Introduction à Spring boot

1 Très légère introduction aux API RESTfull

Cette section est à ajouter au TP Spring-MVC précédent.

L'idée est simple :

- Proposer une API WEB pour récupérer et agir sur les données d'un serveur.
- Utiliser les méthodes HTTP (GET, POST, DELETE, etc) pour coder les actions sur les données.
- Utiliser des langages normalisés pour la description des données (XML et surtout Json).
- Offrir ainsi un service complet accessible aux clients (des applications WEB, des téléphones portables, des dispositifs nomades, etc.)

1.1 Mise en place d'un controleur REST

Commencez par ajouter les dépendances pour Jackson (outils pour transformer une instance Java en Json et vice-versa) :

```
...
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.8</version>
</dependency>
...
```

Créez ensuite une contrôleur REST (une calculatrice à pile) :

```java
package springapp.web;

import java.util.Stack;

import javax.annotation.PostConstruct;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/calculator")
public class RestCalculator {

    protected final Log logger = LogFactory.getLog(getClass());
    private Stack<Integer> numbers = new Stack<>();

    @PostConstruct
    public void init() {
        numbers.push(100);
        numbers.push(200);
        numbers.push(300);
    }

    @GetMapping("/show")
    public Stack<Integer> show() {
        return numbers;
    }

    @GetMapping("/add")
    @ResponseStatus(HttpStatus.OK)
    public void add() {
        Integer val1 = numbers.pop();
        Integer val2 = numbers.pop();
        numbers.push(val1 + val2);
    }

    @PostMapping(value = "/put", consumes = "application/json")
    @ResponseStatus(HttpStatus.CREATED)
    public String put(@RequestBody() Integer id) {
        numbers.push(id);
        logger.info(String.format("put␣%d", id));
        return "Ok";
    }

}
```

Testez l'API Rest avec des requêtes directes :

```
http://localhost:8080/votre_application/actions/calculator/show
http://localhost:8080/votre_application/actions/calculator/add
http://localhost:8080/votre_application/actions/calculator/show
```

La première donne les trois éléments de la pile. La second remplace les deux éléments de la tête de pile par leur addition. etc.

1.2 Une petite application REST

Nous allons maintenant créer un code javaScript coté client qui va interagir avec cette API REST. Commencez par le fichier javaScript suivant :

Fichier WebContent/functions.js

```
function showStack() {
    var base = ($('<a␣href=".">')[0].href);
    $.ajax({
        type : 'GET',
        url : (base + "actions/calculator/show"),
        data : '200',
        timeout : 3000,
        success : function(data) {
            $('#numbers').hide();
            $('#numbers').html("Stack:␣");
            jQuery.each(data, function(i, val) {
                $("#numbers").append("␣-␣");
                $("#numbers").append(document.createTextNode(val));
            });
            $('#numbers').show();
        }
    });
}

function show() {
    showStack();
    $('#message').html("");
}

function add() {
    var base = ($('<a␣href=".">')[0].href);
    $.ajax({
        type : 'GET',
        url : (base + "actions/calculator/add"),
        timeout : 3000,
        error : function() {
            $('#message').html('Addition␣impossible');
            showStack();
        },
        success : function(data) {
            $('#message').html('Addition␣réalisée');
            showStack();
        }
    });
}

function put() {
    var base = ($('<a␣href=".">')[0].href);
    var value = ($('#input').val());
    $.ajax({
        type : 'POST',
        url : (base + "actions/calculator/put"),
        data : value,
        timeout : 3000,
        dataType : "json",
        contentType : "application/json",
        success : function(data) {
            showStack();
            $('#input').html("");
        },
        error : function() {
            showStack();
            $('#input').html("");
        }
    });
}
```

Ces fonctions JavaScript vont utiliser la méthode `ajax` de `JQuery` pour envoyer des requêtes asynchrones vers l'API REST.

Nous pouvons maintenant préparer une page JSP qui va produire une page HTML à destination d'un navigateur. La page chargée va utiliser `JQuery` et proposer une interface minimale pour lister les éléments de la pile, ajouter un nombre et calculer une addition.

Fichier `WebContent/rest-app.js`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Simple stack calculator</title>
<script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.js"></script>
<script src="functions.js"></script>
</head>
<body>
    <h1>Simple stack calculator (rest application)</h1>
    <div>
        <button onclick="show();">Show</button>
        <input id="input" size="10" />
        <button onclick="put();">put</button>
        <span> | </span>
        <button onclick="add();">+</button>
        <span> </span> <span style="color: blue;" id="message"></span>
    </div>
    <p id="numbers"></p>
</body>
</html>
```

Travail à faire : prévoir l'opération de soustraction (fonction JavaScript, bouton html et requête `/calculator/sub` sur l'API REST.

2 Introduction à Spring boot

Le framework **Spring boot** a pour objectif d'accélerer et de simplifier la mise en place d'une application basée sur Spring.

Cet objectif est atteint par l'intégration automatique de nombreux composants et un système d'auto-configuration qui couvre la majorité des cas.

2.1 Mise en place d'un projet

ls *.zip Préparez un environement pour tester une application WEB basée sur Spring-boot :

- Téléchargez le projet Maven [1] déjà préparé.
- Décompressez cette archive.
- Importez, dans Eclipse, un projet Maven et choisissez le répertoire précédent.
- Exécutez ce projet (**Run as .. Java application** classe `mybootapp.Starter`).
- Testez le bon fonctionnement à l'adresse `http://localhost:8081`

1. ens-SpringBoot.zip

Explication : Spring boot lance une instance embarquée de Tomcat pour déployer votre application WEB. Elle est donc directement accessible.

À faire :

- Stopper l'exécution.
- Faites un **Run as ... Maven build... goal : package** dans Eclipse.
- Faites la même chose en ligne de commande :

Build du projet

```
cd repertoire_de_votre_projet
mvn package
```

- Ces deux opérations ont généré une fichier `.war` dans le répertoire `target`.
- Lancez votre application en ligne de commande :

Exécution du projet

```
cd repertoire_de_votre_projet
java -jar target/nom_de_votre_fichier.war
```

- **Moralité** : il est très simple de compiler, déplacer et exécuter une application WEB avec ce type d'outil.

2.2 Explorer ce projet

Les projets **Maven** ont une structure particulière :

- `pom.xml` : configuration de Maven
- `src/main/java` : le code source Java
- `src/main/resources` : les ressources (fichiers XML, images, propriétés, etc.)
- `src/main/resources/static` : les ressources statiques de votre application WEB (CSS, images, autres)
- `src/main/resources/application.properties` : le fichier de paramétrage de Spring boot
- `src/test/java` : le code source Java de test
- `src/test/resources` : les ressources pour le test
- `src/main/webapp` : les fichiers de l'application WEB (pages JSP par exemple)
- `src/main/webapp/WEB-INF` : le répertoire `WEB-INF`
- `src/main/webapp/WEB-INF/jsp` : les pages JSP cachées

Travail à faire :

- Faites fonctionner l'application WEB,
- Explorer le contrôleur `MyControler` et la page `src/main/webapp/WEB-INF/jsp/index.jsp`. Changez le message dans le fichier `src/main/resources/application.properties` et vérifiez sa prise en compte.
- Explorer le contrôleur `CourseControler` et la page `src/main/webapp/WEB-INF/jsp/course.jsp`.

- Explorer la classe DAO : `CourseRepository`. Vous remarquerez que c'est une interface mais que nous n'avons pas d'implantation. C'est Spring qui se charge de construire l'implantation directement à partir des noms de méthodes déclarées dans l'interface. Vous remarquerez le `Like` dans le nom de la méthode utilisée par le contrôleur. C'est la base de la technologie **Spring-data** (plus d'information [2]). Vous pouvez lire la documentation sur les méthodes de requêtes [3].
- Ajoutez à cette application une fonction de suppression des cours dont le nom commence par UE :
  - ▷ Avec cette documentation [4], ajoutez une méthode `deleteLikeFirstName(String pattern)` à la classe `CourseRepository` (utilisez `@Query`).
  - ▷ Ajoutez au contrôleur le traitement de la requete `/course/delete`
  - ▷ Ajoutez à la page `course.jsp` un lien vers la nouvelle action.

2. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/
3. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.query-creation
4. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.modifying-queries