Introduction au framework Spring

Audran YOULOU ZHAZHA, Jcertif University 2012
<audran.zhazha@gmail.com>

Plan

- Problématiques des développements Java EE
- Les réponses de Spring
- L’écosystème Spring
- Programmation par interface
- Injection de dépendances et IoC
- Conterneur léger - Spring IoC
- Spring AOP
- Questions

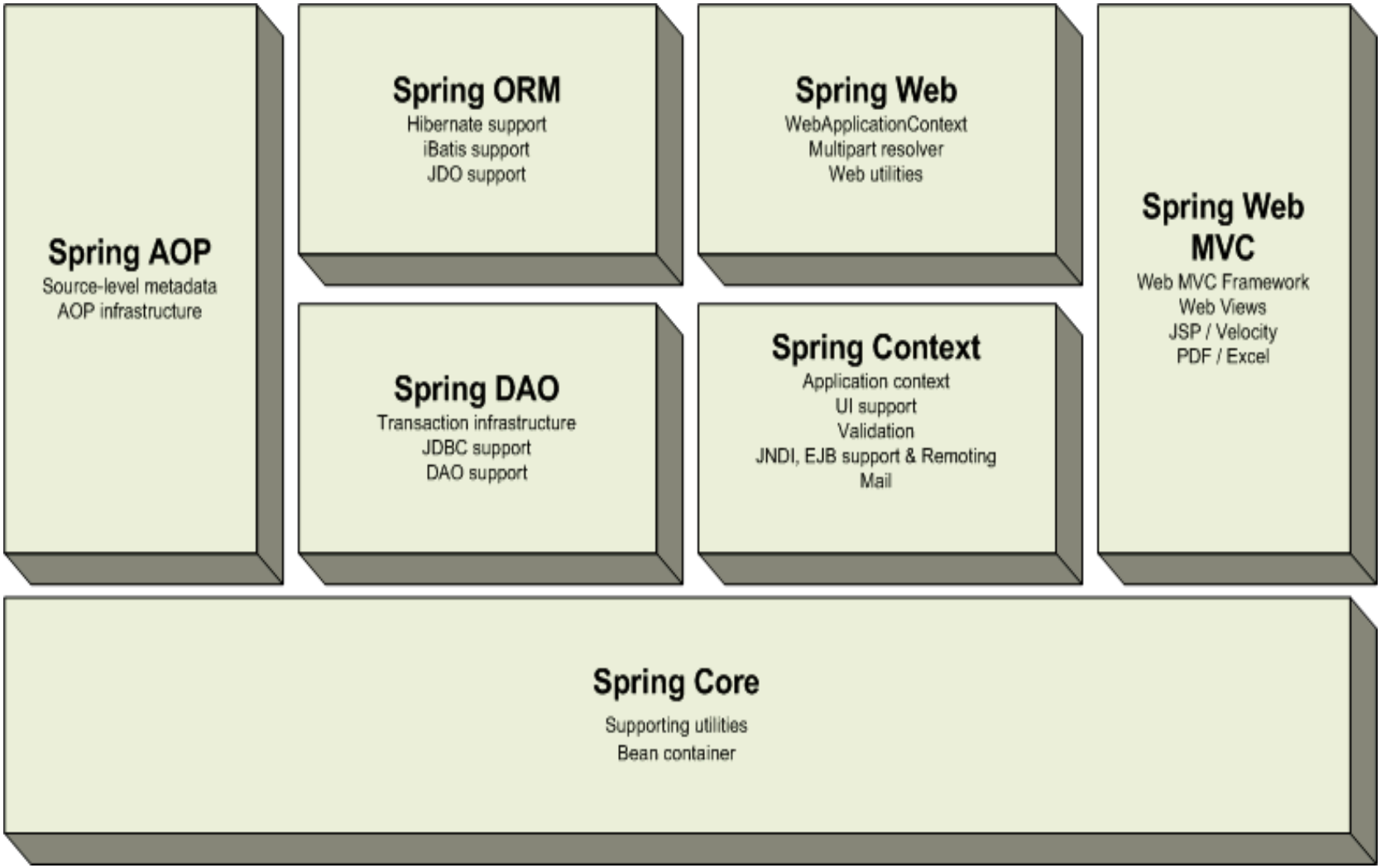
Problématiques des développements Java EE

- **Mauvaise séparation des préoccupations**: des problématiques technique et métier sont mal isolées
- **Complexité**: pléthore de spécifications, serveur d'applications monolithique implémentant la totalité de Java EE
- **Mauvaise interopérabilité**: les technologies ne sont pas toujours interopérables et portables
- **Mauvaise testabilité**: les applications dépendant fortement de l'infrastructure d'exécution, elles sont plus difficilement testables.

Les réponses de Spring

- **Conteneur léger**: fournit un support pour gérer une application via un ensemble de composants. Gère le cycle de vie des composants, mais aussi leurs interdépendances. Permet d'avoir des applications indépendantes du serveur d'applications
- **Programmation orientée aspect (POA)**: le support de la POA de Spring n'impose aucune contrainte et permet d'ajouter du comportement à n'importe quel type d'objet, il améliore la séparation des préoccupations dans une application
- **Intégration de frameworks tiers**: intégration des frameworks et de standards Java EE

L’écosystème Spring



Programmation par interface

En java, la communication entre les couches logicielles s'effectue grâce à la **notion d'interface**

```
public class UserManagerImpl{
    UserDAOJpa dao;
    public UserManagerImpl(){}

}
```

- Le service métier ci – dessus, dépend fortement de l'implémentation du DAO, on a **couplage fort**

```
public class UserManagerImpl{
        UserDAO dao;

        public UserManagerImpl(){
                //TODO
        }
}

public interface UserDAO {
        public void createUser();
}

public class UserDAOJpa implements UserDAO {
        public void createUser(){
                //TODO
        }
}
```

- L'interface permet de découpler le service de l'implémentation du DAO, on a **couplage faible**

Injection des dépendances et IoC

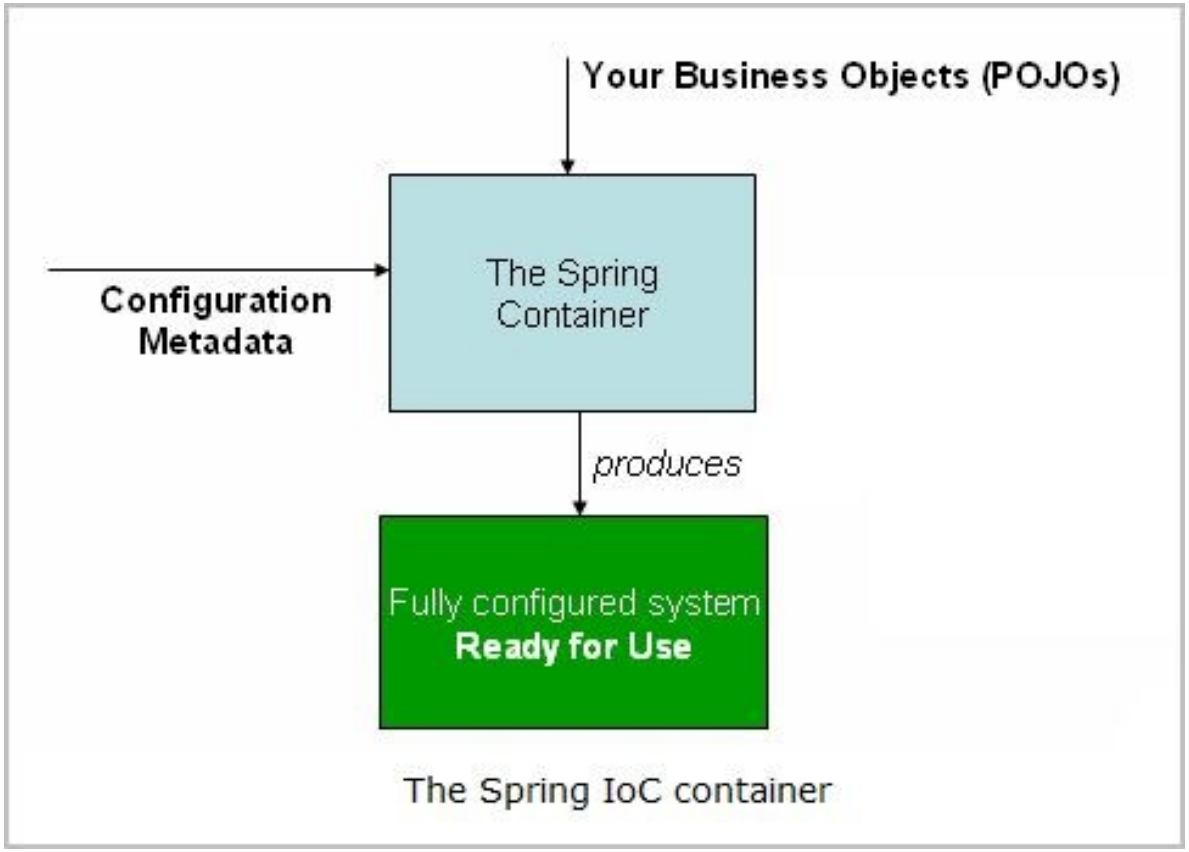
- **Injection des dépendances**
  - L'I.D s'appuie sur un objet assembleur (le conteneur léger)
  - Le conteneur léger de Spring injecte les dépendances de manière appropriée
  - Avec l'I.D, les composants sont plus indépendants de l'environnement d'exécution
- **Inversion de contrôle**
  - Le contrôle fait référence au **flot d'exécution** d'une application
  - L'IOC vise à résoudre les problématiques **d'instanciation** et de **dépendances** entre les composants d'une application

Conteneur léger – Spring IoC

- Le conteneur léger de Spring: **fabrique d'objets,** et a un contrôle total sur les composants d'une application:
  - **Cycle de vie**: contrôle les configurations et toute séquence d'initialisation
  - **Portée**: gère la portée des composants, elle peut être globale à l'application ou spécifique à une requête en cours
  - **Décoration**: le conteneur peut décorer un composant en lui ajoutant un comportement
  - **Infrastructure**: le conteneur peut fournir des services comme un pool de connexions ou même un gestionnaire de transactions

Conteneur léger – Spring IoC

- L'interface **BeanFactory** est le socle de Spring IoC



Conteneur léger – Spring IoC

- Instanciation du conteneur léger de Spring

```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"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="user" class="fr.jcertif.domain.model.User">
    <property name="firstName" value="Azhy"/>
    <property name="lastName" value="Durand"/>
  </bean>

</beans>
```

spring-context.xml

```
public class IntroSpring{
        public static void main(String[] args){
                //Chargement du fichier à partir du système de fichiers
                //context : objet Spring contenant les objets décrits dans le fichier de
                //config. XML
                ApplicationContext context = new FileSystemXMLApplicationContext("spring-
                        context.xml");

                //Chargement du fichier de config. XML à partir du classpath
                //ApplicationContext context = new ClassPathXMLApplicationContext("spring-
                //        context.xml");

                User user = (User) context .getBean("user");

                System.out.println("Utilisateur : "+user .getFirstName()+"
                "+user.getLastName());
        }
}
```

IntroSpring.java

Conteneur léger – Spring IoC

- Utilisation d'un bean Spring
  - Les schémas XML

Nom	Description
beans	Définition des beans et de leurs dépendances
aop	Gestion de la programmation orientée aspect
context	Activation des annotations et positionnement de post-processeurs
util	Déclaration de constantes et de structures de données
jee	Fonctions pour s'interfacer avec JNDI et les EJB
jms	Configuration de beans JMS
lang	Déclaration de beans définis avec des langages de script
p	Définition des propriétés de beans
tx	Déclaration des transactions sur les beans

Conteneur léger – Spring IoC

- Utilisation d'un bean Spring
  - Nommage des beans

```
//Utilisation de l'identifiant id
<bean id="user" class="fr.jcertif.domain.model.User"/>

Ou bien,

//Utilisation de l'identifiant id ou des alias définis par l'attribut name
<bean id="user" class="fr.jcertif.domain.model.User" name="client, visiteur"/>
```

  - Méthodes d'injection

```
//Injection par modificateur
<bean id="user" class="fr.jcertif.domain.model.User">
        <property name="firstName" value="Julien" />
        <property name="lastName" value="Ancele" />
</bean>

Ou bien,

//Injection par constructeur
<bean id="user" class="fr.jcertif.domain.model.User">
        <constructor-arg value="Julien" index="0"/>
        <constructor-arg value="Ancele" index="1"/>
</bean>
```

Conteneur léger – Spring IoC

- Utilisation d'un bean Spring
  - Injection explicite des collaborateurs

```
<bean id="userDAO" class="fr.jcertif.domain.dao.jpa.UserDAOJpa" />

<bean id="userService" class="fr.jcertif.service.impl.UserServiceImpl">
        <property name="dao" ref="userDAO"/>
</bean>

Ou bien,

<bean id="userService" class="fr.jcertif.service.impl.UserServiceImpl">
        <property name="dao">
                <bean class="fr.jcertif.domain.dao.jpa.UserDAOJpa"/>
        </property>
</bean>
```

  - Injection automatique des collaborateurs

```
public class UserServiceImpl implements UserService{
        //Par défaut, Autowired impose une dépendance obligatoire
        @Autowired
        private UserDAO dao;
}
```

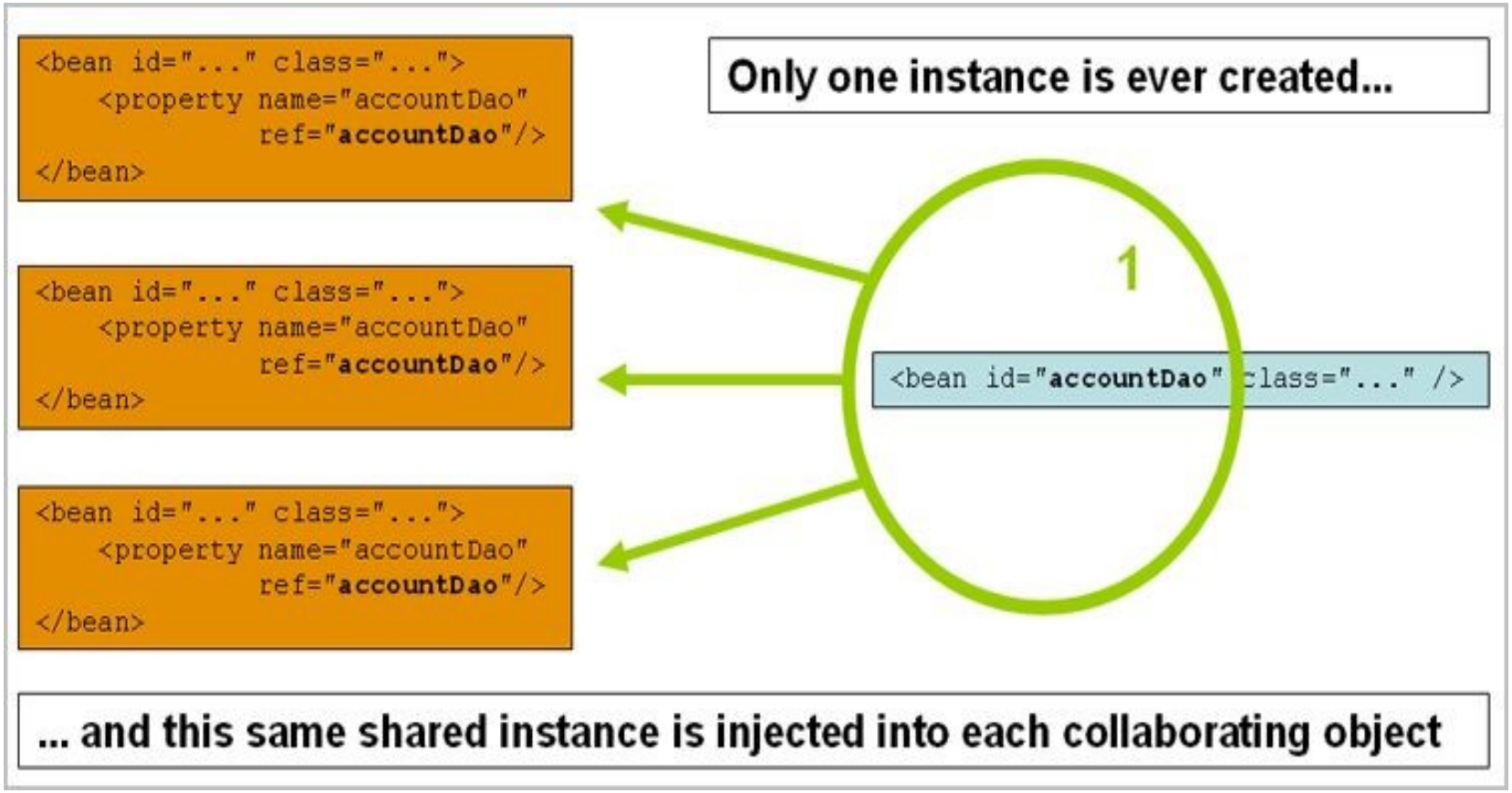
Conteneur léger – Spring IoC

- Injection automatique des collaborateurs (les modes de «Autowired» de Spring)

Mode	Description
no	Aucun autowired n'est effectué, et les dépendances sont assignées explicitement. Il s'agit du mode par défaut
byName	Spring recherche un bean ayant le même nom que la propriété pour réaliser l'injection
byType	Spring recherche un bean ayant le même type que la propriété pour réaliser l'injection
constructor	Similaire au byType, mais fondé sur les types des paramètres du ou des constructeurs
autodetect	Choisit d'abord l'injection par constructeur.

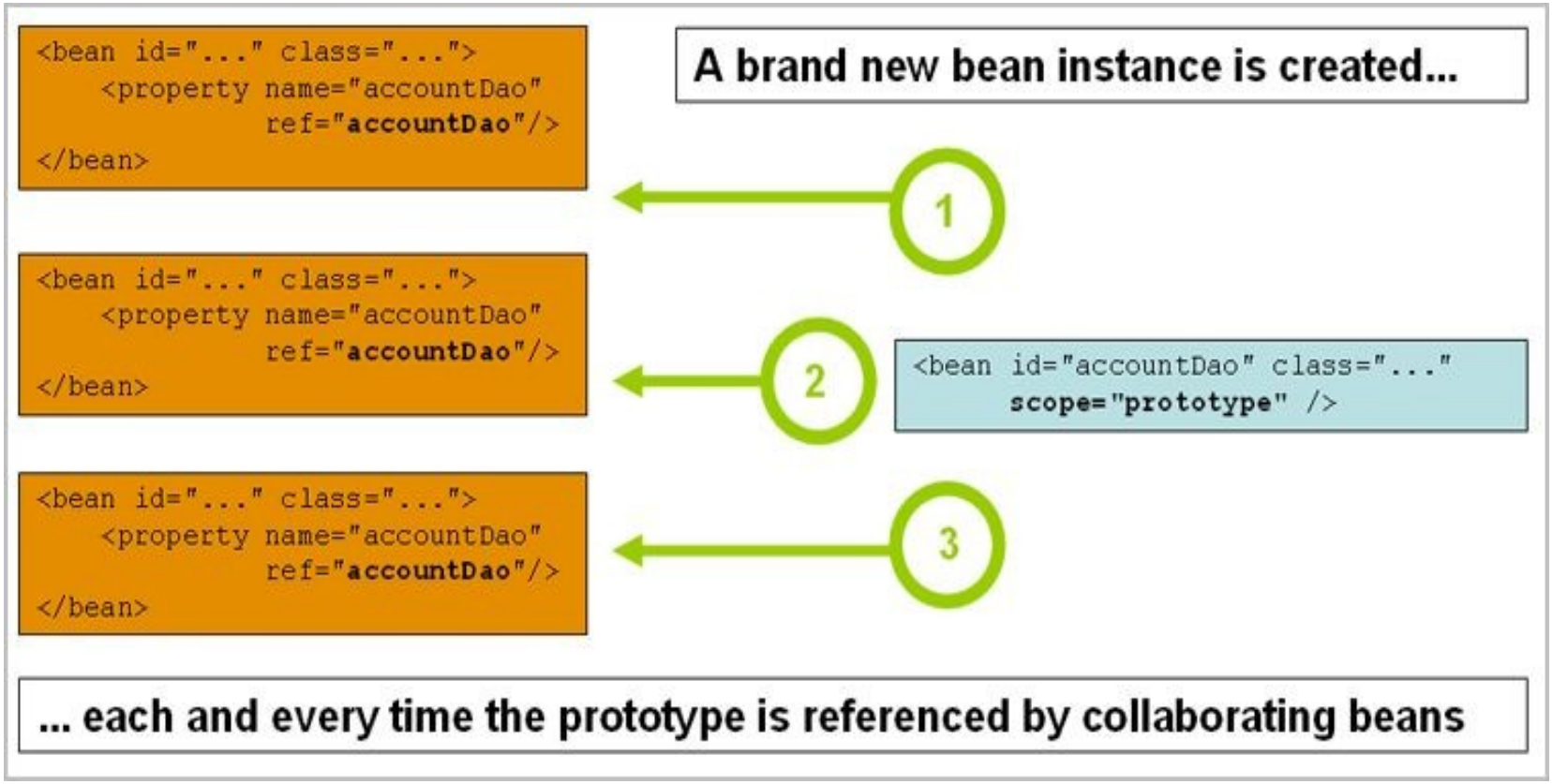
Conteneur léger – Spring IoC

- Sélection du mode d'instanciation, ou portée
  - Singleton



Conteneur léger – Spring IoC

- Sélection du mode d'instanciation, ou portée
  - Prototype



Conteneur léger – Spring IoC

- Sélection du mode d'instanciation, ou portée
  - Request: une instance est créée pour une nouvelle requête HTTP. Portée uniquement valable dans le contexte d'une application web
  - Session: une instance est créée pour une session HTTP. Portée uniquement valable dans le contexte d'une application web
  - Global session: une instance créée pour une session HTTP globale. Portée uniquement valable dans le contexte de Portlet

Spring AOP

- Notions de base de AOP
  - **Aspect**: entité logicielle qui capture une fonctionnalité transversale à une application
  - **Point de jonction**: point dans l'exécution d'un programme autour duquel un ou plusieurs aspects peuvent être ajoutés
  - **Coupe**: désigne un ensemble de points de jonction
  - **Greffon**: bloc de code définissant le comportement d'un aspect
  - **Tisseur d'aspects**: programme qui réalise une opération d'intégration entre un ensemble de classes et un ensemble d'aspects

Spring AOP

- Implémentation de la gestion des traces avec Spring AOP

```
public class MonService {
        public String hello(String msg){
                String s = "Hello "+msg;
                System.out.println(s);
                return s;
        }
}
```

MonService.java

```
public class MonServiceTest extends TestCase {
        public void testMonService() {
                ApplicationContext context = new
                ClassPathXmlApplicationContext( new String[]
                        {"springContext.xml"} );
                MonService monService = (MonService)
                context.getBean("monService");
                monService.hello("from Spring !");
        }
}
```

MonServiceTest .java

Spring AOP

```java
public class MonLogger {
	//Cette méthode est appelée à chaque fois (et avant) qu'une
	//méthode du package fr.jcertif.service est
	//interceptée
	public void logMethodEntry(JoinPoint joinPoint) {
		Object[] args = joinPoint.getArgs();
		//Nom de la méthode interceptée
		String name = joinPoint.getSignature().toLongString();
		StringBuffer sb = new StringBuffer(name + " called
		with: [");
		//Liste des valeurs des arguments reçus par la
		//méthode
		for(int i = 0; i < args.length; i++) {
			Object o = args[i];
			sb.append("'"+o+"'");
			sb.append((i == args.length - 1) ? "" : ", ");
		}
		sb.append("]");
		System.out.println(sb);
	}

	//Cette méthode est appelée à chaque fois (et après) qu'une
	//méthode du package fr.jcertif.service est interceptée
	//Elle reçoit en argument 'result' qui est le retour de la
	//méthode interceptée
	public void logMethodExit(StaticPart staticPart, Object
	result) {
		//Nom de la méthode interceptée
		String name = staticPart.getSignature().toLongString();
		System.out.println(name + " returning: [" + result + "]");
	}
}
```

```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:aop="http://www.springframework.org/schema/aop"
	xsi:schemaLocation="
	http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring- beans-
	2.5.xsd
	http://www.springframework.org/schema/aop
	http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

	<!-- Debut de la configuration AOP -->
	<aop:config>
		<aop:pointcut id="servicePointcut"
		expression="execution(* fr.jcertif.service.*.*
		(..))"/>
		<aop:aspect id="loggingAspect" ref="monLogger">
			<aop:before method="logMethodEntry" pointcut-
			ref="servicePointcut"/>
			<aop:after-returning method="logMethodExit"
			returning="result"
				pointcut-ref="servicePointcut"/>
		</aop:aspect>
	</aop:config>

	<bean id="monLogger" class="fr.jcertif.aop.MonLogger"/>
	<!-- Fin de la configuration AOP -->

	<bean name="monService"
		class="fr.jcertif.service.MonService" />
</beans>
```

Questions ?