Java - Backend avec Vert.x : Intro

Avant-propos

Juste avant de commencer cette entrée en matière de Vert.x, quelques précisions sur des opérateurs Java que je vais utiliser :

Les fonctions lambda | x -> ...
Avant java 8, itérer sur des listes était plutôt fastidieux, java a donc fait un pas vers la programmation fonctionnelle (sujet fort intéressant) en intégrant les lambdas expressions. Déjà présente dans plusieurs langages donc vous risque de les retrouvez souvent (Ruby, C++, JS, Python ...). L'intérêt ce qu'on manipule des copies des listes, ainsi pour chaque opération que l'on va faire on va cloner la liste, effectuer les opérations et retourner une nouvelle liste sans toucher à l'ancienne.

```
List<Integer> num = Arrays.asList(12, 56, 31, 24, 15, 26);
num.map(x -> x*2); // [24, 112, 62, 48, 30, 52]
num.filter(x -> x > 30) // [56, 31]
num.sum() // 164 (oui je me fais chier a calculer ça...)
```

Opérateur ternaire | ? ... : ...
Cet opérateur est vraiment utile lorsqu'on veut tester une condition et appliquer une opération en conséquence de la réponse, super-simple et permet d'être plus concis.

```
boolean majeur = age > 18 ? true : false;
```

L'opérateur de référence | ::
Le dernier et sans doute l'un des plus puissants est l'opérateur 'Méthod Reference' ou 'double colon operator', il permet de faire voyager des arguments de notre méthode vers une autre méthode, ainsi il prend TOUT les arguments en paramètre de notre fonction et les insère dans l'autre. Titre !

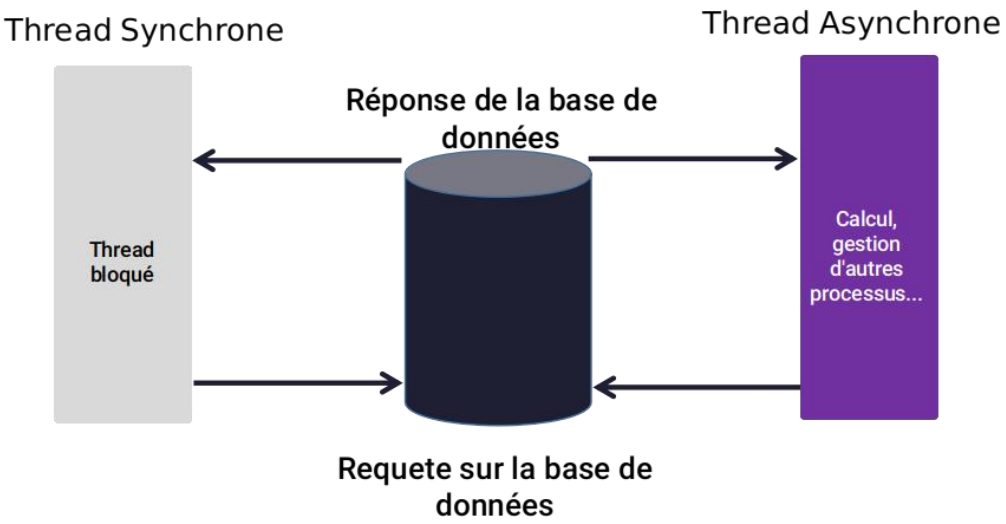
```java
public void print(String TextToPrint){
    /* System.out.println(TextToPrint); */
    System.out::println;
}

public void printNumbers(List<Integer> num){
    /* numbers.map(n -> System.out.println(n)) */
    num.map(System.out::println);
}
```

Un peu complexe à comprendre, je vous mets en commentaire le code dit 'classique' et en dessous transformé avec le ce nouvel opérateur.

Vert.x

En gros, c'est cool.
Non en vrai ce qu'il faut retenir c'est que notre code va tourner au sein d'une Verticle, qui va s'occuper de beaucoup de choses pour nous... Déjà tout notre code devient asynchrone (à l'exception de quelques méthodes, qu'on ne va pas utiliser d'ailleurs), et là ça devient intéressant, surtout quand on fait du web :



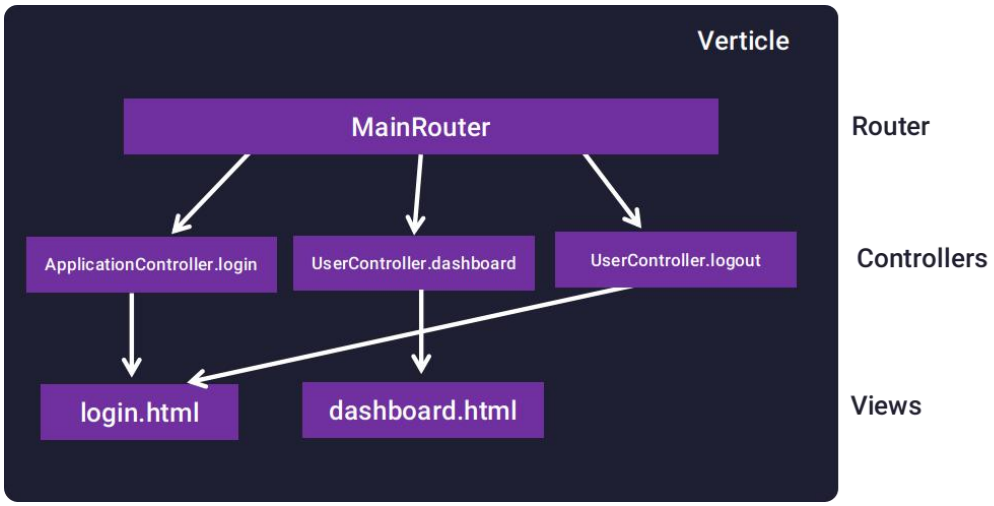
Parce que ce genre de requêtes tu connais ...

Structure et parcours

Bon la accrochez vous bien parce que c'est du lourd (un peu comme toi

HUGO SAGE).

D'un point de vue MVC (Model, View, Controller), on doit avoir un routeur qui dispatche les routes des utilisateurs vers les actions correspondantes, chargé de rendre une vue. Bon en vrai pour nous c'est plutôt des méthodes pas des actions (rappel pour ceux au fond qui n'écoutent rien les méthodes c'est les fonctions d'une classe, et les méthodes dites statiques n'ont pas besoin que l'on instancie un objet de la classe pour fonctionner).
Du coup on a un truc comme ça :



Voilà vu que je m'endors sur mon clavier je vous donne donc le parcours type : un utilisateur va se connecter sur notre serveur sur '/login' et on va lui rendre la page login.

```
router.get('/login').handler(ApplicationController::index)
```

MainRouter.java

```
public static void index(RoutingContext context) {
        render(context, "index.html");
}
```

ApplicationController.java

Spoiler

Quand j’ai le temps je fait une autre doc sur rendre une vue / json, récupérer des paramètres et enfin faire des requêtes avec la base de données.
Tchou je retourne niquer des mères sur les pistes ! Gros bisous

quand tes en route pour niquer des mères