MODULE Spring
Les ressources

HUMAN booster
VOTRE SOLUTION COMPETENCE

Spring - Les ressources

Plan du module Spring – Les ressources

- Introduction
- Types de ressources
- Accès aux ressources
- Localisation des définitions de bean
- Valeurs de ressource

Spring - Les ressources

Introduction

Les ressources - Introduction

Il est intéressant de configurer le fonctionnement d'une application, de manière à l'adapter à un nouveau contexte d'exécution sans changer le code source.

Exemple dans le domaine des objects connectés (IoT) :

- Changement de la fréquence radio utilisée pour la communication entre un device et une gateway ou un datacenter
- Fréquence de lecture des capteurs sur un device
- Adresse IP du datacenter auquel envoyer les données
- ...

Les ressources - Introduction

Une **ressource** représente un élément d'information de ce type :

- Une adresse IP
- Une fréquence radio
- Une fréquence de rafraichissement
- Etc.

Ce genre d'information est généralement représenté par une valeur **statique** (toujours la même) et **simple** (une chaine de caractères, une valeur entière, etc.).

Les ressources - Introduction

Par simplicité, ce genre d'information n'est habituellement pas stocké dans une base de données, mais plutôt :

- dans un fichier, local ou distant
- ou sur un site web.

Java fournit la classe `java.util.Properties` pour supporter ce genre d'information sous forme de couples clé/valeur dans un fichier texte.

Spring généralise ce concept.

Les ressources - Introduction

Spring permet d’accéder à ces informations au moyen de **ressources Spring**.

Une ressource Spring implémente l’interface `Resource`.

Types de ressources

L'interface `org.springframework.core.io.Resource` permet en particulier d'accéder aux informations suivantes :

- Existence de la ressource physique correspondante
- État de la ressource (ouverte ou non, donc lisible ou non)
- Objet `URL` pour lire la ressource
- Objet `File` pour lire la ressource
- Nom du fichier correspondant à la ressource
- Description de la ressource.

Types de ressources

Cette interface étend l'interface `InputStreamSource`, ce qui veut dire qu'on peut lire les données contenues par la ressource comme un stream (fichier, page web, …).

Types de ressources

Spring fournit les implémentations suivantes de cette interface :

- `UrlResource` → pour accéder à tout objet accessible **via une URL** (fichier, cible HTTP, cible FTP, ...)
- `ClassPathResource` → pour accéder à tout fichier local “plain” **dans le class path** de l’application, mais **pas** dans une archive (JAR ou ZIP par exemple)
- `FileSystemResource` → pour accéder à tout fichier local, quelle que soit sa localisation sur le filesystem

Types de ressources

Autres implémentations de cette interface :

- `ServletContextResource` → pour accéder à une ressource située sous la racine de l'application web
- `InputStreamResource` → pour accéder à une ressource particulière, non couverte par une autre implémentation de l'interface
- `ByteArrayResource` → pour accéder à une ressource représentée par un tableau d'octets (données binaires typiquement).

Accès aux ressources

Les ressources peuvent être chargées au moyen d'un chargeur de ressources, un objet qui respecte l'interface `ResourceLoader`.

Cette interface fournit un seul service :

```
Resource getResource(String location);
```

Tous les contextes applicatifs (`ApplicationContext`) implémentent cette interface. Il est donc trivial d'obtenir une ressource à partir d'un contexte applicatif.

Accès aux ressources

La chaine de caractères `Location` peut contenr un préfixe qui précise le type de la ressource :

- `classpath:`
- `file://`
- `http:// ou https://`

Dans ce cas, la ressource retournée aura le type précisé.

Accès aux ressources

Préfixe utilisé	**==>**	**Type de ressource**
`classpath:`	==>	`ClassPathResource`
`file://`	==>	`FileSystemResource`
`http://`	==>	`URLResource`

Accès aux ressources

Si le préfixe n'est **pas** précisé, alors c'est le **contexte applicatif utilisé** qui fixera comment la ressource sera recherchée et obtenue.

Type de contexte applicatif	**==>**	**Type de ressource**
`ClasspathXmlApplicationContext`	`==>`	`ClassPathResource`
`FileSystemXmlApplicationContext`	`==>`	`FileSystemResource`
`WebApplicationContext`	`==>`	`ServletContextResource`

Accès aux ressources

Comment accéder aux ressources ?

- Si l'accès à une ressource est **dynamique**, c'est-à-dire dépend du contexte, alors il est classique de charger la ressource manuellement (par utilisation d'un `ResourceLoader`)
  - Exemple : si la ressource à utiliser dépend du rôle que tient l'utilisateur, alors on parle d'une ressource dynamique
- Si par contre la ressource est statique (toujours la même), alors on peut éviter d'utiliser un `ResourceLoader`, associer au bean la ressource qu'il utilise, et laisser l'IoC container **injecter** la ressource.

Accès aux ressources

Exemple d'association d'une ressource à une propriété d'un bean au travers d'un fichier XML de métadonnées.

```
<bean id="myBean" class="...">
    <property name="template"
       value="chemin/vers/la/ressource.txt"/>
</bean>
```

NB. Puisque ici la localisation de la ressource ne contient pas de préfixe, le type de la ressource créée sera indiqué par le **type de contexte applicatif** dans lequel ce bean sera défini.

Accès aux ressources

Rien n'empêche bien sûr d'indiquer un préfixe.

```
<bean id="myBean" class="...">

    <property name="template"

        value="classpath:chemin/vers/la/
        ressource.txt"/>

</bean>
```

Dans ce cas de figure, le type de la ressource créée sera `ClassPathResource`, **quel que soit** le type de contexte applicatif dans lequel ce bean sera défini.

Accès aux ressources

Une ressource peut être injectée au moyen de l'annotation `@Value`.

```
@Value("classpath:conf/maRessource.txt")

Resource fichierRessource;
```

On peut bien sûr utiliser les autres préfixes supportés, comme `file:` et `url:`.

Localisation des définitions de bean

Rappel : lorsqu'on crée un `ClassPathXmlApplicationContext`, on peut préciser le chemin vers les fichiers XML de métadonnées :

```
ApplicationContext context = new
ClassPathXmlApplicationContext("fichier1.xml",
"fichier2.xml", ...);
```

Ces fichiers de métadonnées sont en fait des **ressources** comme des autres.

Plus généralement, quand on crée un `ApplicationContext`, on peut préciser en argument où se trouve les ressources qu'il utilisera.

Localisation des définitions de bean

La localisation de ces métadonnées dépendra :

- Du type de l'`ApplicationContext` utilisé
- Et de l'utilisation éventuelle d'un préfixe dans la localisation des ressources.

Pour un `ClassPathXmlApplicationContext`, les fichiers seront recherchés à partir du classpath.

Pour un `FileSystemXmlApplicationContext`, les fichiers seront recherchés à partir du répertoire de travail.

Valeurs de ressource

Accéder aux **valeurs** d'une ressource (les informations contenues dans la resource) se fait au moyen :

- de l'objet `File`
- ou de l'objet `URL`

associé à la ressource.

Valeurs de ressource

Il est aussi possible de faire lire les valeurs automatiquement dans le cas ou la ressource consiste en un fichier de propriétés, tout en **évitant de placer le nom du fichier** de propriétés dans le code source (comme ce serait le cas si on utilisait une annotation `@PropertySource`).

Pour cela :

- On définit dans un fichier les propriétés qui nous intéressent
- Et on fait pointer une ressource vers ce fichier, ressource qu'on importe dans le bean concerné au tracvers d'une annotation `@ImportResource`.

Valeurs de ressource

Exemple.

```
@Configuration
@ImportResource(“classpath:/conf/proprietes.xml”)
public class ConfigProjet {
    @Value(“${moteur.fabriquant}”) private String fabriquant;
    @Bean
    public Moteur moteur() {
        return new Moteur(fabriquant);
    }
}
```

Valeurs de ressource

Contenu du fichier `proprietes.xml`.

```
<beans ...>

    <context:property-placeholder
    location="classpath:/com/acme/proprietes_moteur.txt"/>

</beans>
```

Valeurs de ressource

Contenu du fichier `proprietes_moteur.txt`.

```
moteur.fabriquant=Toyota
moteur.cylindree=1.8l
moteur.puissance=240 BHP
...
```