



Spring Core – xml configuration

Bean Scope

Définition scope (ou portée)

- C'est un qualificatif de la manière dont Spring va instancier, "exposer" (on parle ici de visibilité), puis détruire chaque Bean.
- Pour chaque bean , on peut spécifier une portée.
- Chaque bean possède une valeur de portée dite "par défaut" avec spring.

Scope ... en d'autres termes.

La notion de scope renvoie à celle du cycle de vie d'un bean , elle spécifiera =

- Combien de temps un bean vivra
- Combien d'instances seront créées
- Comment le bean sera partagé dans l'environnement Spring



Les valeurs de @Scope

- singleton (default)
- prototype
- request
- session
- global-session

Singleton

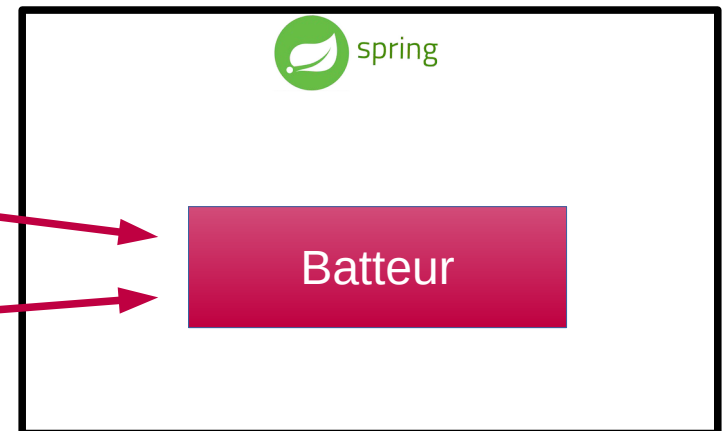
Quand le scope d'un bean est configuré explicitement (ou par défaut) comme un singleton, Spring ne va créer qu'une seule instance de ce bean.

Cette instance sera maintenue en cache

Tous les appels à ce bean accèderont à cette même instance.

File: demoApp.java

```
Musicien musicienA =  
    context.getBean("unMusicien", Musicien.class);  
  
Musicien musicienB =  
    context.getBean("unMusicien", Musicien.class);
```



File: applicationContext.xml

```
<bean id="unMusicien" class="com.springdemo.Batteur" > ... </bean>
```

Spécification explicite

Scope explicite
à singleton

```
<beans ...>  
<bean id="unMusicien" class="com.springdemo.Batteur" scope="singleton"> . . . </bean>  
</beans>
```

singleton (default) : créer une seule instance partagée

prototype : créer un nouvel objet pour chaque requête faite au container

request : exposition à une requête web HTTP (cf web app)

session : exposition à une session web HTTP (cf web app)

global-session : exposition à une session web HTTP globale (cf web app)

Scope = prototype

Un nouvel objet est crée à chaque requête

Ex :

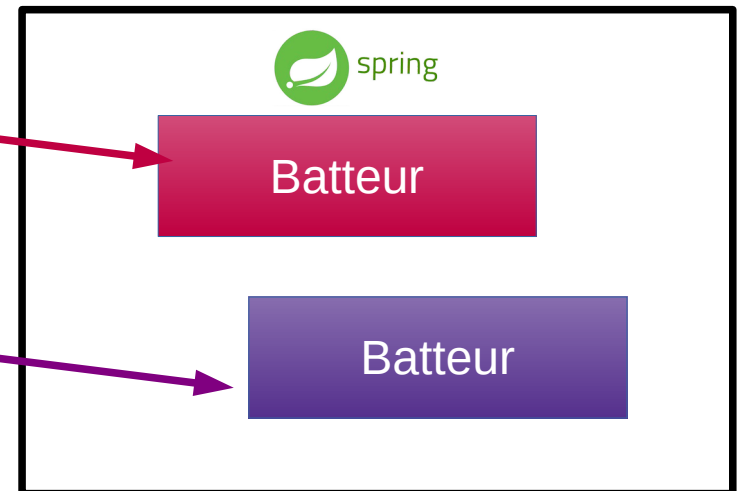
```
<beans . . . >

    <bean id="unMusicien"
          class="com.springdemo.Batteur"
          scope="prototype">
    </bean>

</beans>
```

```
Musicien musicienA =
    context.getBean("unMusicien", Musicien.class);
```

```
Musicien musicienB =
    context.getBean("unMusicien", Musicien.class);
```





Une démo pour illustrer SVP !

beanScope-applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

  <!-- Définir les beans ici -->
  <!-- Définir les Dépendances -->
  <bean id="unPrepareService"
    class="com.springdemo.ZenPreparationService" >
  </bean>

  <bean id="unMusicien" class="com.springdemo.Batteur" >
    <!-- Définir l' Injection par constructeur -->
    <constructor-arg ref="unPrepareService"/>
  </bean>

</beans>
```

BeanScopeDemoApp

```
package com.springdemo;

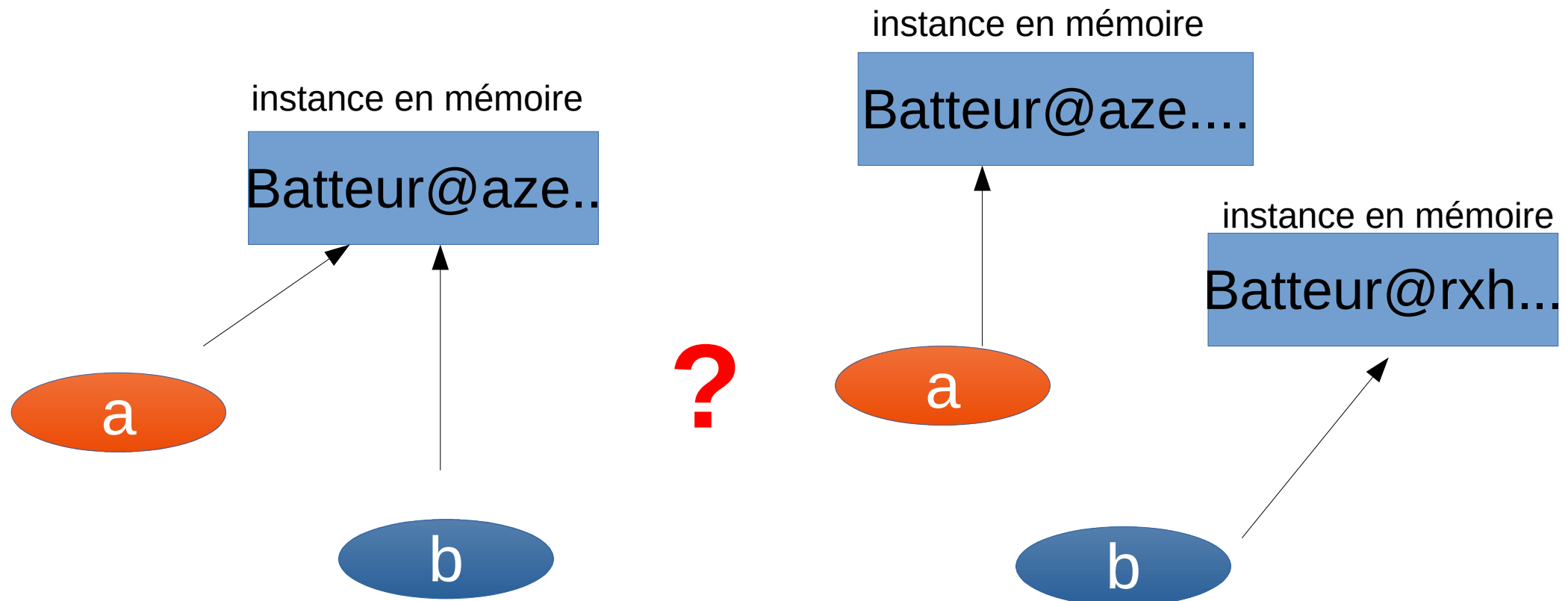
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class BeanScopeDemoApp {

    public static void main(String[] args) {
        // charger le fichier de configuration
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("beanScope-applicationContext.xml");

        // récupère le(s) bean(s) depuis le spring container
        Musicien a= context.getBean("unMusicien", Musicien.class);
        Musicien b = context.getBean("unMusicien", Musicien.class);
    }
}
```

Un problème en java (rappel): a et b référencent -elles le même objet ?



=> " == "

```
public class BeanScopeDemoApp {  
  
    public static void main(String[] args) {  
        //charger le fichier de configuration xml  
        ClassPathXmlApplicationContext context =  
            new ClassPathXmlApplicationContext("beanScope-applicationContext.xml");  
  
        // accéder au bean géré par le spring container  
        Musicien a= context.getBean("unMusicien", Musicien.class);  
        Musicien b = context.getBean("unMusicien", Musicien.class);  
  
        // vérifier si l'adresse mémoire est la même, si il s'agit d'un seul et même objet  
        boolean result = (a == b);  
  
        // afficher result en console  
        System.out.println("\nPointing to the same object: " + result);  
        System.out.println("\nMemory location for theCoach: " + a);  
        System.out.println("\nMemory location for alphaCoach: " + b + "\n");  
  
        // fermer le context  
        context.close();  
    }  
}
```

Run As > Java Application =>

Pointing to the same object: true

Memory location for theCoach: com.springdemo.Batteur@69fb6037

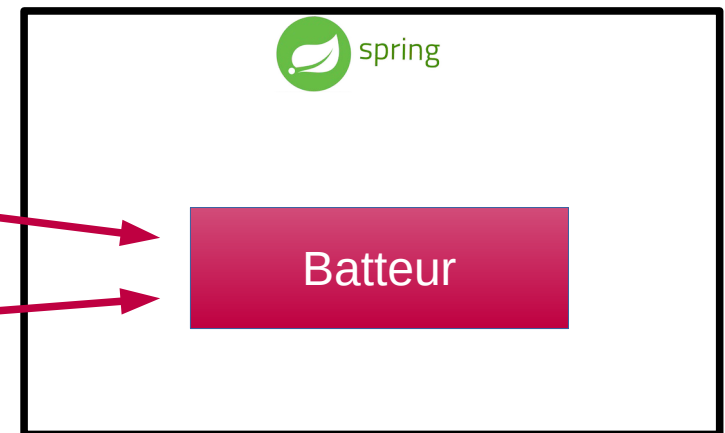
Memory location for alphaCoach: com.springdemo.Batteur@69fb6037

Conclusion

On a illustré le singleton , on a même illustré le singleton par défaut, car on a déclaré le bean "unMusicien" sans préciser l'attribut scope.

File: demoApp.java

```
Musicien musicienA =  
    context.getBean("unMusicien", Musicien.class);  
  
Musicien musicienB =  
    context.getBean("unMusicien", Musicien.class);
```



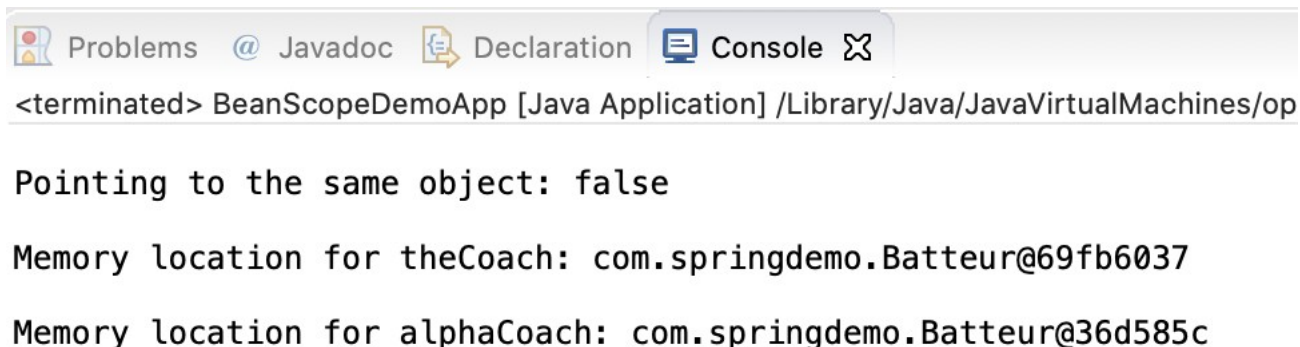
démo scope=prototype

- Modifions le scope = prototype pour que spring fournisse un nouvel objet à chaque requête

```
<beans . . .  
    <bean id="unMusicien" class="com.springdemo.Batteur" scope="prototype">  
        <!-- Définir l' Injection par constructeur -->  
        <constructor-arg ref="unPrepareService"/>  
    </bean>  
</beans>
```

- On ne change rien d'autre par ailleurs et on execute l'application

Run as > Java Application =>



```
<terminated> BeanScopeDemoApp [Java Application] /Library/Java/JavaVirtualMachines/op  
  
Pointing to the same object: false  
  
Memory location for theCoach: com.springdemo.Batteur@69fb6037  
  
Memory location for alphaCoach: com.springdemo.Batteur@36d585c
```