte a fait de grands progrès. Par exemple, LZMA (lien 51) (l'algorithme derrière 7-Zip (lien 52) et XZ (lien 53)) exploite des idées similaires (plus la probabilité de retrouver des suites de bits dans le fichier à compresser, plus la manière compressée de l'écrire doit être courte), mais avec une dépendance entre différentes séquences de bits (chaîne de Markov), ainsi qu'un codage arithmétique : lien 54. Le résultat est un taux de compression souvent bien meilleur que DEFLATE, mais le processus de compression est bien plus lent, tout en gardant une décompression rapide et sans besoins extravagants en mémoire. LZHAM (lien 55) est aussi basé sur les mêmes principes avec des améliorations plus modernes et vise principalement une bonne vitesse de décompression (au détriment de la compression).

Cependant, pour un usage plus courant, les débits en compression et décompression sont aussi importants l'un que l'autre, avec un aussi bon taux de compression que possible. Par exemple, pour des pages Web d'un site dynamique, le serveur doit compresser chaque page indépendamment, puisque le contenu varie d'un utilisateur à l'autre. Plusieurs bibliothèques de compression sont en lice, comme LZ4 (lien 56) (qui se propose comme une bibliothèque très généraliste, comme zlib, mais très rapide en compression et décompression), Brotli (lien 57) (proposé par Google pour un usage sur le Web) ou encore BitKnit (lien 58) (proposé par RAD pour la compression de paquets réseau — cette bibliothèque est la seule non libre dans cette courte liste). Ces deux dernières se distinguent par leur âge : elles ont toutes deux été annoncées en janvier 2016, ce qui est très récent.

Une première comparaison concerne la quantité de code de chacune de ces bibliothèques : avec les années, zlib a accumulé pas loin de vingt-cinq mille lignes de code (dont trois mille en assembleur), largement dépassé par Brotli (pas loin de cinquante mille lignes, dont une bonne partie de tables précalculées). Les deux derniers, en comparaison, sont très petits : trois mille lignes pour LZ4 ou deux mille sept cents pour BitKnit (en incluant les commentaires, contrairement aux autres !).

Tentative de comparaison

Rich Geldreich, spécialiste de la compression sans perte et développeur de LZHAM, propose une méthodologie pour comparer ces bibliothèques : au lieu d'utiliser un jeu de données standard, mais sans grande variété (comme un extrait de Wikipedia, c'est-à-dire du texte en anglais sous la forme de XML : lien 59), il propose un corpus de plusieurs milliers de fichiers (ce qui n'a rien de nouveau, Squash procédant de la même manière : lien 60) et présente les résultats de manière graphique. Cette visualisation donne un autre aperçu des différentes bibliothèques.



Ce graphique montre, sur l'axe horizontal (logarithmique), les débits en décompression (plus un point est à droite, plus la décompression est rapide) et, sur l'axe vertical (logarithmique aussi), le taux de compression (plus il est élevé, plus le fichier a été réduit en taille). Chaque couleur correspond à un algorithme : vert pour zlib, noir pour LZHAM, rouge pour Brotli, bleu pour BitKnit et jaune pour LZ4.

Deux nuages de points sortent du lot : LZ4, tout à droite, est extrêmement rapide en décompression, tout l'opposé de LZHAM, qui propose néanmoins de bien meilleurs taux de compression. zlib montre un comportement assez étrange : le débit de décompression n'augmente plus à partir d'un certain point, contrairement aux autres bibliothèques. L'auteur propose des comparaisons plus spécifiques des taux de compression de chaque bibliothèque en fonction des fichiers : lien 61.

Et donc ?

Cette comparaison montre que les différentes bibliothèques ne sont pas toujours meilleures les unes que les autres, tout dépend du contenu du fichier, de sa taille, des ressemblances par rapport aux estimations des concepteurs (plus particulièrement dans le cas d'algorithmes qui ne s'adaptent pas dynamiquement au contenu et préfèrent utiliser des tables prédéfinies, ce qui évite de transmettre une série d'informations).

Notamment, Brotli est prévu pour le Web : il fonctionne particulièrement bien sur des données textuelles. Tout comme zlib, il utilise des tables précalculées, ce qui lui donne un avantage sur des fichiers plus petits. Au contraire, BitKnit fonctionne très bien sur du contenu binaire, bien plus courant pour les données de jeux vidéo. Ces deux bibliothèques ont donc chacune leurs points forts selon les domaines d'application prévus et y sont meilleures que zlib.

Commentez la news de ***Thibaut Cuvelier*** *en ligne : lien 62*

Un logiciel développé par une startup romande pour traquer les écrivains fantômes a été testé sur Sarkozy et Bayrou

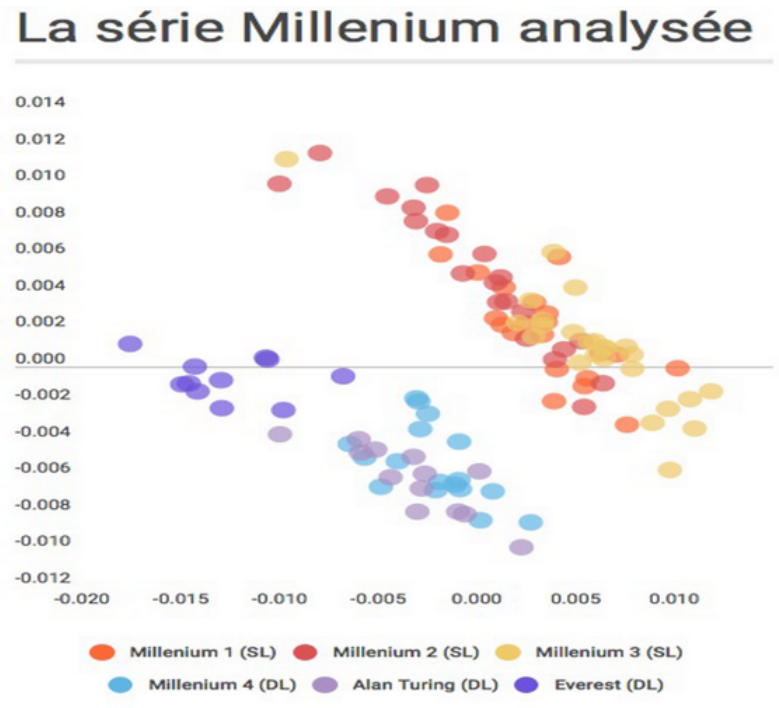
Le phénomène de Ghostwriting ou d'écrivain fantôme est soupçonné de prendre de l'ampleur même dans les meilleures universités de nombreux pays dans le monde. Il s'agit d'une pratique de facilité dans laquelle des étudiants se paient les services d'autrui pour rédiger à leur place des travaux universitaires notamment thèses et mémoires. À l'instar des plagiats, il s'agit d'un phénomène qui préoccupe les universités et grandes écoles, mais contre lequel il n'y a jusqu'à présent pas vraiment de moyen de détection et de lutte.

Une startup romande basée à Martigny a donc décidé de se lancer à la chasse des étudiants qui ont recours à cette forme de tricherie, grâce à un logiciel actuellement en phase de test, qui permettrait de détecter cela.

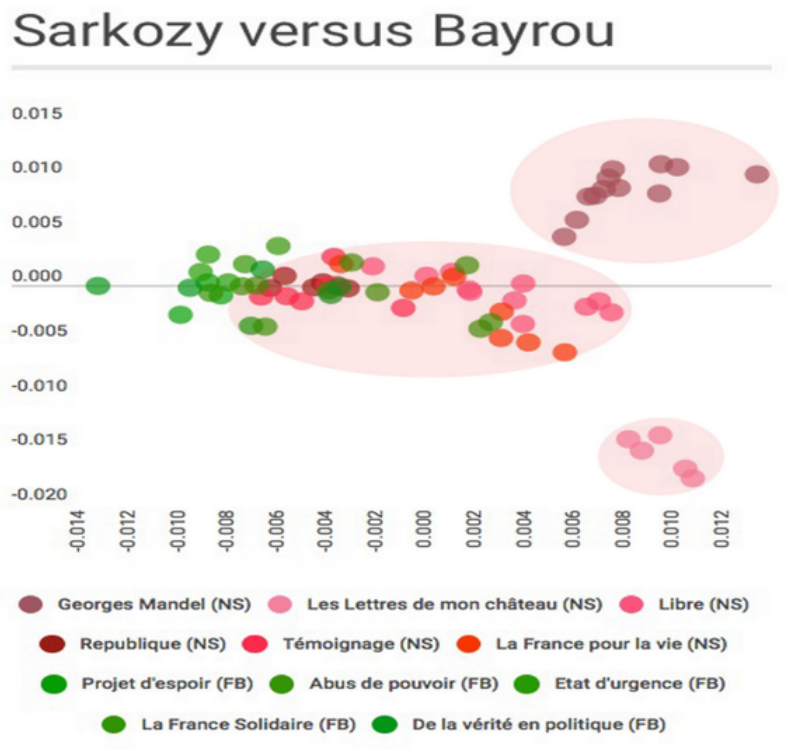
L'algorithme de détection élaboré par la société OrphAnalytics s'inspire de la recherche sur le génome. Selon Claude-Alain Roten, concepteur du logiciel, « chaque individu a un style d'écriture homogène ». En découpant un texte en plusieurs séquences, il est donc fort probable de savoir si ce texte a été écrit par plusieurs personnes. Le programme découpe en effet le texte en plusieurs séquences de taille identique auxquelles il cherche à attribuer une identité. Effets de style, fréquences et longueurs des mots, constructions des phrases, toutes ces caractéristiques stylistiques sont analysées statistiquement. En comparant différents travaux attribués à un même étudiant, on peut également savoir s'il s'agit des mêmes empreintes de style, donc du même auteur.

Le logiciel fonctionnerait dans toutes les branches académiques, du français médiéval à la finance, et dans différentes langues testées. Le logiciel a été testé sur la célèbre série de romans suédois Millénium. La série a été écrite par deux auteurs différents, les trois premiers ouvrages de la série (Millénium 1, 2 et 3) ont été écrits par Stieg Larsson (SL), alors que le quatrième (Millénium 4) a été écrit par David Lagercrantz (DL). Ce dernier a également écrit deux autres livres (Alan Turing et Everest), qui traitent de thèmes totalement différents de la série Millénium. Les résultats de l'analyse statistique des styles d'écriture sont donnés dans le graphique suivant.

Il faut avant tout savoir que chaque point représente l'identité d'une séquence de texte. L'empreinte d'une même personne devrait donc former un même nuage de points. On voit que les ouvrages de Stieg Larsson se distinguent de ceux de David Lagercrantz par deux nuages de points distincts. Ce qui montre donc qu'il s'agit de deux auteurs distincts. Le plus intéressant, c'est que les empreintes des deux derniers livres de David Lagercrantz rejoignent celle de Millénium 4, bien que les thèmes traités par l'auteur soient différents. Ce qui pourrait permettre de dire que l'empreinte d'une personne n'est pas liée au thème traité.



Régulièrement soupçonnés de faire appel à des écrivains fantômes, François Bayrou (FB) et Nicolas Sarkozy (NS) ont été également soumis à ce logiciel à travers respectivement cinq et six ouvrages qui leur sont attribués. À l'issue des analyses, Bayrou peut se voir blanchir alors que le logiciel a permis de détecter trois empreintes stylistiques différentes (trois nuages de points distincts) chez Sarkozy. Claude-Alain Roten fait toutefois remarquer que le logiciel « *ne donne pas une preuve irréfutable du ghostwriting* », mais « *il met en avant les textes suspects* ».



Commentez la news de ***Michael Guilloux*** *en ligne : lien 63*

Intelligence artificielle

Les dernières news Intelligence Artificielle

Boston Dynamics apporte une mise à jour majeure à son robot ATLAS qui fait de lui « l'un des humanoïdes les plus avancés à l'existence »

Boston Dynamics est une petite société américaine située dans le Massachusetts aux États-Unis. Depuis 2013, la société a été rachetée par Alphabet, la société qui englobe Google. Boston Dynamics est reconnue pour ses activités dans la robotique à usage militaire. Parmi ses principales réalisations, on note le robot humanoïde ATLAS conçu pour diverses tâches de recherche et de sauvetage et développé pour la DARPA dans le cadre du DARPA Robotics Challenge.

Boston Dynamics a récemment publié sur YouTube une vidéo qui dévoile ce qu'elle appelle la prochaine génération d'ATLAS. La vidéo met en vedette une version massivement améliorée du robot développé dans le cadre du DARPA Robotics Challenge. Le nouvel ATLAS est plus calme, plus robuste et plus agile. Derrière ces caractéristiques, Boston Dynamics explique que le nouvel ATLAS est « *un énorme bond technologique en avant par rapport à son prédécesseur, qui était déjà un robot assez incroyable* ». « *Le nouvel ATLAS peut faire des choses que nous n'avons jamais vu d'autres robots faire avant, ce qui en fait l'un des humanoïdes les plus avancés à l'existence* », ajoute Marc Raibert, le fondateur de Boston Dynamics.

La nouvelle version d'ATLAS est conçue pour fonctionner à l'extérieur et à l'intérieur des bâtiments. « *Elle est alimentée électriquement et actionnée hydrauliquement. Elle utilise des capteurs dans son corps et ses jambes pour s'équilibrer et des capteurs LIDAR et stéréo dans sa tête qui lui permettent d'éviter les obstacles, évaluer le terrain et qui l'aident à naviguer* » sur le terrain.

Dans la vidéo qui suit, on peut voir comment le nouveau robot sort de la maison, parcourt les bois enneigés en évitant tous les obstacles pour se rendre au travail. Arrivé au travail, il s'attèle à soulever et ranger des caisses sur des étagères sans se soucier de l'ingénieur qui essaie de lui mettre des bâtons dans les roues. Le nouvel ATLAS est poussé par l'ingénieur et tombe, mais il se relève presque sans difficulté après avoir pris appui sur ses membres.

Voir la vidéo : lien 64.

Pour ceux qui se demandent si, dans la vidéo, ATLAS était contrôlé par un opérateur humain ou exécutait ses actions de manière entièrement autonome, Raibert explique que pour les scènes en plein air, un être humain lui fournit la direction générale par radio et le robot utilise ses capteurs stéréo et LIDAR pour faire un ajustement aux variations du terrain. Le nouvel ATLAS gère également de manière autonome le contrôle de ses mouvements et de son équilibre. En ce qui concerne la scène à l'intérieur de la pièce, ATLAS est capable de poursuivre les caisses et de les ranger sur les étagères une fois qu'il reçoit l'instruction de les ranger.

Commentez la news de ***Michael Guilloux*** *en ligne : lien 65*

Developpement Web

Les derniers tutoriels et articles

Implémentation d'un panier en JavaScript et HTML5

Cet article propose une implémentation d'un panier numérique entièrement codé en JavaScript. Ce panier est testé dans une page HTML5 100 % côté client.

1 Introduction

1.1 Public visé

La connaissance du langage JavaScript « orienté objet » est indispensable à la compréhension de cet article. Le lecteur doit également maîtriser le HTML5 en général et plus particulièrement les formulaires et la navigation dans le DOM.

1.2 Objectifs

Dans ce tutoriel, nous verrons dans un premier temps comment réaliser, en JavaScript pur, un objet panier (*caddy* en américain) prêt à l'emploi.

En toute rigueur, cet objet devrait être exploité côté serveur, avec un framework tel que Node.js ou io.js. Il convient donc de préciser qu'un panier côté client sans stockage persistant n'est pas une bonne idée si on veut vendre beaucoup (car beaucoup d'utilisateurs font leurs achats en plusieurs temps). Donc, ce tutoriel ne se suffit pas à lui-même pour démarrer un site e-commerce.

Cependant il ne s'agit pas d'un tutoriel Node.js, aussi nous testerons dans la seconde partie de cet article notre objet panier dans une page HTML5.

1.3 Bibliothèques et outils utilisés pour les besoins de l'article

Tout ce que nous verrons dans cet article s'exécutera localement sur le poste de travail dans un navigateur web. Seul un éditeur de texte (Notepad++ ou SublimeText) sera donc nécessaire pour ce tutoriel.

En revanche, j'ai l'habitude d'utiliser la célèbre bibliothèque Bootstrap (du non moins célèbre réseau social Twitter) pour écrire vite des formulaires qui sont disposés sur une grille. Je ne m'en prive pas dans cet article. Mais là encore, je la télécharge directement au chargement de la page donc, rien à installer.

2 L'objet panier

2.1 Les fonctionnalités du panier

Notre panier permettra de stocker les articles choisis par le client. Il sera constitué de « lignes panier ». Chaque ligne sera dédiée à un article choisi et comprendra : le code produit, la quantité, et le prix unitaire.

Les opérations réalisables par le client avec cet objet seront :

- ajouter un article ;
- retirer un article ;
- calculer le prix d'une ligne ;
- calculer le prix total du panier.

On aura donc un objet *Panier* qui sera une collection d'objets *LignePanier*.

2.2 Code source des objets

Ces deux objets sont écrits à 100 % en JavaScript.

Voici tout d'abord le code de l'objet *ligne panier* :

```
1  function LignePanier (code, qte, prix)
2  {
3      this.codeArticle = code;
4      this.qteArticle = qte;
5      this.prixArticle = prix;
6      this.ajouterQte = function(qte)
7      {
8          this.qteArticle += qte;
9      }
10     this.getPrixLigne = function()
11     {
12         var resultat = this.prixArticle
               * this.qteArticle;
```

```
        return resultat;
    }
    this.getCode = function()
    {
        return this.codeArticle;
    }
}
```

Voici maintenant le code de l'objet *Panier* :

```
function Panier()
{
    this.liste = [];
    this.ajouterArticle = function(code, qte, prix)
    {
        var index = this.getArticle(code);
        if (index == -1) this.liste.push(new LignePanier(code, qte, prix));
        else this.liste[index].ajouterQte(qte);
    }
    this.getPrixPanier = function()
    {
        var total = 0;
        for(var i = 0 ; i < this.liste.length ; i++)
            total += this.liste[i].getPrixLigne();
        return total;
    }
    this.getArticle = function(code)
    {
        for(var i = 0 ; i <this.liste.length ; i++)
            if (code == this.liste[i].getCode()) return i;
        return -1;
    }
    this.supprimerArticle = function(code)
    {
        var index = this.getArticle(code);
        if (index > -1) this.liste.splice(index, 1);
    }
}
```

La collection utilisée afin que *Panier* agrège *LignePanier* est le traditionnel objet JavaScript *Array* qui nous permettra d'accéder à chaque élément qu'il contient en parcourant ses indices.

On remarquera juste que la méthode *ajouterArticle()* vérifie au préalable l'existence dans le panier de l'article passé en paramètre. En cas de présence avérée, on ne fera qu'ajouter les quantités commandées.

3 Test de notre objet

Nous pourrions bien entendu tester cet objet dans la console JavaScript. C'est d'ailleurs ce que j'ai fait pour la mise au point et le débogage. Néanmoins, il me semblait dommage d'écrire du JavaScript sans en voir le rendu dans une interface Web. De plus, les paniers que j'ai vus jusqu'à aujourd'hui étaient traités à travers des sessions (cookies...) ou directement dans le serveur. Alors j'ai pensé qu'il pouvait être intéressant de tenter l'aventure en HTML5 en manipulant le DOM de la page.

3.1 Le cahier des charges

Comme un dessin est souvent préférable à un long discours, vous trouverez ici l'interface Web qui est attendue (lien 66) :

3.2 La page HTML5

Sans plus attendre, voici le source de la page :

```
<!doctype html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>Panier HTML5 + JavaScript</title>
        <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css" rel="stylesheet">
        <script type="text/javascript" src="panier.js"></script>
        <script type="text/javascript">
            function ajouter()
            {
                var code = parseInt(document.getElementById("id").value);
                var qte = parseInt(document.getElementById("qte").value);
                var prix = parseInt(document.getElementById("prix").value);
                var monPanier = new Panier();
                monPanier.ajouterArticle(code, qte, prix);
                var tableau = document.getElementById("tableau");
                var longueurTab = parseInt(document.getElementById("nbreLignes").innerHTML);
                if (longueurTab > 0)
                {
                    for(var i = longueurTab ; i > 0  ; i--)
```

```
21                  {
22                      monPanier.ajouterArticle(parseInt(tableau.rows[i].cells[0].
                            innerHTML), parseInt(tableau.rows[i].cells[1].innerHTML),
                            parseInt(tableau.rows[i].cells[2].innerHTML));
23                      tableau.deleteRow(i);
24                  }
25              }
26              var longueur = monPanier.liste.length;
27              for(var i = 0 ; i < longueur ; i++)
28              {
29                  var ligne = monPanier.liste[i];
30                  var ligneTableau = tableau.insertRow(-1);
31                  var colonne1 = ligneTableau.insertCell(0);
32                  colonne1.innerHTML += ligne.getCode();
33                  var colonne2 = ligneTableau.insertCell(1);
34                  colonne2.innerHTML += ligne.qteArticle;
35                  var colonne3 = ligneTableau.insertCell(2);
36                  colonne3.innerHTML += ligne.prixArticle;
37                  var colonne4 = ligneTableau.insertCell(3);
38                  colonne4.innerHTML += ligne.getPrixLigne();
39                  var colonne5 = ligneTableau.insertCell(4);
40                  colonne5.innerHTML += "<button class=\"btn btn-primary\" type=\"
                        submit\" onclick=\"supprimer(this.parentNode.parentNode.cells[
                        0].innerHTML)\"><span class=\"glyphicon glyphicon-remove\"></
                        span> Retirer</button>";
41              }
42              document.getElementById("prixTotal").innerHTML = monPanier.
                    getPrixPanier();
43              document.getElementById("nbreLignes").innerHTML = longueur;
44          }
45
46          function supprimer(code)
47          {
48              var monPanier = new Panier();
49              var tableau = document.getElementById("tableau");
50              var longueurTab = parseInt(document.getElementById("nbreLignes").
                    innerHTML);
51              if (longueurTab > 0)
52              {
53                  for(var i = longueurTab ; i > 0  ; i--)
54                  {
55                      monPanier.ajouterArticle(parseInt(tableau.rows[i].cells[0].
                            innerHTML), parseInt(tableau.rows[i].cells[1].innerHTML),
                            parseInt(tableau.rows[i].cells[2].innerHTML));
56                      tableau.deleteRow(i);
57                  }
58              }
59              monPanier.supprimerArticle(code);
60              var longueur = monPanier.liste.length;
61              for(var i = 0 ; i < longueur ; i++)
62              {
63                  var ligne = monPanier.liste[i];
64                  var ligneTableau = tableau.insertRow(-1);
65                  var colonne1 = ligneTableau.insertCell(0);
66                  colonne1.innerHTML += ligne.getCode();
67                  var colonne2 = ligneTableau.insertCell(1);
68                  colonne2.innerHTML += ligne.qteArticle;
69                  var colonne3 = ligneTableau.insertCell(2);
70                  colonne3.innerHTML += ligne.prixArticle;
71                  var colonne4 = ligneTableau.insertCell(3);
72                  colonne4.innerHTML += ligne.getPrixLigne();
73                  var colonne5 = ligneTableau.insertCell(4);
74                  colonne5.innerHTML += "<button class=\"btn btn-primary\" type=\"
                        submit\" onclick=\"supprimer(this.parentNode.parentNode.cells[
                        0].innerHTML)\"><span class=\"glyphicon glyphicon-remove\"></
                        span> Retirer</button>";
75              }
76              document.getElementById("prixTotal").innerHTML = monPanier.
                    getPrixPanier();
77              document.getElementById("nbreLignes").innerHTML = longueur;
78          }
79      </script>
80  </head>
```

```
81      <body>
82          <section  class="container">
83              <article class="well form-inline pull-left col-lg-5">
84                  <legend>Gestion du panier</legend>
85                  <label class="col-lg-3">Identifiant</label> : <input type = "number"
                        id = "id" style="width:120px" class="input-sm form-control"></
                        input><br><br>
86                  <label class="col-lg-3" >Quantité</label> : <input type = "number" id
                        = "qte" style="width:120px" class="input-sm form-control"></input>
                        <br><br>
87                  <label class="col-lg-3">Prix</label> : <input type = "number" id = "
                        prix" style="width:120px" class="input-sm form-control"></input><
                        br><br>
88                  <button class="btn btn-primary" type="submit" onclick="ajouter()"><
                        span class="glyphicon glyphicon-shopping-cart"></span> Ajouter au
                        panier</button>
89              </article>
90          </section>
91          <section class="container">
92              <article class="well form-inline pull-left col-lg-5">
93                  <legend>Contenu du panier</legend>
94                  <table id="tableau" class="table">
95                      <thead>
96                          <tr>
97                              <th>Code</th>
98                              <th>Qte</th>
99                              <th>Prix unitaire</th>
100                             <th>Prix de la ligne</th>
101                             <th>Supprimer</th>
102                         </tr>
103                     </thead>
104                 </table>
105                 <br><label>Prix du panier total</label> : <label id = "prixTotal"></
                        label>
106                 <label id = "nbreLignes" hidden>0</label>
107             </article>
108         </section>
109     </body>
110 </html>
```

Le scénario retenu pour coder cette page est basé sur la gestion de deux événements majeurs, un clic sur le bouton ajouter ou supprimer. Comme on n'utilise pas de session, il est difficile de garder en mémoire le contenu du panier. Aussi j'ai choisi de recharger le panier depuis le tableau d'affichage en bas de page à chaque clic sur l'un des boutons. Le nombre de lignes du panier sera lu et écrit dans le champ caché id= "nbreLignes".

4 Conclusion

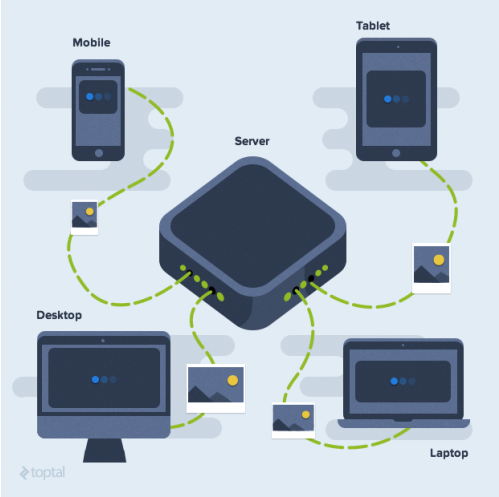
Cet article a permis de voir comment implémenter un panier en JavaScript comme on l'aurait fait avec n'importe quel autre langage. Car aujourd'hui, JavaScript est un langage aussi noble que Java ou PHP, surtout depuis qu'il s'exécute dans des serveurs. D'ailleurs, dans un prochain tutoriel je montrerai comment faire tourner ce panier plus proprement côté serveur avec Node.js.

Retrouvez l'article de ***Marc Autran*** *en ligne : lien 67*

One Size Fits Some - Guide des solutions d'images adaptatives en Responsive Web Design

Alors que les mobiles et tablettes (lien 68) s'approchent de leur but, la domination du monde, le design web et la technologie s'engagent dans une course pour s'adapter au nombre toujours croissant de tailles d'écran à gérer. Cependant, élaborer des outils pour répondre aux défis engendrés par ce phénomène amène tout un nouveau lot de problèmes, dont l'un des derniers buzzwords à la mode, le « web responsive » : lien 69. C'est le défi de faire fonctionner le web sur la majorité, si ce n'est la totalité, des appareils sans dégrader l'expérience utilisateur. Au lieu de concevoir du contenu adapté aux ordinateurs de bureau et ordinateurs portables, l'information doit être disponible sur les téléphones mobiles, les tablettes ou toute autre surface connectée au web. Cependant, cette évolution Responsive Web Design (lien 70) s'est avérée être difficile et parfois douloureuse.
Bien qu'il puisse être presque trivial d'adapter du contenu textuel, la partie délicate survient lorsque l'on prend en considération du contenu multimédia comme des images responsive, des infographies, des vidéos et autres contenus créés avec uniquement les ordinateurs fixes à l'esprit. Cela n'amène pas seulement la question d'afficher le contenu correctement, mais aussi la manière dont il sera consommé en utilisant différents appareils. Les utilisateurs de mobiles sont différents des utilisateurs du desktop. Des choses comme les forfaits Data ou la vitesse de traitement doivent également être prises en considération. Google a commencé à mettre en avant les sites « mobile-friendly » (lien 71) dans ses résultats de recherche, en spéculant (lien 72) sur le fait que cela amènerait un gain substantiel dans le rang de la page (Pagerank) de ce type de sites. De précédentes solutions ont résolu ce problème en utilisant des domaines spécifiques aux mobiles (et des redirections), mais cela a augmenté la complexité et est vite passé de mode. Tous les sites n'ont pas les moyens de prendre cette direction.

1 Sur la quête des images Responsive Web



Jusqu'ici, les développeurs et designers devaient s'assurer que le chargement de leur site était optimisé pour les utilisateurs sur mobiles. Plus de 20 % du trafic web (lien 73) aujourd'hui concerne les utilisateurs de mobiles, et le nombre croît toujours. Comme les images sont une des parts les plus grandes du contenu d'une page en taille de données, il devient prioritaire de réduire cette charge. Plusieurs tentatives ont été faites pour simplifier le rognage d'images en *responsive design*, incluant des solutions à la fois côté serveur et côté client. Avant de discuter de ces solutions, nous devons d'abord comprendre quelles sont les limitations du mécanisme actuel de lien des images.

La balise <img> a seulement un seul attribut source reliant directement à l'image elle-même. Il n'y a pas moyen de déterminer le type correct d'image à charger sans composant additionnel.

Ne pouvons-nous pas juste avoir toutes les tailles d'image incluses dans le HTML, et utiliser des règles « CSS display :none » pour toutes les cacher sauf la bonne image ? C'est la solution la plus logique dans

un monde parfaitement logique. De cette manière, le navigateur peut ignorer les images non affichées et, en théorie, ne pas les télécharger. Cependant, les navigateurs sont optimisés au-delà du sens commun de la logique. Pour faire en sorte que la page soit affichée plus rapidement, le navigateur précharge tout le contenu lié à celle-ci avant même que les feuilles de style et scripts nécessaires soient totalement chargés. Au lieu d'ignorer les grandes images destinées aux ordinateurs de bureau, on se retrouve à charger toutes les images, ce qui résulte en un chargement encore plus lourd. Cette technique CSS fonctionne uniquement pour les images prévues en tant qu'arrière-plans, car elles peuvent être définies dans les feuilles de style (avec éventuellement des *media queries*).

Donc, que faut-il faire pour un site web ?

2 Solutions de mise à l'échelle d'images côté serveur

À moins que le site soit destiné exclusivement au mobile ou qu'il possède un sous-domaine dédié mobile, nous sommes contraints d'utiliser la technique du User Agent *sniffing* pour détecter le type de périphérique et s'en servir pour renvoyer des images à la bonne taille à l'utilisateur. Cependant, tout développeur peut attester du manque de fiabilité de cette approche. De nouveaux User Agents apparaissent sans cesse, ce qui rend très fastidieux le maintien à jour d'une liste exhaustive. Et bien sûr, cela ne tient pas compte de la grande facilité pour les utilisateurs de modifier leur User Agent.

2.1 Adaptive Images

Cependant, certaines solutions côté serveur sont dignes d'intérêt. Les Adaptive images lien 74 sont une très bonne solution pour ceux qui préfèrent un *fix* côté serveur pour les *images responsive*. Cela ne nécessite aucun balisage spécial, mais utilise un petit fichier JavaScript et fait le gros du travail en *back-end*. Il nécessite PHP et un serveur *nginx* correctement configuré. Aussi, il ne repose pas sur l'User Agent *sniffing*, mais sur la largeur d'écran. Les Adaptive images fonctionnent très bien pour réduire la taille des images, mais c'est également pratique quand de grandes images ont besoin d'une direction artistique ; par exemple, des techniques de réduction comme le rognage - et non juste une mise à l'échelle.

2.2 Sencha Touch

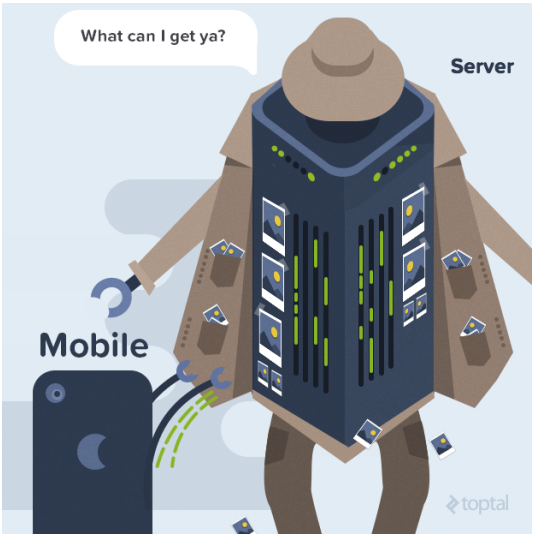
Sencha Touch (lien 75) est une autre solution *back-end* pour les images *responsive*, cependant il convient de la designer comme une solution tierce d'un prestataire externe. Sencha Touch retaillera vos images selon le User Agent reçu. Ci-dessous, un exemple basique montrant comment le service fonctionne :

```
1  <img src="http://src.sencha.io/http://
       example.com/images/kitty_cat.jpg"
       alt="My Kitty Cat">
```

Il y a aussi une option pour spécifier les tailles de l'image, dans le cas où vous ne voulez pas que Sencha retaille l'image automatiquement.

Finalement, les solutions côté serveur (et de prestataires externes) nécessitent d'allouer des ressources pour traiter les requêtes avant d'envoyer en retour la bonne image. Cela requiert un temps précieux et ralentit l'échange requête/réponse. Une meilleure solution pourrait être de déterminer par l'appareil lui-même quelles ressources requêter directement, et laisser le serveur répondre en conséquence.

3 Solutions côté client



Ces derniers temps, il y a eu de gros progrès côté navigateur pour résoudre le cas des images *responsive*.

L'élément <picture> a été introduit et approuvé par le W3C dans la spécification HTML5. Actuellement, il n'est pas répandu largement sur les différents navigateurs, mais cela ne sera pas long avant qu'il soit disponible nativement. En attendant, nous reposons sur des *polyfills* JavaScript pour supporter cet élément. Les *polyfills* sont une excellente solution pour les navigateurs obsolètes ne supportant pas cet élément.

Il y a aussi le cas de l'attribut srcset (lien 76) qui est supporté par plusieurs navigateurs basés sur le moteur Webkit pour l'élément <img>. Il peut être utilisé sans JavaScript et s'avère être une très bonne solution si les navigateurs non Webkit peuvent être ignorés. C'est un palliatif utile dans le cas où les autres solutions échouent, mais cela ne devrait pas être considéré comme une solution globale.

3.1 Picturefill

Picturefill (lien 77) est une bibliothèque *polyfill* pour l'élément <picture>. C'est l'une des bibliothèques les plus populaires parmi les solutions côté client pour le rognage et la mise à l'échelle d'images *responsive*. Il y a deux versions : Picturefill v1 imite l'élément <picture> en utilisant un élément *span* tandis que Picturefill v2 utilise l'élément<picture> pour les navigateurs le supportant, et un *polyfill* pour les autres (IE9 par exemple). Il y a un certain nombre de limitations et solutions de contournement (lien 78), en particulier pour Android 2.3 - qui est involontairement un bon exemple où l'attribut img srcset vient à la rescousse. Ci-dessous, un exemple de code HTML pour Picturefill v2 :

```
1 <picture>
2     <source srcset="/images/kitty_large.
          jpg" media="(min-width: 768px)">
3     <source srcset="/images/kitty_medium.
          jpg" media="(max-width: 767px)">
4     <img srcset="/images/kitty_small.jpg"
           alt="My Kitty Cat">
5 </picture>
```

Pour améliorer la performance pour les utilisateurs avec des forfaits limités en data, Picturefill peut être combiné avec du chargement à la demande : (lien 79). De plus, cette bibliothèque *lazyload* peut offrir un support navigateur plus large et s'occuper de cas particuliers plutôt que de compter sur des patchs correctifs.

3.2 Imager.js

Imager.js (lien 80) est une bibliothèque créée par les équipes de BBC News (lien 81) pour parvenir à des images *responsive* en suivant une approche différente de celle utilisée par Picturefill. Là où Picturefill essaie d'apporter l'élément <picture> aux navigateurs non supportés, Imager.js se concentre sur le fait de télécharger uniquement les images appropriées tout en gardant un œil sur la vitesse du réseau. Il intègre également une solution de chargement à la demande sans nécessiter de dépendance externe. Cela fonctionne grâce à des éléments *placeholder* remplacés à l'exécution par des éléments <img>. Le code ci-dessous décrit ce comportement :

```
1 <div>
2     <div class="image-load" data-src="
          http://example.com/images/kitty_{
          width}.jpg" data-alt="My Kitty
          Cat"></div>
3 </div>
4 <script>
5 new Imager({availableWidths: [480, 768,
      1200]});
6 </script>
```

Le HTML final ressemblera à ceci :

```
<div>
   <img src="http://example.com/images/
       kitty_480.jpg" data-src="http://
       example.com/images/kitty_{width}.
       jpg" alt="My Kitty Cat" class="
       image-replace">
</div>
<script>
new Imager({availableWidths: [480, 768,
    1200]});
</script>
```

Le support navigateur est bien meilleur que pour Picturefill au prix d'une solution plus pragmatique, moins tournée vers l'avenir.

3.3 Brassage de sources (Source Shuffling)

Le brassage de sources (lien 82) approche le problème des images *responsive* d'un angle légèrement différent que le reste des bibliothèques côté client. Cela rejoint davantage l'école de pensée « Mobile first », par le fait qu'il dessert la plus petite résolution par défaut. Lors de la détection d'un appareil disposant d'un plus grand écran, il échange la source de l'image pour sa version agrandie. Cela ressemble davantage à un *hack* qu'à une bibliothèque tout équipée. C'est une excellente solution pour des sites principalement destinés au mobile, mais cela implique de télécharger deux ressources au lieu d'une pour les ordinateurs à grand écran et tablettes.

D'autres bibliothèques JavaScript notables sont :

- HiSRC (lien 83) - un plugin jQuery pour les images *responsive*. La dépendance à jQuery peut être un problème ;
- Mobify.js (lien 84) - une bibliothèque plus générale pour du contenu *responsive*, incluant des images, des feuilles de style et même des bouts de JavaScript. Il « capture » le DOM avant de charger les ressources. Mobify est une solution puissante, mais peut-être un peu *overkill* si l'objectif se limite à des images *responsive*.

4 Résumé

Finalement, c'est au développeur de décider quelle approche choisir pour les images en Responsive Web Design (lien 85) pour ses utilisateurs. Cela signifie que collecter des données et effectuer des tests vous donnera une meilleure idée de la bonne approche à prendre.

Pour résumer, voici la liste des questions à se poser avant de choisir une solution d'image *responsive*.

- Est-ce que les navigateurs obsolètes sont un problème ? Si ce n'est pas le cas, considérez une approche plus moderne (exemple, Picturefill, srcset).
- Est-ce que le temps de réponse est critique ? Si ce n'est pas le cas, optez pour une solution côté serveur ou un prestataire externe.
- Est-ce que la solution doit être gérée au sein de l'entreprise ? Les solutions tierces seront évidemment éliminées d'office.
- Y a-t-il un tas d'images déjà présentes sur un site qu'il faut essayer d'adapter en *responsive* ? Y a-t-il des préoccupations quant à la validation ou la sémantique des éléments ? Cela nécessitera alors une solution côté serveur pour router vers les bonnes images, donc quelque chose comme les Adaptive images.
- Est-ce que la direction artistique est un problème (en particulier pour les grandes images avec beaucoup d'informations) ? Une bibliothèque comme Picturefill peut être une meilleure solution dans un tel scénario. Aussi, toute solution côté serveur peut fonctionner aussi bien.
- Est-ce que la désactivation de JavaScript est une préoccupation ? Cela élimine toutes les solutions côté client, il reste alors les solutions côté serveur ou externes reposant sur le User Agent *sniffing*.
- Le temps de réponse sur mobile est-il plus important que le temps de réponse sur ordinateurs fixes ? Le brassage de sources sera alors plus approprié.
- Y a-t-il un besoin de fournir un comportement *responsive* pour tous les aspects du site, et pas juste les images ? Mobify.js peut mieux convenir.
- Est-ce qu'on est arrivé à un monde parfait ? Utilisez alors CSS uniquement et display :none !

5 À propos de l'auteur

Kado Damball est un développeur JavaScript intéressé par les données et la visualisation de données. C'est aussi un passionné de machine *learning* et de *data mining*, un sous-produit de son BA en économies. Il travaille actuellement en tant que développeur freelance.

Retrouvez l'article de ***Kado Damball*** *traduit par Sylvain Pollet-Villard en ligne : lien 86*

Access

Les derniers tutoriels et articles

Réalisez un assistant de présaisie pour alimenter la base d'un serveur central

1 Avant-propos

Ce document a pour but de vous montrer comment concevoir un assistant de présaisie des données permettant à un panel d'utilisateurs de saisir les informations nécessaires à la mise à jour d'une base de données centrale située sur un serveur distant.

L'idée de ce tutoriel est née d'une situation existante pour laquelle j'ai été confronté à concevoir le même prototype d'outil que j'ai ici largement simplifié pour ne pas rendre la mise en œuvre trop complexe, notamment pour les débutants.

1.1 Niveau

Vous devez être relativement à l'aise avec Microsoft Access et connaître la conception avancée de formulaires, mais également la manipulation des requêtes avec Visual Basic, en ayant une bonne approche de ce langage afin de mettre en pratique cet exemple.

Ce tutoriel s'adresse plus particulièrement aux personnes qui possèdent déjà de bonnes notions concernant la gestion des événements avec Visual Basic sur Microsoft Access incluant la manipulation des données à l'aide de DAO.

1.2 Contact

Pour tous renseignements complémentaires, veuillez me contacter directement (Langue Argyronet) par MP : lien 87.

Si votre question est, disons **publique et technique**, merci de poster un message dans le **forum** afin d'en faire partager le contenu...

1.3 Le projet proposé

Le projet abordé dans ce tutoriel est fourni sous forme de code source à titre d'exemple qui, une fois compilé, assemblé et interprété peut nécessiter des adaptations ou des modifications en rapport avec vos besoins.

Il est tout à fait possible que votre vision des choses à l'égard de la méthodologie employée au sein des différentes fonctions ou procédures soit différente de la mienne. Aussi, n'hésitez pas à proposer vos idées si vous pensez qu'elles peuvent apporter un plus à ce projet.

Notez que j'ai cherché avant tout à simplifier l'ensemble pour que tout un chacun puisse l'assimiler de façon aisée.

1.4 Contexte métier

Le projet présenté ici concerne des données relatives à des prestations de service supervisées par certains sites pour certaines communes. Ces prestations sont soumises à des contrats nantis d'une durée de validité. Libre à vous de vous en inspirer pour votre propre situation métier ou tout autre développement particulier.

2 Pourquoi ce projet

2.1 Contexte

Ce tutoriel est né, comme bien souvent en ce qui me concerne, de l'idée d'une réalisation professionnelle qui m'a été demandée et que j'ai souhaité partager avec vous, du fait je la trouve intéressante à exploiter.

J'espère que ce sera le cas pour vous...

Pour clarifier un peu les choses, le projet présenté ici ne peut être mis en application que si vous êtes dans une situation où un besoin de centralisation des données est nécessaire, et qu'il faut préparer les données du serveur d'une part, mais aussi que votre application de gestion soit exploitée par différents sites où l'activité professionnelle (*pour le cas présenté ici*) exerce sur les communes spécifiques, ces dernières étant associées ou non à des contrats.

Bien entendu, il faut considérer que la présentation de ce projet reste surtout et avant tout théorique, car je suis bien conscient qu'il est difficile de se mettre dans une situation identique à celle qui m'a poussé à créer cet assistant.

Ceci dit, l'objectif est surtout de comprendre les points culminants de ce tutoriel à savoir :

— voir comment il est relativement aisé **d'enchaîner dynamiquement** l'ouverture de formulaires à l'image de ce que vous rencontrez lorsque vous lancez un programme d'installation ;

— approfondir vos connaissances concernant plus particulièrement l'usage de **procédures génériques** associées à des **événements**, qui évite ainsi la **redondance** de code et donc par voie de conséquence, une maintenance plus aisée ;

— voir également comment il est possible de vérifier qu'un **sous-formulaire est complété** comme attendu au moment de la saisie d'une part, et au moment de la validation globale d'autre part. J'entends par validation globale, les comportements associés au passage à l'écran précédent ou à l'écran suivant qui se traduit par le fait de conserver ou non le contenu (confirmé par un message).

D'une manière générale, du fait que cet assistant est destiné à des non-informaticiens, on fait en sorte de rendre les messages affichés qui concernent la sauvegarde, le soient de manière optimiste. En d'autres termes, si l'utilisateur n'est pas en mesure de renseigner une valeur, qu'il ne soit pas obligé de le faire.

2.2 De quoi avez-vous besoin ?

Pour mettre en œuvre ce projet, vous n'avez besoin que de Microsoft Access dans une version supérieure ou égale à la version 2000. Il n'y a pas de référence particulière autre que celles définies par défaut dans Microsoft Access lors de la création de nouveaux projets.

Libre à vous de préférer DAO 3.6 ou le nouveau moteur existant depuis la version 2007.

3 Présentation du projet

Le projet est ici et expressément une base de données Access **non scindée**. Je précise cela, car il est plus facile de distribuer une application dans cette configuration d'une part, et cela évite les problèmes d'installation d'autre part. Par ailleurs, du fait du côté volatil de l'application si je puis dire, il n'est pas nécessaire de s'investir dans une structure **frontale/dorsale**.

3.1 Structure fonctionnelle des formulaires

Dans le projet présenté ici, les éléments prépondérants sont bien entendu les formulaires. Ils sont au nombre de onze. Parmi ces formulaires, figure un formulaire d'accueil qui propose et énumère ce qui va être fait et un formulaire de fin qui récapitule la saisie.

Entre les deux, figurent tous les autres formulaires de saisie des données utilisateur qu'ils soient formulaires simples ou continus.

Concernant les formulaires de saisie, chacun d'entre eux est pourvu de boutons qui permettent de **créer** ou de **supprimer** un enregistrement, et également, d'accéder au formulaire **précédent** ou au formulaire **suivant**.

Pour chacun des formulaires qui contiennent des sous-formulaires, sont présentes deux étiquettes qui pour l'une, permet de savoir si le formulaire et dûment complété ou non et pour l'autre, le nombre d'enregistrements existants.

Il est rappelé ici que les écrans doivent être construits de telle sorte à ce qu'ils soient compris par des personnes qui n'ont pas l'habitude d'utiliser l'outil informatique et à qui l'on demande de saisir des informations avec un mode qualifié de simple.

4 Les tables du projet

Au niveau des tables, du fait que ce projet concerne dans son côté métier l'exploitation de prestations de services par rapport à des communes, certaines tables contiennent des valeurs prédéfinies.

Dans l'exemple présenté, trois tables contiennent respectivement l'ensemble des communes, des départements et des régions pour la France.

Les autres tables contiennent des valeurs qui concernent les types de contrats ou les privilèges utilisateur. Si vous décidez de mettre en œuvre ce projet dans son intégralité, il vous faudra sans doute procéder à des adaptations selon votre propre contexte professionnel.

4.1 La structure des tables de données fixes

4.1.a tbl_ListeDepartements (les départements*)

Cette table possède trois champs :

— le champ **IDDepartement** de type Texte li-

mité à trois caractères ;

— le champ **Departements** de type Texte limité à soixante-quatre caractères ;

— le champ **IDRegion** de type numérique Entier long .

La clé primaire est affectée au champ **IDDepartement**.

Nom du champ	Type de données
IDDepartement	Texte
Departement	Texte
IDRegion	Numérique

4.1.b tbl_ListeDesCommunes (Les communes*)

Cette table possède cinq champs :

— le champ **IDCommune** représenté par un identifiant de type Numérotation automatique ;

— le champ **CodeINSEE** de type Texte limité à cinq caractères ;

— le champ **NomCommune** de type Texte limité à soixante-quatre caractères ;

— le champ **CodePostal** de type Texte limité à cinq caractères ;

— le champ **IDDepartement** de type Texte limité à trois caractères .

La clé primaire est affectée aux champs **CodeINSEE** + **NomCommune** (CodeINSEE indexé sans doublons).

Nom du champ	Type de données	Description
IDCommune	NuméroAuto	Identifiant
CodeINSEE	Texte	Code INSEE de la commune
NomCommune	Texte	Nom de la commune
CodePostal	Texte	Code postal de la commune
Region	Texte	Nom de la région

4.1.c tbl_ListeDesRegions (les régions*)

Cette table possède deux champs :

— le champ **IDRegion** de type Numérotation automatique ;

— le champ **Region** de type Texte limité à soixante-quatre caractères.

La clé primaire est affectée au champ IDRegion.

Nom du champ	Type de données
IDRegion	NuméroAuto
Region	Texte

Du fait que ces tables (*) sont facultatives, dans la mesure où elles ne sont utilisées dans ce projet qu'à titre d'exemple, vous n'êtes pas obligé de les mettre en œuvre. . .

4.1.d tbl_Privileges (les privilèges utilisateur**)

Cette table possède trois champs :

— le champ **IDPrivilege** de type Numérique Entier long ;

— le champ **CodePrivilege** de type Texte limité à cinq caractères ;

— le champ **LibellePrivilege** de Texte limité à soixante-quatre caractères.

La clé primaire est affectée aux champs **IDPrivilege** + **CodePrivilege** tous indexés sans doublon.

Nom du champ	Type de données	Description
IDPrivileges	Numérique	Identifiant
CodePrivileges	Texte	Code identifiant le privilège
LibellePrivileges	Texte	Libellé du privilège

4.1.e tbl_TypesContrat (les types de contrats**)

Cette table possède deux champs :

— le champ **IDTypeContrat** de type Numérique Entier long ;

— le champ **TypeContrat** de type Texte limité à soixante-quatre caractères.

La clé primaire est affectée au champ **IDTypeContrat** indexé sans doublon.

Nom du champ	Type de données	Description
IDTypeContrat	Numérique	Identifiant
TypeContrat	Texte	Libellé du contrat

4.2 La structure des tables de données utilisateur

4.2.a tblCommunesSite (les associations communes et site)

Cette table possède trois champs :

— le champ **IDCommune** de type Numérique Entier long ;

— le champ **NoSite** de type Numérique Entier long ;

— le champ **IDDepartement** de type Texte limité à trois caractères.

La clé primaire est affectée aux champs **IDCommune** + **NoSite** tous indexés avec doublons.

Nom du champ	Type de données	Description
IDCommune	Numérique	Identifiant de la commune
NoSite	Numérique	Identifiant du site
IDDepartement	Texte	Identifiant du département

4.2.b tblContratCommunes (les associations contrat et communes)

Cette table possède deux champs :

- le champ **IDCommune** de type Numérique Entier long ;
- le champ **IDContrat** de type Texte limité à cinq caractères.

La clé primaire est affectée aux champs **IDCommune** + **IDContrat** tous indexés avec doublons.

Nom du champ	Type de données	Description
IDCommune	Numérique	Identifiant commune
IDContrat	Texte	Identifiant contrat

4.2.c tblContrats (Les contrats)

Cette table possède cinq champs :

- le champ **IDCommune** représenté par un identifiant de type Numérotation automatique ;
- le champ **IDContrat** de type Texte limité à cinq caractères ;
- le champ **LibelleContrat** de type Texte limité à 128 caractères ;
- le champ **DateFin** de type Date ;
- le champ **IDTypeContrat** de type Numérique Entier long.

La clé primaire est affectée au champ **IDContrat** indexé sans doublon.

Nom du champ	Type de données
IDContrat	Texte
LibelleContrat	Texte
DateDebut	Date/Heure
DateFin	Date/Heure
IDTypeContrat	Numérique

4.2.d tblPersonnel (Le personnel)

Cette table possède six champs :

- le champ **IDPersonne** représenté par un identifiant de type Numérotation automatique ;
- le champ **NomFamille** Texte limité à soixante-quatre caractères ;
- le champ **Prenom** de type Texte limité à soixante-quatre caractères ;
- le champ **EMAIL** de type Texte limité à soixante-quatre caractères ;
- le champ **Login** de type Texte limité à soixante-quatre caractères ;
- le champ **NoSite** de type Numérique Entier long ;
- le champ **IDPrivileges** de type Numérique Entier long.

La clé primaire est affectée aux champs **NomFamille** + **Prenom** + **NoSite** tous indexés avec doublons.

Nom du champ	Type de données	Description
IDPersonne	NuméroAuto	Identifiant
NomFamille	Texte	Nom de l'employé
Prenom	Texte	Prénom de l'employé
EMAIL	Texte	Adresse mail
Login	Texte	Compte de connexion
NoSite	Numérique	Identifiant du Site
IDPrivileges	Numérique	Identifiant privilèges

4.2.e tblSites (Les sites)

Cette table possède quatre champs :

- le champ **NoSite** de type Numérique Entier long ;
- le champ **LibelleSite** de type Texte limité à soixante-quatre caractères ;
- le champ **IDCommune** de type Numérique Entier long ;
- le champ **DateOuverture** de type Date.

La clé primaire est affectée aux champs **NoSite** + **LibelleSite**

NoSite est indexé sans doublon et LibelleSite est indexé avec doublons.

Nom du champ	Type de données	Description
NoSite	Numérique	Identifiant
LibelleSite	Texte	Libellé du Site
IDCommune	Numérique	Ville d'implantation
DateOuverture	Date/Heure	Date de mise ne activité du site

4.3 Données des tables

Pour simplifier les choses pour le développeur, j'ai adopté intentionnellement une convention de nommage pour les tables qui contiennent des données prédéfinies avec un caractère **underscore** (_).

Rappel
Celles qui possèdent un astérisque(*) sont des tables facultatives puisqu'elles ne contiennent que les régions, les départements et les communes, et concernent plus particulièrement ce projet. Celles qui possèdent deux astérisques(**) sont des tables directement liées au projet si l'on s'en réfère à l'aspect métier de ce dernier.

Toutes les autres tables sont dédiées à recevoir des données saisies par les utilisateurs. Ce seront ces tables qui seront remises à blanc, si toutefois l'utilisateur en fait la demande au chargement du projet. Nous verrons cela un petit peu plus bas dans le processus d'utilisation des formulaires.

Exemple de données dans les tables

4.3.a Liste des départements (tbl_ListeDepartements)

IDDepartement	Departement	IDRegion
01	Ain	23
02	Aisne	20
03	Allier	3
04	Alpes-de-Haute-Provence	22
05	Hautes-Alpes	22
06	Alpes-Maritimes	22
...	...	...

4.3.b Liste des régions (tbl_ListeDesRegions)

IDRegion	Region
1	Alsace
2	Aquitaine
3	Auvergne
4	Basse-Normandie
5	Bourgogne
...	...

Liste des communes (tbl_ListeDesCommunes)

IDCommune	Code INSEE	Commune	Code Postal	IDDepartement
447	98611	ALO	98610	98
603	98711	ANAA	98760	98
1307	98712	ARUE	98701	98
1308	98713	ARUTUA	98761	98
3007	98801	BELEP	98811	98
4276	98714	BORA BORA	98730	98
4510	98802	BOULOUPARI	98812	98
4524	98803	BOURAIL	98870	98
5855	98804	CANALA	98813	98
10485	98805	DUMBEA	98837	98
11481	98715	FAAA	98704	98
11508	98716	FAKARAVA	98790	98
...	...	...	...	...

4.3.c Liste des types de contrats (tbl_TypesContrat)

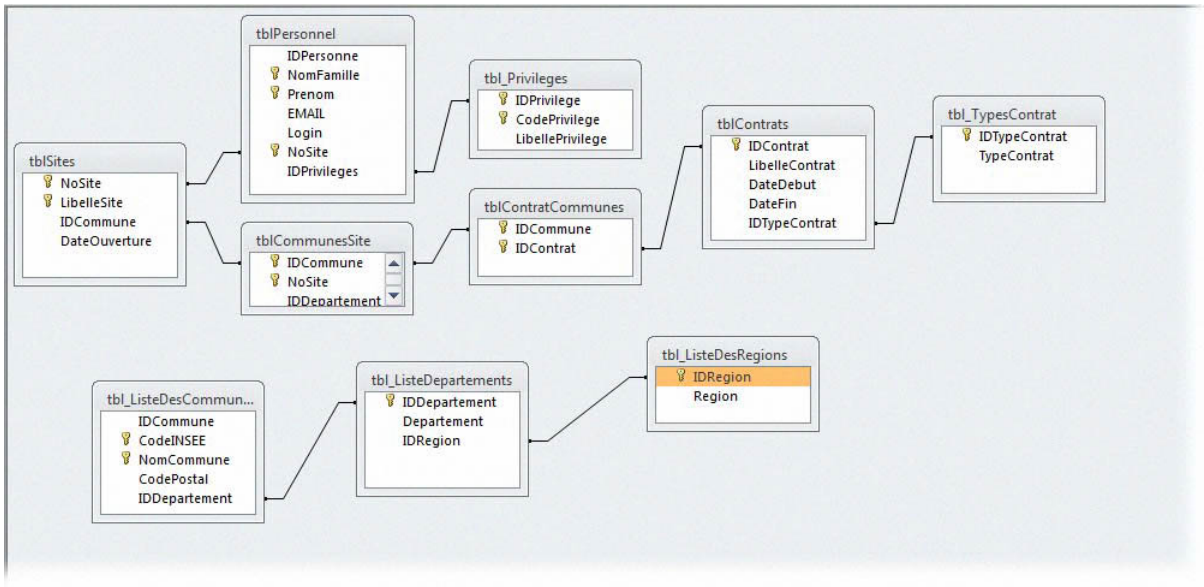
IDTypeContrat	Type de contrat
1	Contrat communal
2	Contrat intercommunal
3	Contrat fictif

4.3.d tbl_Privileges

IDPrivilege	Code Privilège	Libellé Privilèges
1	ADMR	Administrateur Régional
2	ADMS	Administrateur du Site
3	USERA	Utilisateur Agence
4	GUEST	Invité ou Externe

5 Le schéma relationnel de l'application

Bien qu'il ne soit pas obligatoire d'en élaborer un, le schéma relationnel a été mis en place dans ce projet. Il est représenté par l'illustration ci-dessous :



Cela permet, en tout cas ici, de mieux appréhender la structure...
Vous pouvez remarquer qu'il y a deux jeux de tables distincts :

— un jeu de tables concernant **les données utilisateur** ;
— un jeu de tables concernant uniquement les **communes**, leur **département** et leur **région**.

6 Conception des formulaires

Il existe autant de formulaires qu'il y a d'étapes.

Dans l'exemple présenté ici, je me suis limité à cinq écrans qualifiés à proprement parler de saisies, auxquels j'ai ajouté un formulaire d'accueil et un formulaire final récapitulatif.

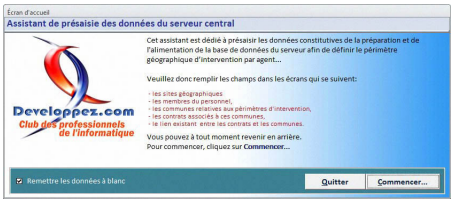
Ce projet comporte donc **onze formulaires** :

— le premier et le dernier sont les formulaires de début et de fin ;
— trois d'entre eux possèdent un sous-formulaire ;
— un autre est un sous-formulaire de type pop-up ;
— deux autres sont des formulaires simples.

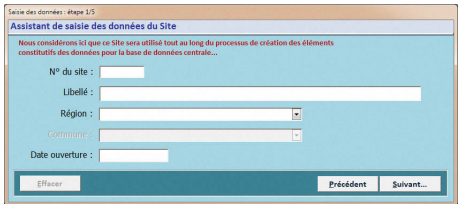
6.1 Les différents formulaires (en mode utilisation)

Dans cet assistant, parmi les cinq formulaires de saisie, se trouvent des formulaires dits « **simples** ». Les autres seront élaborés avec une structure **Parent/Enfant**, autrement dit, un formulaire principal et un sous-formulaire, ce dernier possédant une **structure continue**.

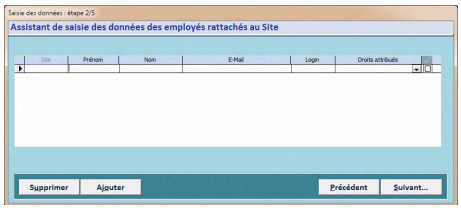
6.1.a Le formulaire d'accueil



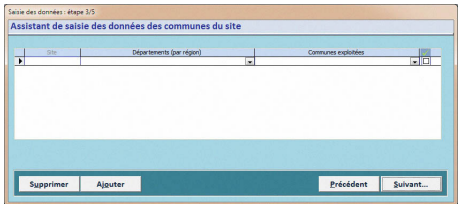
6.1.b Le formulaire de saisie du site



6.1.c Le formulaire de saisie du personnel

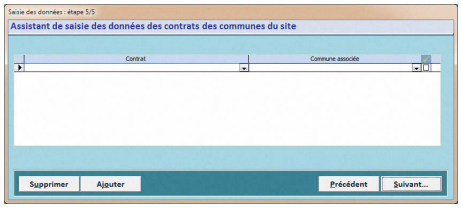


6.1.d Le formulaire de saisie des communes

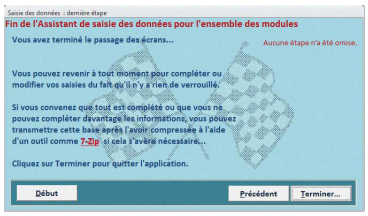


6.1.e Le formulaire de saisie des contrats

6.1.f Le formulaire de saisie de l'association Contrat-Communes



6.1.g Le formulaire de fin



7 Description structurelle des formulaires

7.1 La source de données formulaires

Tous les formulaires dédiés à la saisie se voient affecter leur source de données dynamiquement par code. Il est rappelé ici que l'objectif de ce tutoriel est de vous apprendre à minimiser l'écriture du code d'une part, et de faire en sorte que chaque feuille de code représentée par la classe des formulaires soit la **plus générique possible**.

Considérant que pratiquement tous les écrans restent identiques selon leur type (*simple ou continu*), il était important de faire en sorte que la source de données bénéficie des mêmes privilèges.

Vue éclatée des formulaires (description des contrôles)
Pour faciliter la mise en œuvre des formulaires, les paragraphes qui vont suivre vous présentent une photographie du formulaire en **Mode création** tel qu'il est vu à l'écran lorsqu'il s'agit d'un formulaire simple, et en **vue éclatée** lorsqu'il s'agit d'un sous-formulaire.
En effet, du fait que certains contrôles sont volontairement cachés en mode formulaire, il est plus aisé de se rendre compte de la façon dont l'ensemble des contrôles a été agencé et disposé.

7.2 Les formulaires simples

Ils sont au nombre de quatre :

deux formulaires sans données saisies et deux avec données saisies par l'utilisateur.

7.2.a Formulaire sans données utilisateur

Ces formulaires ne sont pas structurés de la même façon.

Ils sont essentiellement composés de contrôles ***étiquette*** informative et de boutons de navigation.

Ces deux formulaires sont :

— le formulaire d'accueil : **frmStart** ;

— le formulaire de fin : **frmEnd**.

7.2.b Formulaire avec données utilisateur

Ces formulaires sont tous structurés de la même façon.

Ils possèdent tous un contrôle *étiquette* permettant d'affecter dynamiquement le contenu du titre qui est alimenté directement au moment du chargement.

Juste au-dessous de celui-ci se trouvent les contrôles des champs de la source.

Dans la partie basse de chacun d'eux, se trouvent quatre boutons de commande :

— un bouton **Supprimer** qui recevra la légende **Sauvegarder** en cours de saisie ;

— un bouton **Ajouter** qui recevra la légende **Annuler** en cours de saisie ;

— un bouton intitulé **Précédent** qui permet de revenir à l'écran précédent ;

— et un bouton **Suivant** qui permet de passer à l'écran suivant.

Au niveau des propriétés

Vous définirez les propriétés ci-après comme suit :

— **Cycle** à « **Enregistrement en cours** » ; cette propriété permet de spécifier ce qui se produit lorsque vous appuyez sur la touche **TAB** pendant que le dernier contrôle d'un formulaire dépendant est activé ;

— **Menu Contextuel** à « **Non** » ; cette propriété permet de spécifier s'il doit être affiché lorsque vous cliquez sur un objet d'un formulaire à l'aide du bouton droit de la souris ;

— **Type Recordset** à « **Feuille de réponse dynamique** » pour permettre de spécifier quel genre de jeu d'enregistrements est disponible pour le formulaire ;

— **Propriété Remarque** (*Tag*) définie à **Null** ou **NotNull** selon les champs pour lesquels vous souhaitez ne pas considérer la nullité comme bloquante.

Explications
Tous les champs des formulaires simples voient leur propriété **Tag** définie à **Null** ou **NotNull** :
cette propriété est utilisée dans une fonction qui contrôle le bon contenu des champs.

7.3 Les formulaires avec un sous-formulaire

7.3.a Les formulaires parents

Ils sont au nombre de trois...

Ces formulaires sont tous structurés de la même façon.

Ils possèdent tous un contrôle *étiquette* permettant d'affecter dynamiquement le titre, le sujet et juste au-dessous de ceux-ci, se trouvent des zones de texte cachées que j'ai intentionnellement coloriées en jaune avec une police de couleur rouge.

Attention
Ces champs voient leur propriété **Visible** dépendante de la constante **SHOW_HIDDEN_CONTROLS**.

En leur milieu se trouve un sous-formulaire pour lequel il n'y a volontairement pas de notion **Champs pères** et **Champs fils**.

Les sous-formulaires portent toujours le même nom (**sfrmDataWizard**), et ce, quel que soit leur formulaire **Parent**.

En dessous de leur sous-formulaire se trouvent deux contrôles **étiquette**, l'un spécifiant le nombre **d'enregistrements inscrits** dans le sous-formulaire, l'autre spécifiant par effet de **clignotement** que la ligne est **complète** ou **incomplète** dans le sous-formulaire.

Cette dernière étiquette clignote grâce à l'événement **Timer**.

Dans la partie basse de chacun des formulaires parents sont dessinés quatre boutons de commande :

— un bouton **Supprimer** qui recevra la légende **Sauvegarder** en cours de saisie ;

— un bouton **Ajouter** qui recevra la légende **Annuler** en cours de saisie ;

— un bouton **Précédent** qui permet de revenir à l'écran précédent ;

— et un bouton **Suivant** qui permet de passer à l'écran suivant.

7.3.b Les formulaires enfants (ou sous-formulaire)

Chacun des sous-formulaires possède dans la partie droite de sa **zone de détail**, une case à cocher dont la formule calcule en permanence (**donc au moment de la saisie**) le fait que la ligne est complétée ou non.

On considère que la ligne est complétée à partir du moment où la formule stockée dans le champ caché (*de couleur jaune*) nommé **txtCheckdata** renvoie bien le résultat attendu.

Ce champ possède une formule appellant une fonction écrite en VBA qui calcule **la somme** de la nullité des champs de données avec leur type associé via la fonction **Nz()**.

La fonction en elle-même retourne **0** ou **1** selon que le champ est nul ou non, et ce, quel que soit son type. Selon le nombre de champs où la valeur est requise, une certaine somme est attendue.

Ainsi, pour la table du personnel du site, l'ensemble des six champs est obligatoire ; cela implique que la formule doit renvoyer tout simplement **6**.

La case à cocher prendra la valeur **True** ou **False** selon ce résultat.

Exemple de formule pour un sous-formulaire :
la zone de texte cachée « **txtCheckdata** »

```
1  =(NullData([NomDuChamp1];4)+NullData([
       NomDuChamp2];10)+NullData([
       NomDuChamp3];10))
```

et la case à cocher associée « **chkCompleted** »

```
1  =([txtCheckdata]=3)
```

la fonction **NullData()** sera expliquée un peu plus loin dans ce tutoriel.

Dans la partie basse de chacun des sous-formulaires, au niveau du **pied de formulaire**, se trouvent des zones de texte cachées dont deux d'entre elles calculent en permanence le nombre d'enregistrements pour l'une et la valeur de la case à cocher pour l'autre.

Les autres zones de texte cachées reprennent selon la source de données, les champs utilisés pour accéder à l'enregistrement, et donc leur identifiant d'une part et leur libellé d'autre part ; les valeurs de ces champs sont utilisées au niveau du **formulaire parent** pour pouvoir afficher le contenu dans un message, notamment lorsqu'il s'agit de supprimer la ligne.

> **Remarque**
> Chacune des zones de texte cachées situées dans le pied de formulaire a une propriété **Visible** définie à **False** et une hauteur fixée à **0**.

7.3.c Au niveau des propriétés

Vous définirez les propriétés ci-après comme suit :

- **Cycle** à « **Tous les enregistrements** » ; cette propriété permet de spécifier ce qui se produit lorsque vous appuyez sur la touche **TAB** pendant que le dernier contrôle d'un formulaire dépendant est activé ;
- **Menu Contextuel** à « **Non** » ; cette propriété permet de spécifier s'il doit être affiché lorsque vous cliquez sur un objet d'un formulaire à l'aide du bouton droit de la souris ;
- **Type Recordset** à « **Feuille de réponse dynamique** » pour permettre de spécifier quel genre de jeu d'enregistrements est disponible pour le formulaire.

8 Description détaillée des formulaires

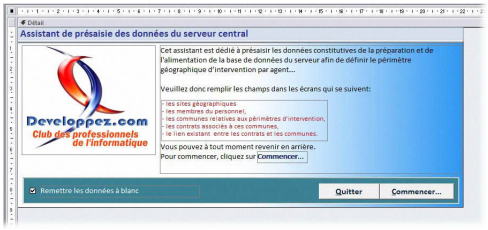
8.1 Le formulaire d'accueil

Ce formulaire se nomme **frmStart**.

Ce formulaire dépourvu de données est composé principalement de contrôles *étiquette* dédiés à présenter quelque peu l'interface, mais aussi :

- d'une **image** représentant un **logo** par exemple ;
- d'une **case à cocher** permettant de **remettre à blanc** les données ;
- et de **boutons** de commande, un pour **quitter**, l'autre pour **démarrer**.

Le titre de ce formulaire est statique du fait que c'est le formulaire d'accueil.



8.2 Le formulaire de saisie du site

Ce formulaire se nomme **frmDataWizard1**.

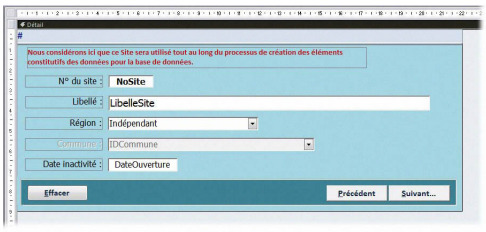
Il s'agit du formulaire qui contient les coordonnées du site.

Ce formulaire possède :

- un contrôle de type **étiquette** présentant succinctement l'objet du formulaire ;

— trois **zones de texte** et deux listes déroulantes ;

— et trois **boutons** de commande.



La première zone de texte définira le numéro du site, la deuxième, le libellé et la troisième, la date d'ouverture du site.

La zone de liste déroulante indépendante permet de lister les départements et leur région et possède le code événementiel associé pour pouvoir filtrer les communes correspondant à celui sélectionné.

La zone de liste des communes contient l'ensemble des communes issues de la table via une instruction SQL filtrée sur le critère précité.

Dans la partie basse du formulaire se trouvent trois boutons :

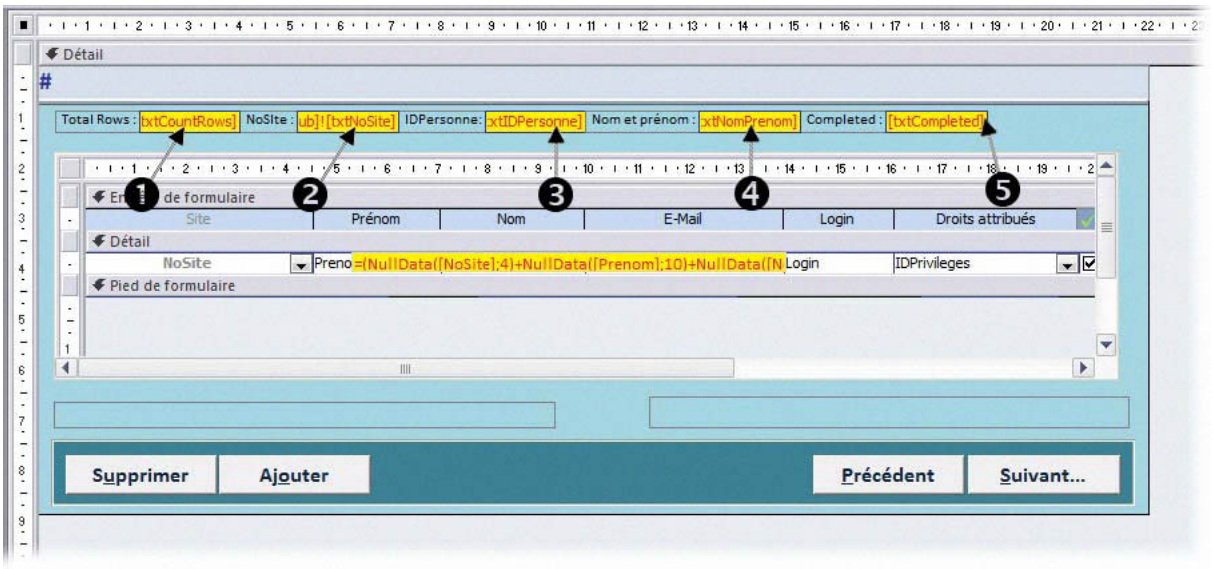
— un bouton intitulé **Effacer** qui permet de remettre à blanc la saisie en cours ;

— un bouton intitulé **Précédent** qui permet de revenir à l'écran précédent ;

— et un bouton **Suivant** qui permet de passer à l'écran suivant.

8.3 Le formulaire de saisie du personnel

Ce formulaire est nommé : **frmDataWizard2**.
Il s'agit du formulaire permettant de saisir les employés du site.



La partie haute du formulaire contient les cinq champs cachés qui récupèrent les valeurs du pied de formulaire du sous-formulaire.

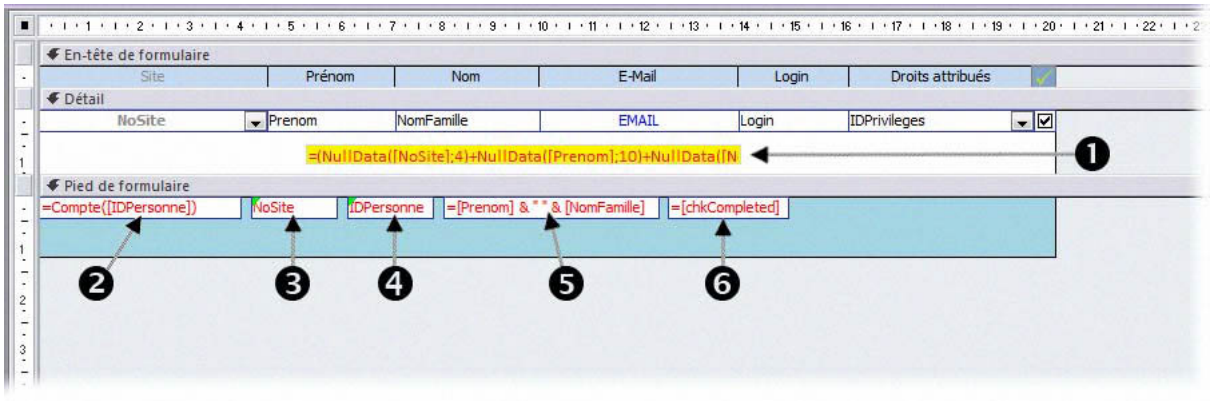
Juste en dessous du sous-formulaire, se trouvent les deux étiquettes qui calculent par programme le nombre d'employés enregistrés d'une part et si la ligne en cours de saisie dans le sous-formulaire est complète d'autre part.

N°	Champ	Formule ou Source	Description
❶	txtTotalRows	=[frmDataWizard2_sub] ![txtCountRows]	Valeur de txtCountRows*
❷	txtNoSite	=[frmDataWizard2_sub] ![txtNoSite]	Valeur de txtNoSite*
❸	txtIDPersonne	=[frmDataWizard2_sub] ![txtIDPersonne]	Valeur de txtIDPersonne*
❹	txtNomPrenom	=[frmDataWizard2_sub] ![txtNomPrenom]	Valeur de txtNomPrenom*
❺	txtRowCompleted	=[frmDataWizard2_sub] ![txtCompleted]	Valeur de txtCompleted*

* issues des champs du sous-formulaire de la zone pied de formulaire (les champs portent le même nom pour plus de commodité).

8.3.a Le sous-formulaire de saisie du personnel

Ce formulaire est nommé : **frmDataWizard2_sub**.
Il possède tous les champs de la table du personnel (*tblPersonnel*).



Dans la partie **En-tête** sont posées des étiquettes correspondant à chacun des champs de la table et dans la partie **Détail**, les champs eux-mêmes.
Dans la zone de détail se trouvent les champs cachés :

— la zone de texte « **txtCheckdata** » dont la formule associée est :

```
1  =(NullData([NoSite];4)+NullData([Prenom];10)+NullData([NomFamille];10)+NullData([EMAIL
       ];10)+NullData([LOGIN];10)+NullData([IDPrivileges];4))
```

— la case à cocher « **chkCompleted** » dont la formule associée est :

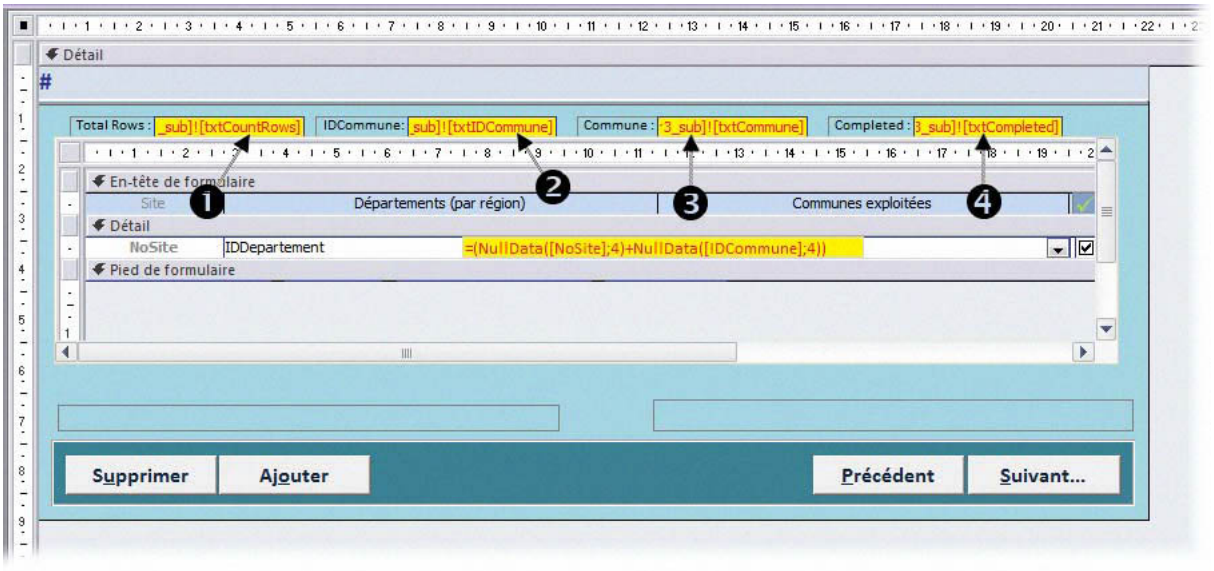
```
1  =([txtCheckdata]=6)
```

Dans le pied du formulaire se trouvent les champs cachés dont les formules permettent de comptabiliser le nombre d'employés, de renvoyer l'identifiant, le libellé (*ici nom et prénom*) et la complétion de l'enregistrement recevant le focus (*donc en cours*).

N°	Champ	Formule ou Source	Description
❶	txtCheckdata	=NullData(de chaque champ)	Calcule la somme des Nz()+Nz()+...
❷	txtCountRows	=Compte([IDPersonne])	Compte le nombre d'enregistrements
❸	txtNoSite	NoSite	Renvoie le numéro du site en cours
❹	txtIDPersonne	IDPersonne	Renvoie l'ID de l'employé en cours
❺	txtNomPrenom	=[Prenom] & " " & [NomFamille]	Renvoie le nom et le prénom en cours
❻	txtCompleted	=[chkCompleted]	Renvoie la valeur de la case à cocher

8.4 Le formulaire de saisie des communes

Ce formulaire est nommé : **frmDataWizard3**.
Il s'agit du formulaire permettant de sélectionner les communes exploitées par le site.



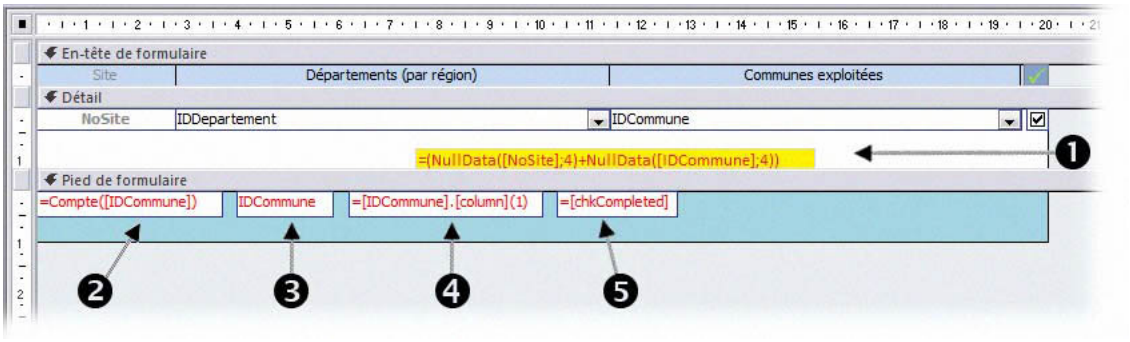
À l'instar du formulaire de saisie des employés, celui-ci possède quatre champs cachés dont les formules restent identiques à ceci près qu'elles ne pointent pas vers les mêmes champs, excepté bien entendu celle qui compte le nombre de lignes et celle qui retourne la complétion de l'enregistrement en cours.

N°	Champ	Formule ou Source	Description
❶	txtTotalRows	=[frmDataWizard3_sub] ![txtCountRows]	Valeur de txtCountRows*
❷	txtIDCommune	=[frmPerimeter3_sub] ![txtIDCommune]	Valeur de txtIDCommune*
❸	txtCommune	=[frmPerimeter3_sub] ![txtCommune]	Valeur de txtCommune*
❹	txtRowCompleted	=[frmDataWizard3_sub] ![txtCompleted]	Valeur de txtCompleted*

* issues des champs du sous-formulaire de la zone pied de formulaire (les champs portent le même nom pour plus de commodité).

8.4.a Le sous-formulaire de saisie des communes

Ce formulaire est nommé : **frmDataWizard3_sub**.
Il possède tous les champs de la table du personnel (***tblCommunesSite***).



Dans la partie **En-tête** seront posées des étiquettes correspondant à chacun des champs de la table et dans la partie **Détail**, les champs eux-mêmes.
Dans la zone de détail se trouvent les champs cachés :

— la zone de texte « **txtCheckdata** » dont la formule associée est :

```
1 =(NullData([NoSite];4)+NullData([IDCommune];4))
```

— la case à cocher « **chkCompleted** » dont la formule associée est :

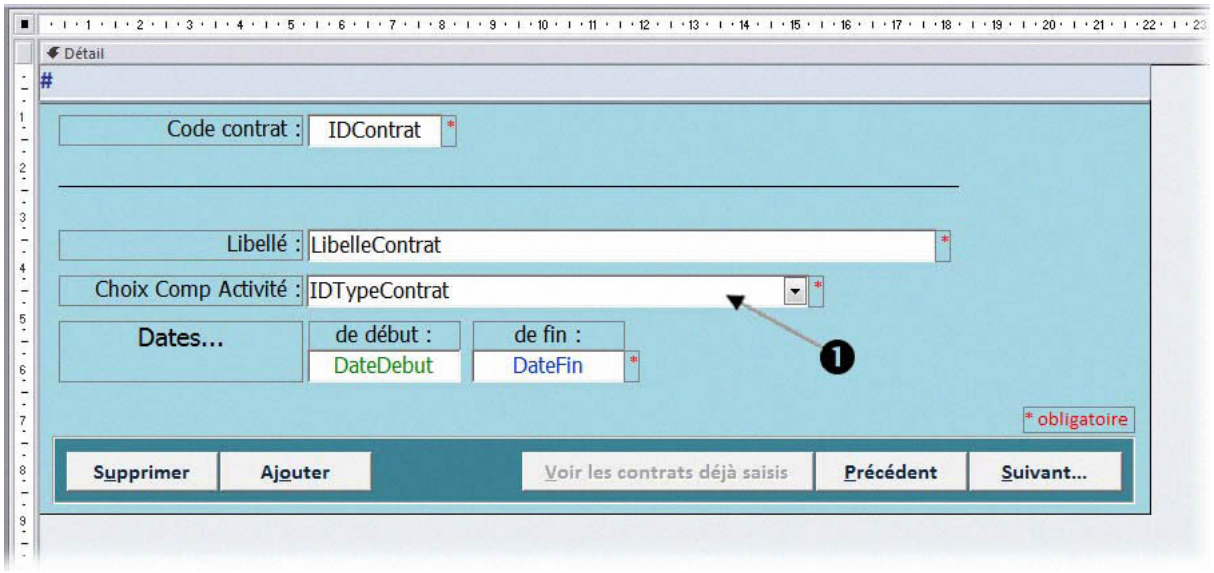
```
1 =([txtCheckdata]=2)
```

Dans le pied du formulaire se trouvent les champs cachés dont les formules permettent de comptabiliser le nombre de communes, de renvoyer l'identifiant, le libellé (*ici nom de la commune*) et la complétion de l'enregistrement recevant le focus (*donc en cours*).

N°	Champ	Formule ou Source	Description
❶	txtCheckdata	=NullData(de chaque champ)	Calcule la somme des Nz()+Nz()+...
❷	txtCountRows	=Compte([IDPersonne])	Compte le nombre d'enregistrements
❸	txtIDCommune	IDCommune	Renvoie l'ID de la commune en cours
❹	txtCommune	=[IDCommune].[column](1)	Valeur de txtNomCommune
❺	txtCompleted	=[chkCompleted]	Valeur de la case à cocher

8.5 Le formulaire de saisie des contrats

Ce formulaire est nommé : **frmDataWizard4_Popup**.
Il s'agit du formulaire qui contient les différents contrats.



Ce formulaire possède :

— quatre zones de texte et une liste déroulante pour les données propres au contrat ;
— cinq boutons de commande. La première zone de texte définira le numéro du contrat, la seconde, son libellé, et les deux dernières, respectivement la date de début et la date de fin du contrat. La zone de liste déroulante permet de lister les types de contrats. Dans la partie basse du formulaire se trouvent les cinq boutons :
— un bouton intitulé **Supprimer** qui permet de remettre à blanc la saisie en cours ;
— un bouton intitulé **Ajouter** qui permet d'ajouter un nouveau contrat ;
— un bouton intitulé **Voir les contrats déjà saisis** qui permet de visualiser tous les contrats déjà enregistrés ;
— un bouton intitulé **Précédent** qui permet de revenir à l'écran précédent ;
— et un bouton **Suivant** qui permet de passer à l'écran suivant.

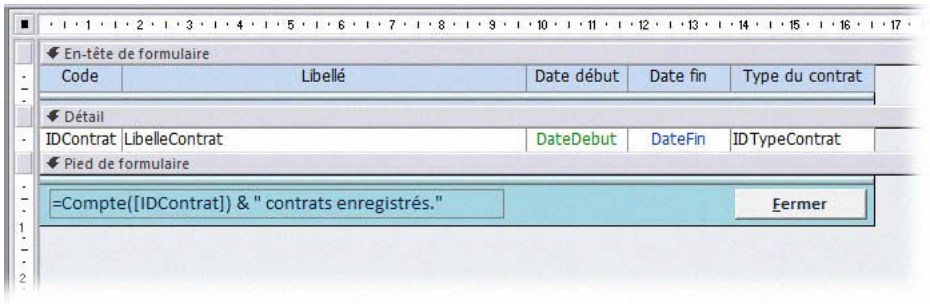
❶ Le champ **IDTypeContrat** représenté par une **Zone de liste déroulante** est attaché à la table **tbl_TypesContrat** à l'aide d'une requête :

```
1  SELECT tbl_TypesContrat.IDTypeContrat, tbl_TypesContrat.TypeContrat
2  FROM tbl_TypesContrat;
```

Pour plus de commodité, j'ai stocké cette clause SQL dans une requête nommée : **qry_CBOListeTypesDeContrat**.

8.5.a Le formulaire « popup » des contrats

Ce formulaire est nommé : **frmDataWizard4_Popup**
Il s'agit du formulaire qui contient l'ensemble des contrats que vous avez déjà saisis.



Il est à noter que le bouton d'accès à ce formulaire reste grisé tant que le nombre de contrats est inférieur ou égal à un.

Ce formulaire est construit en mode continu avec une structure **En-tête** et **Pied de formulaire**.

— Dans la partie **En-tête** seront posées des étiquettes correspondant à chacun des champs de la table et dans la partie **Détail**, les champs eux-mêmes.

— Dans le Pied du formulaire se trouvent :

1. Un champ avec une formule qui permet de comptabiliser le nombre de contrats existants ;
2. Et sur la droite, un bouton permettant de refermer la fenêtre.

Dans ce formulaire, tous les contrôles sont verrouillés là où la propriété **Arrêt tabulation** est définie à **False**.

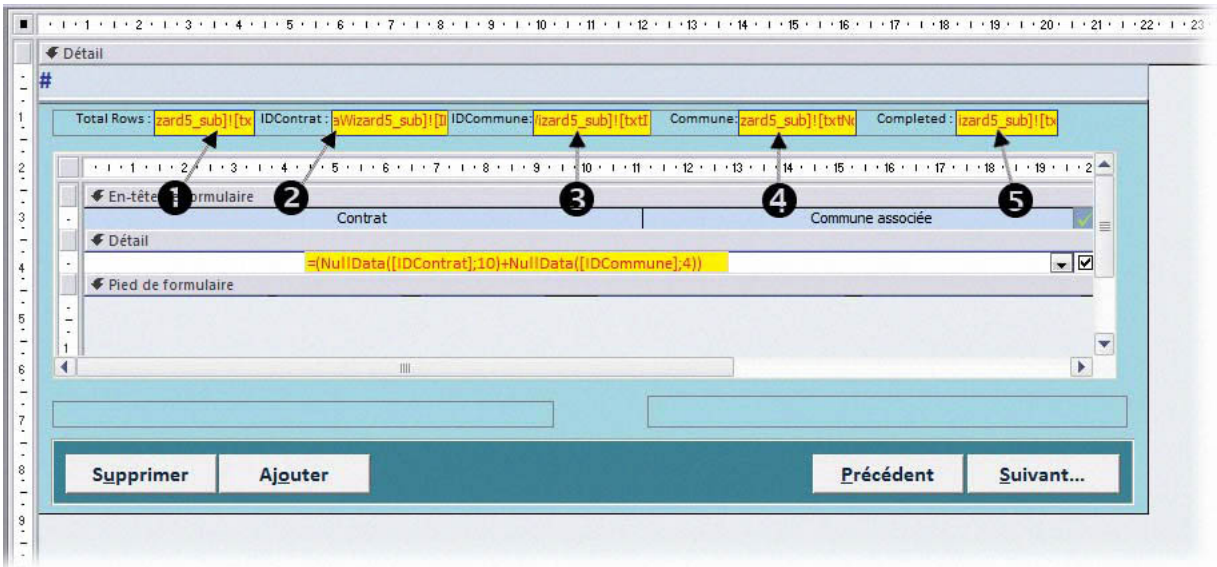
En haut à gauche et de façon **invisible**, se trouve un champ nommé ***txtFocus*** et qui **reçoit le focus** en permanence.

La formule du champ de comptabilisation est la suivante :

```
1  =Compte([IDContrat]) & " contrats enregistrés."
```

8.6 Le formulaire de saisie de l'association Contrat-Communes

Ce formulaire est nommé : **frmDataWizard5**.
Il s'agit du formulaire permettant de sélectionner les communes exploitées par le site.



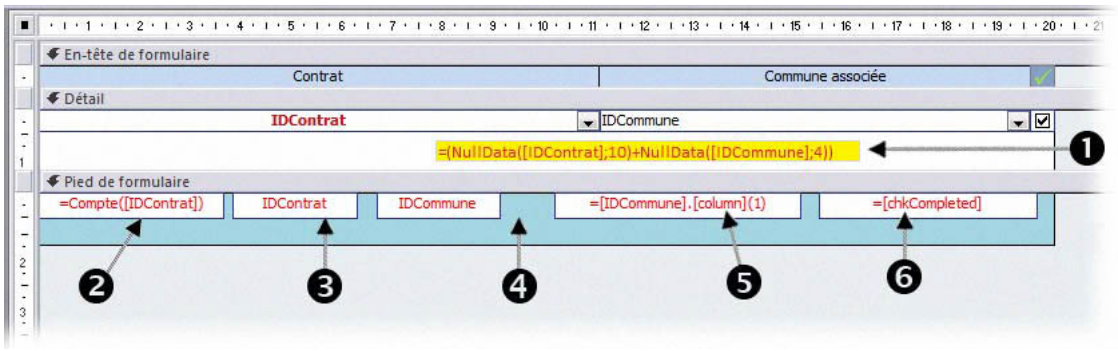
À l'instar du formulaire de saisie des employés, celui-ci possède quatre champs cachés dont les formules restent identiques à ceci près qu'elles ne pointent pas vers les mêmes champs, excepté bien entendu celle qui compte le nombre de lignes et celle qui retourne la complétion de l'enregistrement en cours.

N°	Champ	Formule ou Source	Description
❶	txtTotalRows	=[frmDataWizard5_sub]![txtCountRows]	Valeur de txtCountRows*
❷	txtIDContrat	=[frmPerimeter5_sub]![IDContrat]	Valeur de txtIDContrat*
❸	txtIDCommune	=[frmPerimeter5_sub]![txtIDCommune]	Valeur de txtIDCommune*
❹	txtNomCommune	=[frmPerimeter5_sub]![txtNomCommune]	Valeur de txtNomCommune*
❺	txtRowCompleted	=[frmDataWizard5_sub]![txtCompleted]	Valeur de txtCompleted*

* issues des champs du sous-formulaire de la zone pied de formulaire (les champs portent le même nom pour plus de commodité).

8.6.a Le sous-formulaire de saisie de l'association Contrat-Communes

Ce formulaire est nommé : **frmDataWizard5_sub**.
Ce formulaire possède tous les champs de la table du personnel (*tblContratCommunes*).



Dans la partie **En-tête** seront posées des étiquettes correspondant à chacun des champs de la table et dans la partie **Détail**, les champs eux-mêmes.
Dans la zone de détail se trouvent les champs cachés :

— la zone de texte « **txtCheckdata** » dont la formule associée est :

```
1 =(NullData([IDContrat];10)+NullData([IDCommune];4))
```

— la case à cocher « **chkCompleted** » dont la formule associée est :

```
1 =([txtCheckdata]=2)
```

Dans le pied du formulaire se trouvent les champs cachés dont les formules permettent de comptabiliser le nombre d'employés, de renvoyer l'identifiant, le libellé (*ici nom de la commune et N° du contrat*) et la complétion de l'enregistrement recevant le focus (*donc en cours*).

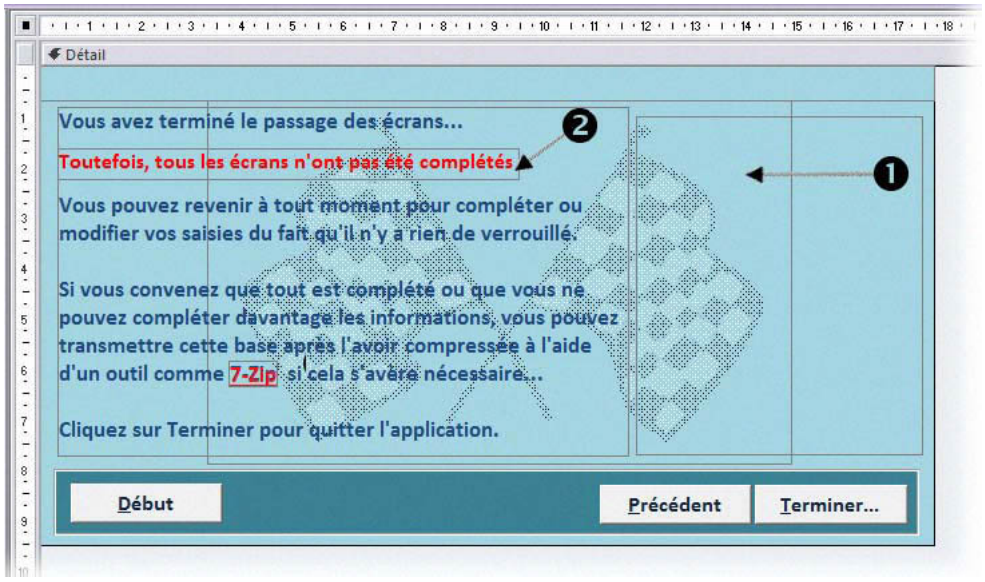
N°	Champ	Formule ou Source	Description
❶	txtCheckdata	=NullData(de chaque champ)	Calcule la somme des Nz()+Nz()+...
❷	txtCountRows	=Compte([IDContrat])	Compte le nombre d'enregistrements
❸	txtIDContrat	IDContrat	Renvoie l'ID du contrat en cours
❹	txtIDCommune	IDCommune	Renvoie l'ID de la commune en cours
❺	txtNomCommune	=[IDCommune].[column](1)	Renvoie le nom de la commune en cours
❻	txtCompleted	=[chkCompleted]	Renvoie la valeur de la case à cocher

8.7 Le formulaire de fin

C'est le dernier formulaire qui s'affiche pour résumer les étapes omises (*s'il y en a*) et pour clore l'application après confirmation.

Le projet a été élaboré pour être ouvert autant de fois que nécessaire de manière à ce que l'enrichissement des informations puisse se faire sur plusieurs jours.

Ce formulaire, tout comme le formulaire d'accueil, est composé essentiellement de contrôles type **étiquette** et trois boutons de commande.



Deux de ces étiquettes sont alimentées par code, les autres possèdent un texte statique.

Dans la partie basse du formulaire se trouvent les trois boutons :

— un bouton intitulé **Début** qui permet de revenir au début des étapes ;

— un bouton intitulé **Précédent** qui permet de revenir à l'écran précédent ;

— et un bouton **Terminer** qui permet de fermer l'application après confirmation.

N°	Champ	Formule ou Source	Description
❶	lblStepStatus	Code	Affiche la mention de complétion selon les étapes
❷	lblStepList	Code	Affiche la liste des étapes qui ont été omises

Retrouvez la suite de l'article de ***Jean-Philippe Ambrosino*** *en ligne : lien 88*

Virtualisation

Les blogs Open Source

Tutoriel vidéo : Quoi de neuf Docker vous explique en vidéo ce qu'est Docker

Developpez.com s'associe avec la chaîne YouTube « Quoi de neuf Docker » et de son auteur Nicolas De Loof pour vous proposer des vidéos autour du système de conteneur appelé Docker. L'originalité des vidéos proposées par Nicolas est de rendre accessible la technologie Docker à un public non spécialiste. C'est donc au travers d'astuces et de séquences humoristiques que Nicolas explique dans chaque séquence vidéo de 10 minutes environ les concepts autour de Docker.

Dans cette première vidéo, Nicolas De Loof tente de répondre à la question « mais bon sang c'est quoi Docker ? »

Voir la vidéo : lien 89

N'hésitez pas à laisser vos commentaires à la suite de cette discussion.

Bonne lecture vidéo et à très bientôt pour une prochaine vidéo.

Allez plus loin avec Docker : lien 90

Commentez la news de ***Mickael Baron*** *en ligne : lien 91*

Cloud Computing

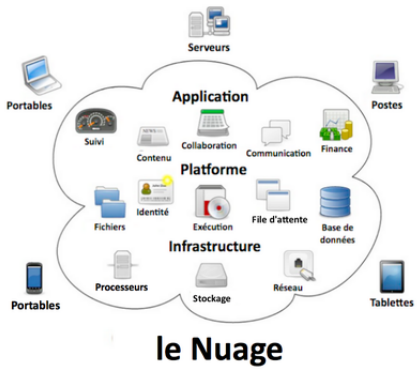
Les derniers tutoriels et articles

Découvrir et apprendre la solution de cloud computing : Click & Cloud

Click & Cloud est une solution PaaS (Platform as a Service) très pratique pour les développeurs. Elle offre une plate-forme supportant plusieurs langages de programmation dont Java, PHP, Ruby, Node.js, Python et les langages .NET. Elle propose aussi une intégration avec Docker, par des containers compatibles. Dans ce tutoriel, vous allez découvrir et apprendre cette solution très intéressante et facile à prendre en main.

1 Introduction

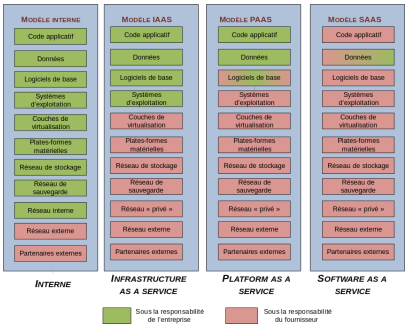
Le cloud computing est une approche informatique qui consiste à exploiter via Internet (ou tout autre réseau WAN) des ressources système et applicatives (serveurs, stockage, outils de collaboration et d'administration, etc.) distantes. Ces ressources distantes sont dites en *cloud* (dans le nuage).



Le cloud computing se présente comme une offre de services regroupés en trois catégories :

- **IaaS** : Infrastructure as a Service est l'offre de bas niveau qui consiste à offrir la couche matérielle (caractéristiques serveurs : processeurs, mémoire, stockage, réseau) sans le système d'exploitation, ni les les applications ;
- **PaaS** : Platform as a Service est l'offre incluant le PaaS, à laquelle s'ajoute le système d'exploitation et des outils d'administration. Elle se distingue des solutions des offres d'hébergement Web classiques, par sa modularité et son élasticité à s'adapter aux besoins, avec une possibilité de consommer à la carte ;
- **SaaS** : Software as a Service est une offre complète qui propose la couche applicative prête à l'emploi.

Les possibilités offertes par chaque offre se récapitulent comme suit :



Click & Cloud est une offre de type PaaS qui se veut aussi prête à l'emploi pour les développeurs d'applications Web. C'est une solution complète, robuste et fiable qui propose une facilité de prise en main, de configuration, d'administration et de déploiement d'applications.

Dans ce tutoriel vous allez découvrir Click & Cloud et apprendre à gérer vos projets de développement.

2 Présentation générale de Click & Cloud

2.1 Qui est le fournisseur de cette solution ?

Click & Cloud est une solution fournie par Owentis, l'un des leaders européens des solutions de cloud computing.

L'offre Click & Cloud est un environnement hébergé en France et cible, en priorité, les développeurs et les entreprises françaises.

Toutefois, en tant que solution bâtie selon les standards internationaux, tout le monde pourrait être satisfait par la qualité des services proposés.

2.2 Quelles sont les caractéristiques générales de Click & Cloud ?

Click & Cloud présente les caractéristiques générales suivantes :

- **langages supportés** : Java, PHP, Ruby, Node.js, Python et les langages .NET ;
- **intégration Docker** : compatibilité des containers avec Docker ;
  - standard évolué des packages applicatifs,
  - bibliothèques d'applications prêtes à l'emploi ;
- **serveurs d'applications** : Tomcat, Glassfish, Jetty, Apache et Nginx ;
- **système de gestion de base de données (SGBD)** :
  - **SQL** : MariaDB, PostgreSQL ets MySQL,
  - **No SQL** : MongoDB et CouchDB ;
- séparation des containers entre les bases de données et les serveurs d'application.

Les composants supportés sont donc :

Serveurs d'applications :

- Tomcat 6 & 7
- TomEE
- Jetty
- Glassfish with connection pools
- Apache
- NGINX
- Elastic VDS

Support JVM-based :

- Clojure
- JRuby
- ColdFusion
- Groovy
- Scala

SGBD :

- MySQL
- MariaDB
- PostgreSQL
- MongoDB
- CouchDB

Versions JAVA :

- JDK 6
- JDK 7
- JDK 8

Versions PHP :

- PHP 5.3
- PHP 5.4
- PHP 5.5

Versions Ruby :

- Ruby 1.9.2
- Ruby 1.9.3
- Ruby 2.0.0
- Ruby 2.1.1

Versions Python :

- Python 2.7
- Python 3.3
- Python 3.4

Versions Node.js :

- Node.js 0.10

2.3 Quelles sont les principales fonctionnalités offertes ?

Les fonctionnalités principales offertes par Click & Cloud sont :

- gestion simplifiée par des outils simples pour gérer les applications et l'environnement ;
- administration à partir d'une application mobile iOS ;
- compatibilité avec Docker ;
- équilibrage de charge HTTP et TCP ;
- haute disponibilité de la plate-forme pour la fiabilité des applications, via des techniques de *clustering* et de réplication ;
- migration à chaud d'application entre serveurs ;
- domaines personnalisables ;
- intégration multiples domaines ;
- support de domaines swap ;
- réplication de sessions ;
- sécurité HTTPS avec certificats privés SSL ;

— possibilité de mapper et/ou paramétrer une adresse IP publique ;
— accès FTP/FTPS, API, SSH ;
— possibilité Memcached ;
— compte de travail collaboratif ;
— journalisation du serveur d'application ;
— outils d'administration des bases de données ;
— reconfiguration du serveur Web ;
— statistiques et rapports ;
— sauvegarde des environnements partagés ;
— possibilité de cloner un environnement ;
— démarrage / arrêt des environnements ;
— configuration des paramètres d'application ;
— gestion du cycle de vie des applications ;
— planification des tâches ;
— intégration continue ;
— installation de l'application en un seul clic ;
— et bien d'autres.

3 Démarrer avec Click & Cloud

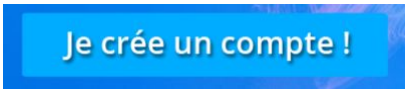
3.1 Quelle est la logique d'exploitation ?

Exploiter la plate-forme Click & Cloud obéit à une logique simple. Elle se résume aux quatre points suivants :

1. Créer son compte (si on n'en a pas encore) ; la plate-forme offre la possibilité de créer un compte gratuitement avec un accès gratuit de 30 jours sur cette plate-forme.
2. Créer son environnement.
3. Déployer son application.

3.2 Comment créer son compte Click & Cloud et se connecter ?

Pour créer son compte Click & Cloud, il faut cliquer sur le bouton « **je crée un compte** » qui est présent sur presque toutes les pages :



Il faut juste entrer son adresse mail :

Puis il faut patienter quelques secondes, pour la création du compte :

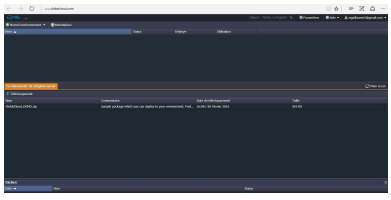
Le compte est créé et un émail avec le mot de passe est envoyé à l'adresse indiquée.

Il faut maintenant récupérer le mot de passe dans le mail reçu et se connecter au site :

Après déconnexion, pour une reconnexion à partir de la page d'accueil (lien 92), il suffit de cliquer sur « J'ai un compte » :

3.3 Comment se présente l'interface principale ?

Une fois connecté, vous êtes dirigé à la page principale qui se présente comme ceci :



— **Dans le coin supérieur gauche**, vous avez les boutons pour créer votre environnement ou aller au MarketPlace, qui est abordé plus tard.
— **Dans le coin supérieur droit**, vous allez à la configuration des paramètres du compte, qui fait l'objet d'un prochain chapitre.
— **Dans la zone au centre**, vous aurez la liste des environnements que vous aurez créés.

— **La deuxième zone centrale** contiendra, sous forme d'onglets, les différentes pages que vous aurez ouvertes. Vous pourrez, par exemple, ouvrir le gestionnaire de déploiement, les paramètres du compte et le MarketPlace. Il est possible de garder ces trois pages simultanément ouvertes, et de naviguer entre elles via les onglets qui seront automatiquement disponibles dans cette zone.

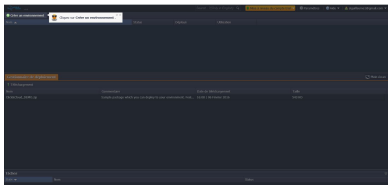
— **La zone totalement en dessous**, contiendra la liste des tâches en cours, terminées, ou échouées.

Il est possible de redimensionner chacune de ces zones à souhait.

3.4 Comment créer et paramétrer son environnement ?

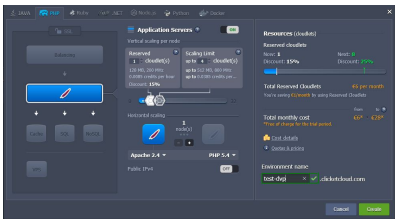
3.4.a Présentation générale de la création d'environnement

À la toute première connexion, vous êtes invités à créer un environnement :



Il convient de remarquer qu'un package par défaut est configuré et prêt à être déployé.

Cliquez sur « **créer un environnement** » :



Vous avez la fenêtre pour créer votre environnement, en remarquant les points suivants :

— **en onglets** : le choix du langage ;

— **dans le panneau à gauche** : le choix de la sécurité (SSL ou pas), l'activation de l'option d'équilibrage de charge, le choix du serveur d'application Web, l'activation du cache, le choix du type de SGBD et du système d'exploitation du serveur ;

— **dans la zone centrale** : les performances système (processeur, mémoire vive, réseau) à réserver pour l'environnement ; il faut noter que ces performances, mesurées en unité *cloudlet*, impactent directement le coût ; c'est aussi dans cette zone que vous choisissez le serveur d'application et la version du langage ;

— **dans le panneau à droite** : le récapitulatif sur les performances système de votre environnement et son coût total.

Dans l'exemple suivant, il va s'agir de créer un environnement avec les caractéristiques suivantes :

— nom de l'environnement : ***test-dvp*** ;

— sécurité TLS : HTTPS ;

— équilibrage de charge activé ;

— cache activé ;

— langage :PHP 7 ;

— serveur d'application : Apache ;

— SGBD :MySQL ;

— système d'exploitation du serveur : Linux CentOS 7.1 ;

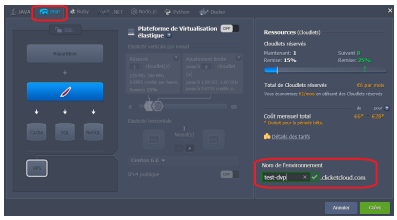
— performances système élastiques verticalement :

  — mémoire vive : 512 Mo extensibles à 1 Go,

  — processeur : 800 MHz extensibles à 1,60 Go.

3.4.b Étape 1 : choix du langage et définition du nom de l'environnement

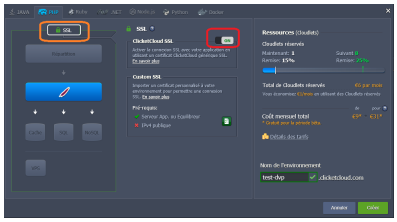
Dans l'interface qui se présente, il faut commencer par choisir l'onglet PHP (remarquez que selon les onglets, les options possibles dans la page diffèrent) :



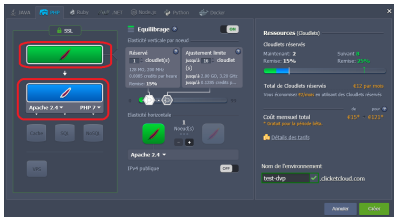
Le nom de l'environnement va constituer une URL par laquelle votre application sera directement disponible.

3.4.c Étape 2 : activation de la sécurité SSL

Pour activer la sécurité SSL sur l'environnement, il faut cliquer sur le bouton grisé SSL (avec l'icône du cadenas ouvert), puis vous passez la valeur **ClicketCloudSSL** à **ON** en cliquant sur le bouton présenté, comme suit :



Vous avez la possibilité d'importer votre propre certificat SSL.



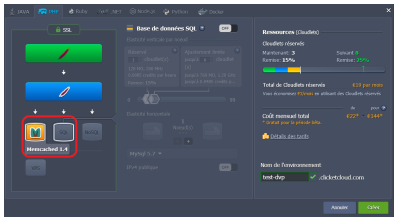
3.4.d Étape 3 : activer l'équilibrage de charge et choisir le serveur d'application

Pour le faire, il faut cliquer sur le bouton grisé « **Répartition** », ensuite, il faut choisir le serveur d'application Apache 2.4 dans la liste déroulante proposée, puis la version PHP 7 (par défaut, vous avez la version PHP 5.4) :

Remarquez qu'après avoir activé l'équilibrage de charge, le prix total (affiché dans la zone de droite) est passé à la hausse.

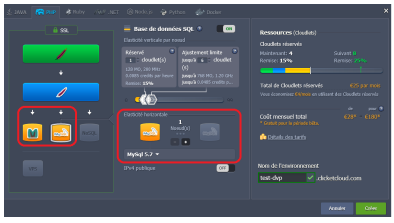
3.4.e Étape 4 : activation du cache

Pour activer le cache, il suffit de cliquer sur le bouton grisé « cache » :



3.4.f Étape 5 : choix du SGBD

Cliquez sur le bouton grisé « **SQL** », puis dans la liste déroulante choisissez MySQL 5.7 :

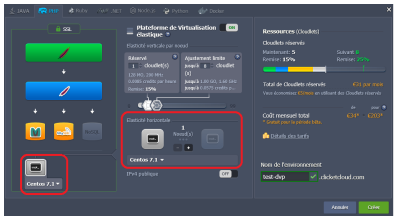


Remarquez que pour l'élasticité horizontale, vous pouvez augmenter le nombre de nœuds de base de données. Il est également possible d'activer un accès par adresse IP publique.

Click & Cloud vous donne la possibilité d'avoir plusieurs bases de données de types différents dans un même environnement. Par exemple, il est possible d'avoir une base de données SQL, MySQL et une autre base de données NoSQL, MongoDB.

3.4.g Étape 6 : choix du système d'exploitation

Pour ce faire, cliquez sur le bouton grisé « VPS » et dans la liste déroulante, choisissez « CentOS 7 ».



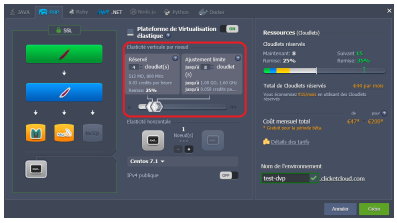
Remarquez que pour l'élasticité horizontale, vous pouvez augmenter le nombre de nœuds. Il est également possible d'activer un accès par adresse IP publique.

3.4.h Étape 7 : paramétrer les performances système

L'unité de mesure des performances système utilisée par Click & Cloud, c'est le **cloudet**.

Un **cloudet** équivaut à un système de 128 Mo de mémoire vive et 400 MHz de puissance processeur. Pour avoir les performances souhaitées, il faut 04 **cloudets** réservés et extensibles à 08 **cloudets**.

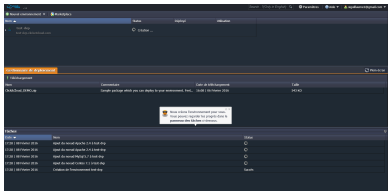
Ce sont les valeurs à indiquer dans la zone centrale :



3.4.i Étape 8 : exécution de la création

Tous les paramètres sont définis. Il faut maintenant cliquer sur le bouton « **Créer** » dans le coin inférieur droit.

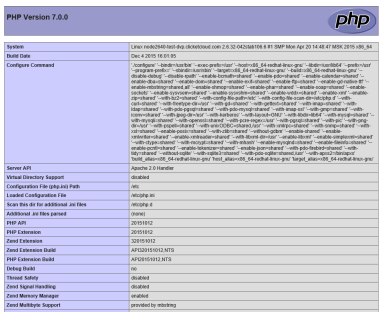
Vous pouvez observer l'exécution de la création dans la zone de suivi des tâches :



L'environnement est maintenant créé.

Vous avez également reçu deux mails de notification : un mail pour vous notifier de la création effective de l'environnement et un autre mail avec les paramètres de connexion au serveur de bases de données, ainsi que les comptes FTP et SSH pour se connecter aux nœuds.

À ce stade, quand on lance dans le navigateur cette URL : lien 93, on a la page par défaut de phpinfo() :

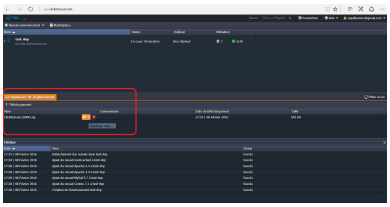


La prochaine phase va consister à déployer l'application **HelloWorld** préchargée.

3.5 Comment déployer une application ?

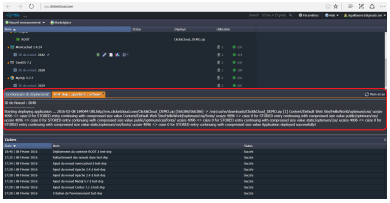
L'environnement créé dans la section précédente est venu avec une application préchargée, qu'il faut maintenant déployer.

Il faut cliquer sur le bouton de déploiement dans le gestionnaire de déploiement (le bouton apparaît sur survol) :



Il faut ensuite cliquer sur le nom de l'environnement, la fenêtre de définition du contexte apparaît :

Il est possible de lancer le déploiement sans contexte. On clique donc sur « Déployer » sans indiquer de contexte. L'application va s'installer avec succès à la racine du site :



On vérifie que l'application est déployée à la racine de notre site :

4 Paramètres du compte

4.1 Qu'est-ce que les paramètres de compte ?

Les paramètres de compte accessibles depuis le bouton de menu, dans le coin supérieur droit, permettent de définir la configuration générale des accès à la plate-forme associée au compte de l'utilisateur. Ces paramètres sont appliqués, par défaut, à tous les environnements créés sous ce compte.

Il s'agit notamment des paramètres d'accès SSH et ceux de la collaboration.

4.2 Comment paramétrer les clés SSH ?

Click & Cloud vous permet d'ajouter deux types de clés SSH :

— **les clés privées** : vous pouvez ajouter des clés privées GIT pour accéder aux dépôts présents sur ce SVN :

— **les clés publiques** : elles vous permettent d'accéder à votre plate-forme Click & Cloud à partir d'un client SSH :

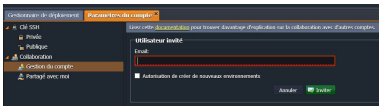
4.3 Comment configurer la collaboration ?

La collaboration consiste à avoir un environnement partagé entre plusieurs personnes, par

exemple, une équipe de développeurs travaillant sur un projet commun.

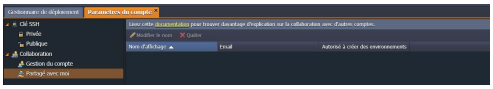
Il est donc possible d'ajouter d'autres personnes sur sa plate-forme, et/ou de voir les invitations faites par d'autres personnes.

Pour inviter, un collaborateur, cliquez sur « **Collaboration** », puis sur « **Gestion du compte** » :



> ⚠ Si vous cliquez sur l'option « **Autorisation de créer de nouveaux environnements** », la personne invitée pourra créer de nouveaux environnements sur votre profil, avec les impacts de coût que cela induira.

L'interface « Partagé avec moi » donne la liste des plates-formes (profils) que d'autres personnes auront partagées avec vous.



5 Zoom sur quelques possibilités importantes de la plate-forme Click & Cloud

5.1 Qu'est-ce que l'élasticité verticale ?

L'élasticité verticale est un élément de la haute disponibilité qui consiste à donner une capacité d'extension aux performances système. Concrètement cela correspond à ajouter des cloudets à sa réserve.

5.2 Qu'est-ce que l'élasticité horizontale ?

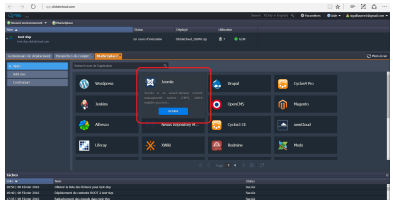
L'élasticité horizontale est un élément de la haute disponibilité qui consiste à configurer plusieurs nœuds pour la couche applicative : serveurs d'application et/ou serveurs de base de données.

5.3 Qu'est-ce que le MarketPlace ?

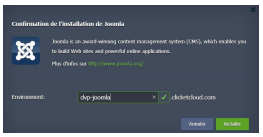
C'est une plate-forme où il est proposé un ensemble de contenus que vous pouvez installer et configurer dans votre environnement. Il existe trois types de contenus :

- des applications Web disponibles en libre accès telles que des CMS, des forums, etc.
- des addons ;
- des containers.

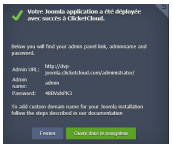
Par exemple, pour installer un CMS comme Joomla, vous cliquez sur « MarketPlace », puis « Apps », après vous survolez « Joomla » et vous cliquez sur « Installer » :

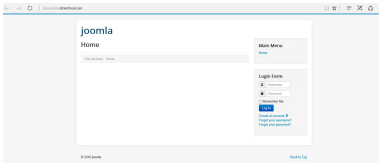


Vous entrez le nom du nouvel environnement où Joomla sera déployé, puis vous cliquez « Installer » :



Une fois que l'« App » est déployée, il est possible de le visualiser dans le navigateur, avec les paramètres définis par l'assistant d'installation :





6 Gestion d'un environnement

6.1 Vue générale de l'administration d'un environnement

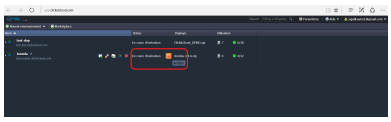
Click & Cloud propose un panel de tâches pour gérer aisément ses environnements. Parmi ces tâches, on retrouve :

- redémarrer son environnement ;
- partager de façon granulaire un environnement ;
- modifier les paramètres prédéfinis ;
- surveiller les performances ;
- suivre les statistiques de consommation ;
- et bien d'autres tâches.

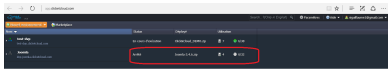
6.2 Comment redémarrer son environnement ?

Un environnement peut être arrêté pour des raisons de maintenance, par exemple.

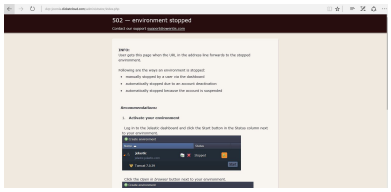
Dans l'illustration suivante, l'environnement dvp-joomla sera arrêté. Pour cela, il faut survoler l'environnement, et au niveau de la colonne « Status », on voit apparaître un bouton rouge avec un message de survol « Arrêter » :



Il faut cliquer sur le bouton « OUI » dans le message d'avertissement qui se présente, pour confirmer l'arrêt. Et ensuite patientez jusqu'à l'arrêt de l'environnement.

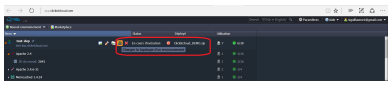


On va sur l'URL de l'environnement, et on constate qu'il est bien arrêté :



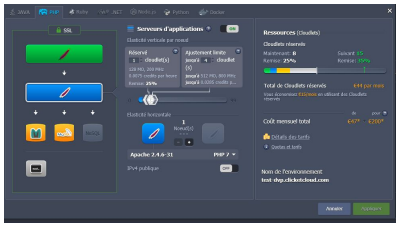
Pour redémarrer l'environnement, il suffit de cliquer sur le bouton « Démarrer » qui a remplacé le bouton « Arrêter » au même emplacement.

6.3 Comment modifier les paramètres définis pendant la création de l'environnement ?



Il est possible de modifier tous les paramètres définis pendant l'installation. Pour cela, il faut survoler l'environnement à modifier et cliquer sur le bouton sur lequel s'affiche « Changer la topologie » au survol :

On obtient la même interface de création de l'environnement :

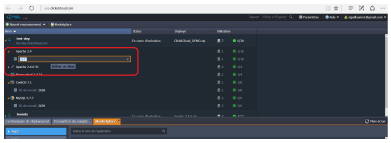


Toutes les valeurs peuvent être modifiées, sauf le nom de l'environnement, ce qui est normal.

Toutefois, on verra dans la section suivante qu'il est possible de donner un alias à son environnement.

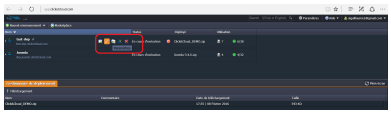
6.4 Comment définir les paramètres avancés d'un environnement ?

Il est possible d'apporter des détails simples ou avancés à votre environnement. Par exemple, vous pouvez donner un alias à votre environnement et aux différents nœuds pour leur donner un nom convivial.

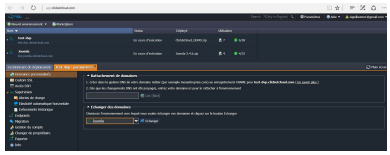


Mais des paramètres plus avancés, présentés dans les caractéristiques et les fonctionnalités peuvent être définis.

Pour cela, il faut survoler le nom de l'environnement à paramétrer et cliquer sur le bouton qui affiche « Paramètres » au survol :



Les paramètres à définir se présentent :



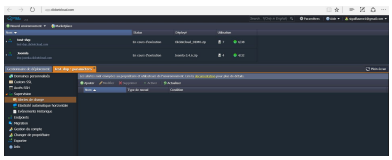
Avec ces paramètres, on voit comment il est possible de :

- lier un domaine (URL professionnelle, par exemple) à l'environnement qui a été créé, et dont l'URL est sur le domaine de clicet-cloud.com ;
- personnaliser sa sécurité SSL ;
- paramétrer ses accès SSH à l'environnement ;
- superviser son environnement en définissant des alertes ;
- utiliser d'autres ressources (endpoints) que celles de Click & Cloud ;
- migrer l'environnement sur un autre serveur ;
- gérer de façon granulaire le compte pour cet environnement spécifiquement ;
- changer le propriétaire de l'environnement ;
- exporter l'environnement, pour sauvegarder les paramètres, par exemple.

6.5 Comment surveiller les performances et la consommation ?

6.5.a Surveillance de la consommation

Il est possible de définir des alertes sur la consommation des ressources par son environnement. Pour cela, dans les paramètres de l'environnement, il faut cliquer sur « Supervision » puis sur « Alertes de charge » :



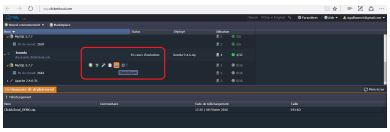
Il faut maintenant cliquer sur « Ajouter » pour définir une alerte, qui enverra un message lorsque la consommation processeur et mémoire d'un nœud Apache va excéder 75 % sur 5 mn :

L'alerte a pour nom « Surveillance Nœud Apache ».

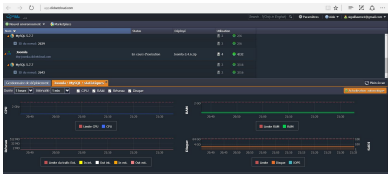
Il est possible de modifier, désactiver ou supprimer une alerte, à partir de la liste des alertes.

6.5.b Statistiques d'exploitation des ressources

Pour visualiser les statistiques d'exploitation des ressources par un nœud, il faut survoler le nœud et cliquer sur le bouton qui affiche « Statistiques », au survol :



Dans l'exemple ci-dessous, il s'agit des statistiques de consommation du nœud MySQL de l'environnement Joomla, qui a été précédemment créé :



7 Conclusion

Dans ce tutoriel, il a été question de découvrir la solution de PaaS Click & Cloud, et d'en apprendre l'installation et la configuration d'environnement.

Pendant votre pratique, vous verrez encore d'autres possibilités qu'offre cette excellente solution de cloud computing telle que la journalisation, la gestion granulaire des nœuds, etc.

Retrouvez l'article de ***Guillaume Sigui*** *en ligne : lien 94*

Liste des liens

Page 2
lien 1 : ... http://akrzemi1.developpez.com/tutoriels/c++/preconditions/partie-1/

Page 3
lien 2 : ... https://akrzemi1.wordpress.com/2012/01/11/concept-axioms-what-for/
lien 3 : ... http://akrzemi1.developpez.com/tutoriels/c++/preconditions/partie-1/

Page 5
lien 4 : ... http://home.roadrunner.com/~hinnant/stack_alloc.html
lien 5 : ... https://akrzemi1.wordpress.com/2012/08/12/user-defined-literals-part-i/#positive_double
lien 6 : ... http://rk.dl.pl/r/constrained_value
lien 7 : ... http://www.boost.org/doc/libs/1_53_0/libs/optional/doc/html/index.html
lien 8 : ... https://github.com/akrzemi1/Optional/

Page 6
lien 9 : ... http://akrzemi1.developpez.com/tutoriels/c++/preconditions/partie-2/

Page 8
lien 10 : ... http://doc.qt.io/qt-5/qsgnode.html

Page 9
lien 11 : ... http://www.kdab.com/
lien 12 : ... http://www.developpez.com/redirect/2406
lien 13 : ... http://kdab.developpez.com/tutoriels/qt-quick-opengl/01-presentation-generale/

Page 10
lien 14 : ... http://perl.developpez.com/cours/#TutorielsPerl6
lien 15 : ... http://rakudo.org/downloads/star/
lien 16 : ... http://www.developpez.net/forums/d1560445/autres-langages/perl/langage/sortie-officielle-perl-6-version-1-0-production/

Page 11
lien 17 : ... http://loic-guibert.developpez.com/tutoriels/git/get-started/fichiers/git-cheatsheet.pdf
lien 18 : ... http://git-scm.com/downloads
lien 19 : ... http://git-scm.com/
lien 20 : ... http://git-scm.com/book/fr/v2
lien 21 : ... http://www.alexgirard.com/git-book/
lien 22 : ... http://loic-guibert.developpez.com/tutoriels/git/get-started/fichiers/git-cheatsheet.pdf
lien 23 : ... http://git-scm.com/downloads/logos

Page 12
lien 24 : ... http://git-scm.com/book/fr/v2

Page 13
lien 25 : ... http://git-scm.com/book/fr/v2
lien 26 : ... http://www.git-scm.com/docs/git-remote
lien 27 : ... http://www.git-scm.com/book/fr/v2/Les-bases-de-Git-Travailler-avec-des-dépôts-distants

Page 16
lien 28 : ... http://loic-guibert.developpez.com/tutoriels/git/get-started/fichiers/git-cheatsheet.pdf

Page 17
lien 29 : ... http://git-scm.com/book/fr/v2
lien 30 : ... http://loic-guibert.developpez.com/tutoriels/git/get-started/

Page 18
lien 31 : ... http://tcuvelier.developpez.com/tutoriels/algo/introduction-algorithmes-structures-donnees/#LIII

lien 32 : ... http://khayyam.developpez.com/articles/algo/voyageur-de-commerce/genetique/

Page 19

lien 33 : ... https://youtu.be/qYIhA3O9Nz0
lien 34 : ... http://www.acm.org/press-room/news-releases/2015/knuth-godel-prizes-2015/view
lien 35 : ... http://www.developpez.net/forums/d1561114/general-developpement/algorithme-mathematiques/general-algorithmique/algorithme-quasipolynomial-l-isomorphisme-graphes/
lien 36 : ... https://www.phoronix.com/scan.php?page=news_item&px=Linux-4.2-Radeon-DRM-Pull
lien 37 : ... http://www.phoronix.com/scan.php?page=news_item&px=HSA-Offloading-GCC6-Hopes
lien 38 : ... http://www.developpez.com/actu/33310/Cplusplus-AMP-le-nouvel-outil-de-Microsoft-pour-la-conception-d-applications-paralleles-en-Cplusplus-utilisant-la-puissance-du-GPU/

Page 20

lien 39 : ... http://www.anandtech.com/show/7083/nvidia-to-license-kepler-and-future-gpu-ip-to-3rd-parties
lien 40 : ... http://blog.developpez.com/dourouc05/p12898/hpc/amd-se-lance-dans-le-hpc
lien 41 : ... http://www.developpez.net/forums/d1555625/general-developpement/algorithme-mathematiques/programmation-parallele-calcul-scientifique-haute-performance-hpc/coprocesseurs-calcul-heterogene/sc15-amd-lance-l-initiative-boltzmann/
lien 42 : ... http://blog.developpez.com/dourouc05/p12991/hpc/cntk-la-solution-de-microsoft-pour-lapprentissage-profond
lien 43 : ... http://www.nature.com/news/google-ai-algorithm-masters-ancient-game-of-go-1.19234
lien 44 : ... https://fr.wikipedia.org/wiki/Verre_de_spin

Page 21

lien 45 : ... http://dillgroup.stonybrook.edu/#/landscapes
lien 46 : ... http://www.developpez.net/forums/d1567063/general-developpement/algorithme-mathematiques/intelligence-artificielle/pourquoi-l-apprentissage-profond-reseaux-neuronaux-prometteurs/
lien 47 : ... http://zlib.net/
lien 48 : ... https://en.wikipedia.org/wiki/LZ77_and_LZ78
lien 49 : ... http://tcharles.developpez.com/Huffman/

Page 22

lien 50 : ... http://www.zlib.net/feldspar.html
lien 51 : ... http://mattmahoney.net/dc/dce.html#Section_523
lien 52 : ... http://www.7-zip.org/
lien 53 : ... http://tukaani.org/xz/
lien 54 : ... https://en.wikipedia.org/wiki/Arithmetic_coding
lien 55 : ... https://github.com/richgel999/lzham_codec
lien 56 : ... http://cyan4973.github.io/lz4/
lien 57 : ... http://www.developpez.com/actu/95088/Brotli-la-nouvelle-bibliotheque-de-compression-libre-et-open-source-de-Google-est-disponible-sur-le-canal-Canary/
lien 58 : ... http://www.radgametools.com/oodle.htm
lien 59 : ... https://cs.fit.edu/%7Emmahoney/compression/textdata.html
lien 60 : ... https://quixdb.github.io/squash-benchmark
lien 61 : ... http://richg42.blogspot.be/2016/01/compression-ratio-plots-of-two-zlib.html
lien 62 : ... http://www.developpez.net/forums/d1572574/general-developpement/algorithme-mathematiques/general-algorithmique/faut-mettre-zlib-retraite/

Page 24

lien 63 : ... http://www.developpez.net/forums/d1571471/club-professionnels-informatique/actualites/logiciel-developpe-startup-romande-traquer-ecrivains-fantomes/

Page 25

lien 64 : ... https://youtu.be/rVlhMGQgDkY
lien 65 : ... http://www.developpez.net/forums/d1570819/general-developpement/algorithme-mathematiques/intelligence-artificielle/boston-dynamics-apporte-mise-jour-majeure-robot-atlas/

Page 27

lien 66 : ... http://marcautran.developpez.com/applis/panier/panier.html

Page 29

lien 67 : ... http://marcautran.developpez.com/tutoriels/javascript/panier/

Page 30

lien 68 : ... http://www.toptal.com/designers/ipad
lien 69 : ... http://www.toptal.com/responsive-web
lien 70 : ... http://www.toptal.com/designers/responsive
lien 71 : ... https://developers.google.com/webmasters/mobile-sites/
lien 72 : ... https://gigaom.com/2014/11/19/mobile-friendly-sites-might-get-a-boost-in-google-search-results/
lien 73 : ... http://www.computerworld.com/article/2486642/internet/mobile-browser-usage-share-hits-20--for-the-first-time.html

Page 31

lien 74 : ... http://adaptive-images.com/
lien 75 : ... http://www.sencha.com/learn/how-to-use-src-sencha-io/

Page 32

lien 76 : ... https://html.spec.whatwg.org/multipage/embedded-content.html#attr-img-srcset
lien 77 : ... https://github.com/scottjehl/picturefill
lien 78 : ... http://scottjehl.github.io/picturefill/#support
lien 79 : ... http://codepen.io/sheadawson/pen/fvFLc
lien 80 : ... https://github.com/BBC-News/Imager.js
lien 81 : ... http://bbc.co.uk/

Page 33

lien 82 : ... http://jordanm.co.uk/2012/05/13/source-shuffling-responsive-images-based-on.html
lien 83 : ... https://github.com/teleject/hisrc
lien 84 : ... http://www.mobify.com/mobifyjs/
lien 85 : ... http://www.toptal.com/responsive-web/responsive-design-is-not-enough-we-need-responsive-performance
lien 86 : ... http://web.developpez.com/tutoriels/images-responsive-web/

Page 34

lien 87 : ... http://www.developpez.net/forums/u37622/argyronet/

Page 51

lien 88 : ... http://argyronet.developpez.com/office/access/prefilldataformswiz/

Page 52

lien 89 : ... https://youtu.be/zO4-1D9LdDw
lien 90 : ... http://virtualisation.developpez.com/cours/#Docker
lien 91 : ... http://www.developpez.net/forums/d1563596/systemes/virtualisation/docker/tutoriel-video-quoi-neuf-docker-explique-video-qu-docker/

Page 55

lien 92 : ... http://www.clicketcloud.com/

Page 58

lien 93 : ... https://test-dvp.clicketcloud.com/

Page 61

lien 94 : ... http://clicketcloud.developpez.com/tutoriels/decouvrir-apprendre-cloud-computing/