# Spring4 quick start

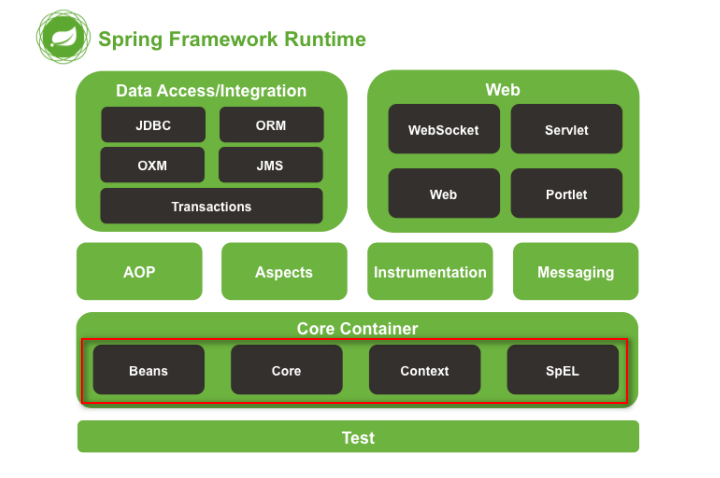
## Content

* Spring第一天：Spring的概述、SpringIOC入门（XML）、Spring的Bean管理、Spring属性注入
* Spring第二天：Spring的IOC的注解方式、Spring的AOP开发（XML）
* Spring第三天：Spring的AOP的注解开发、Spring的声明式事务、JdbcTemplate。
* Spring第四天：SSH的整合、HibernateTemplate的使用、OpenSessionInViewFilter的使用。

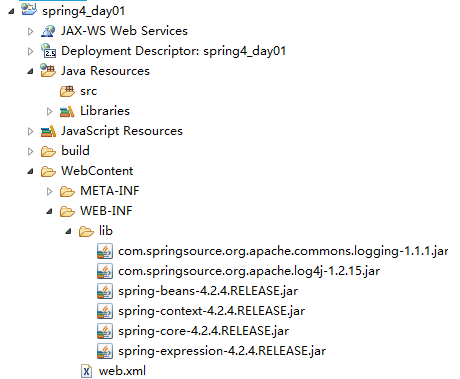
**Doc: $Spring/doc/pdf/…….**

## IOC (Inversion of Control) 控制反转

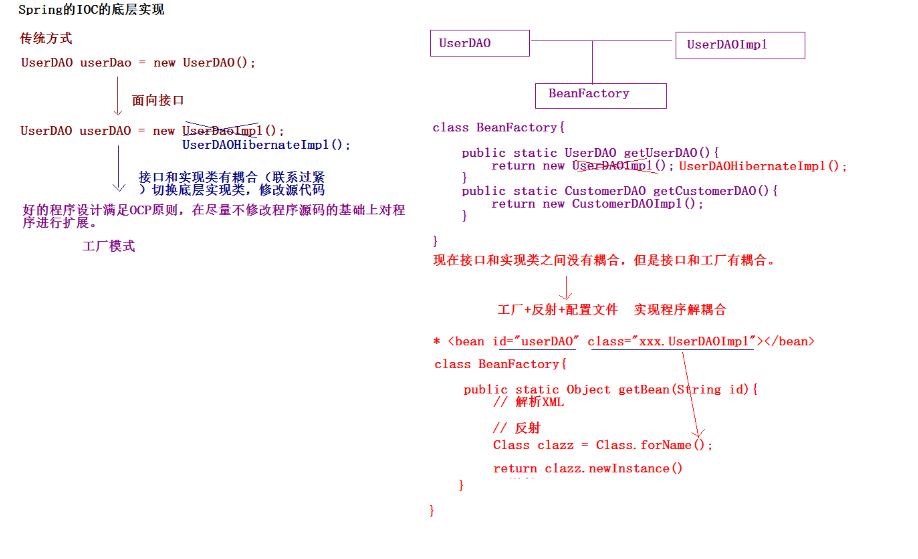
控制反转：将对象的创建权反转给（交给）Spring。



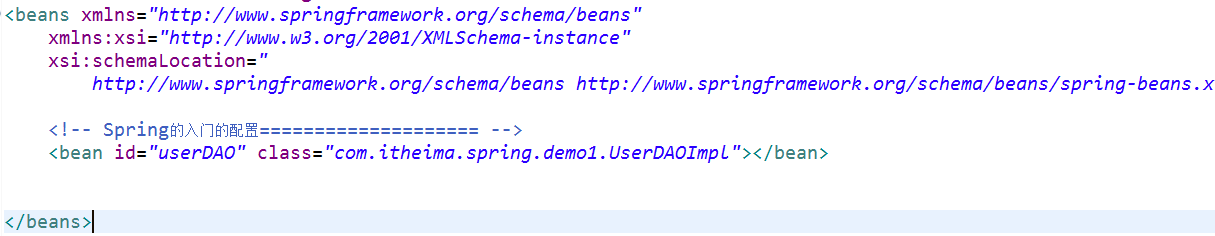
### Basic Jars



### IOC design ideas



### Configuration applicationContext.xml



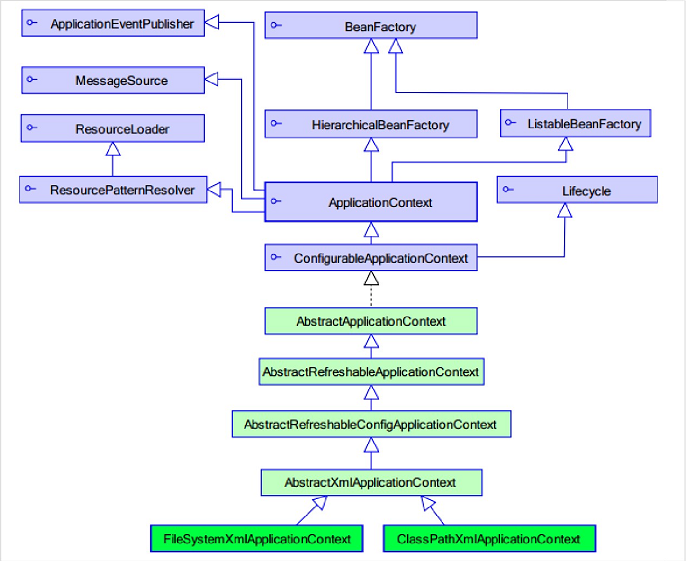
### IOC & DI (Dependency Injection)

IOC use reflection then invoke implementation’s default constructor (no arguments) to create object instance, the object’s attribute is null. That’s useless.

So, DI is for that purpose. Set class attributes.

### Spring IOC use factory design panttern

**Spring’s Bean factory hierarchy**



**BeanFactory & ApplicationContext**

* BeanFactory: deprecated**,** 用getBean的时候，才会生成类的实例。( get into bean’s constructor and proceed DI)
* ApplicationContext：加载配置文件的时候，就会将Spring管理的类都实例化。

ApplicationContext两个实现类

* + **ClassPathXmlApplicationContext ：加载类路径下的配置文件**
  + FileSystemXmlApplicationContext ：加载文件系统下的配置文件

### <bean>标签的id和name的配置

id :使用了约束中的唯一约束。里面不能出现特殊字符的。

name :没有使用约束中的唯一约束（理论上可以出现重复的，但是实际开发不能出现的）。里面可以出现特殊字符。

Spring和Struts1框架整合的时候

<bean name=”/user” class=””/>

#### Bean的生命周期的配置（了解）

* init-method :Bean被初始化的时候执行的方法
* destroy-method :Bean被销毁的时候执行的方法（Bean是单例创建，工厂关闭）

#### Bean的作用范围的配置（重点）

* scope ：Bean的作用范围
  + **singleton ：默认的，Spring会采用单例模式创建这个对象。**
  + **prototype ：多例模式。（Struts2和Spring整合一定会用到）**
  + request ：应用在web项目中，Spring创建这个类以后，将这个类存入到request范围中。
  + session ：应用在web项目中，Spring创建这个类以后，将这个类存入到session范围中。
  + globalsession ：应用在web项目中，必须在porlet环境下使用。但是如果没有这种环境，相对于session。

### Spring的Bean的实例化方式（了解）

Bean已经都交给Spring管理，Spring创建这些类的时候，有几种方式：

#### 无参构造方法的方式（默认）

configure



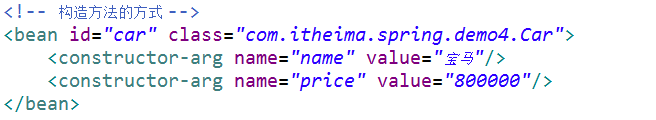
#### Static factory

#### 实例化工厂

# Spring DI

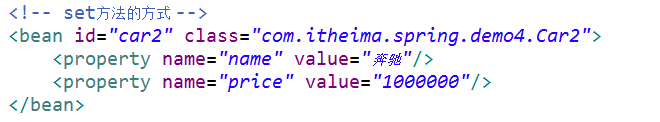
#### 构造方法的方式的属性注入

* 构造方法的属性注入

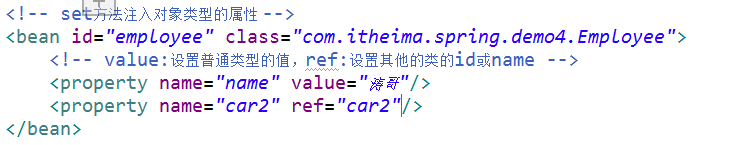


#### Set方法的方式的属性注入

* Set方法的属性注入

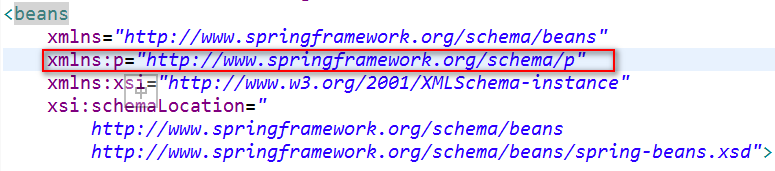


* Set方法设置对象类型的属性

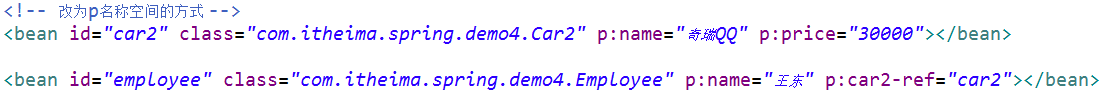


#### P名称空间的属性注入（Spring2.5以后）

* 通过引入p名称空间完成属性的注入：
  + 写法：
    - 普通属性 p:属性名=”值”
    - 对象属性 p:属性名-ref=”值”
* P名称空间的引入

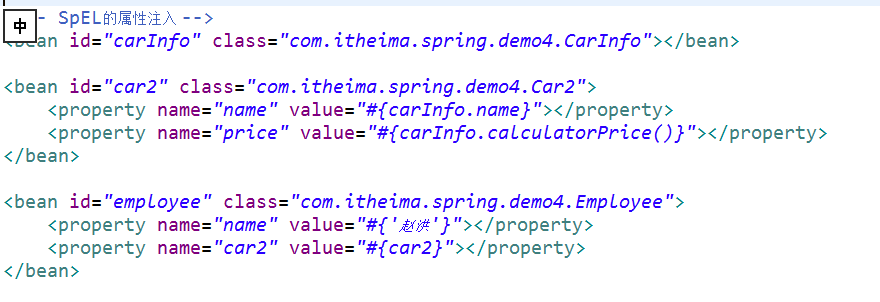


* 使用p名称空间



#### SpEL的属性注入（Spring3.0以后）

* SpEL：Spring Expression Language，Spring的表达式语言。
  + 语法：
    - #{SpEL}



### 集合类型属性注入(了解)

#### 配置

<!-- Spring的集合属性的注入============================ -->

<!-- 注入数组类型 -->

<bean id=*"collectionBean"* class=*"com.itheima.spring.demo5.CollectionBean"*>

<!-- 数组类型 -->

<property name=*"arrs"*>

<list>

<value>王东</value>

<value>赵洪</value>

<value>李冠希</value>

</list>

</property>

<!-- 注入list集合 -->

<property name=*"list"*>

<list>

<value>李兵</value>

<value>赵如何</value>

<value>邓凤</value>

</list>

</property>

<!-- 注入set集合 -->

<property name=*"set"*>

<set>

<value>aaa</value>

<value>bbb</value>

<value>ccc</value>

</set>

</property>

<!-- 注入Map集合 -->

<property name=*"map"*>

<map>

<entry key=*"aaa"* value=*"111"*/>

<entry key=*"bbb"* value=*"222"*/>

<entry key=*"ccc"* value=*"333"*/>

</map>

</property>

</bean>

## Spring的分模块开发的配置

### 分模块配置

#### 在加载配置文件的时候，加载多个



#### 在一个配置文件中引入多个配置文件



# Spring IOC Annotations

### Configuration

<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">  
  
    <!-- set base package using spring annotation -->  
    <context:component-scan base-package="com.itheima.dao.\*></context:component-scan>  
</beans>

1. Import context xml namespace
2. Set component scan, this is for class level annotation use, when use xml + annotation spring develop, xml for bean management, annotation for attribute injection. Can be omitted.

### @Annotations

## Bean annotation \*\*\*@Conponent("name")\*\*\*  
subtypes  
- @Service  
- @Controller  
- @Repository  
  
## Spring DI  
1. common  
    - @Value("value")  
2. Bean  
    - @Autowired  
    - @Qualifier  
  
    - or @Resource(name="value")  
3. others  
    - lifecycle  
        - @PostConstruct  
        - @PreDestroy  
    - scope  
        - @Singleton  
        - @Prototype  
        - @Session  
        - @GlobalSession  
        - @Request

# Spring AOP

AOP, aspect oriented programming.

### AOP design ideas

### Question & solution

\*\*Question: wanna enhance save method, add some check\*\*

```java

class UserDao{

void save(){

// save user

}

}

```

\*\*common solution\*\*

```java

class UserDao{

void save(){

// save user

}

}

class UserDaoChild extends UserDao{

void check(){}

void save(){

check();

// save user

}

}

```

but some class is final, can't be inherited, there two solution.

\*\*\*Box design pattern, for class implements a interface\*\*\*

```java

interface IUserDao{

void save();

}

final class UserDao implements IUserDao{

@Override

void save(){

// save user

}

}

class UserDaoBox implements IUserDao{

UserDao userDao = null;

UserDaoBox(UserDao userDao){

this.userDao = userDao;

}

@Override

void save(){

check();

userDao.save();

}

void check();

}

···

Dynamic proxy design pattern

Interface IUserDao{ void save(); }

Class UserDaoImpl {

Void save() { }

}

Class UserDaoProxy

### Spring AOP Implementation

1. JDK Dynamic Proxy: use at earlier version. It’s limited by classes which must be interface derived.
2. Cglib Dynamic Proxy: (a Third Party proxy framework,)Deprecated, now is JAssist

### AOP terms

* 连接点：可以被拦截的点。
* 切入点：真正被拦截的点。
* 通知：增强方法
* 引介：类的增强
* 目标：被增强的对象
* 织入：将增强应用到目标的过程。
* 代理：织入增强后产生的对象
* 切面：切入点和通知的组合

### Basic AOP Flow

* + - * 引入jar包
      * 编写目标类并配置
      * 编写切面类并配置
      * 进行aop的配置

<aop:config>

<aop:pointcut **expression=”execution(表达式)”** id=”pc1”/>

<aop:aspect >

<aop:before method=”” pointcut-ref=”pc1”/>

</aop:aspect>

</aop:config>

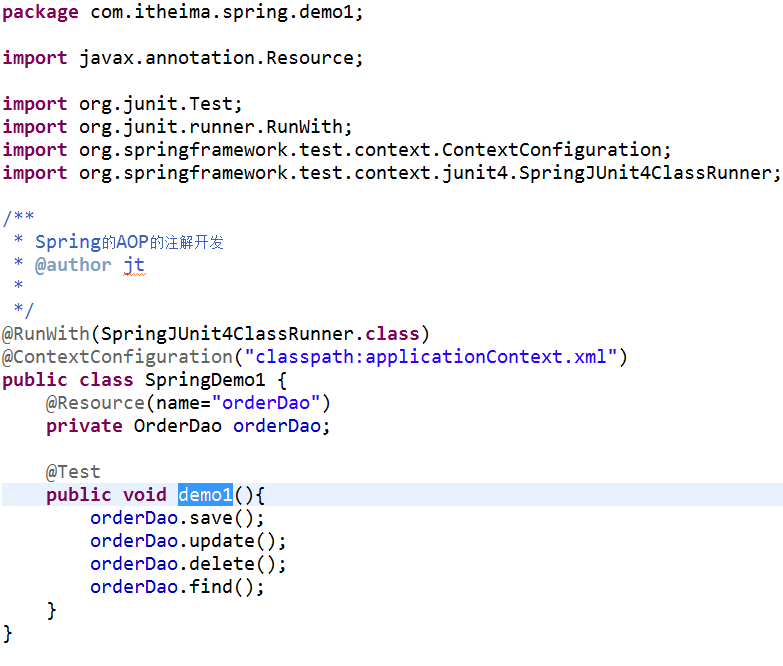
### 通知类型

* + - * 前置通知
      * 后置通知
      * 环绕通知
      * 异常抛出通知
      * 最终通知

# Spring Unit Test

Import spring unit test jar. **Spring-test-xxx.jar**

Example:



# Spring AOP Annotations (AspectJ based)

### Jar and Xml Configuration

* Xml namespace and schema location
* 在配置文件中开启注解的AOP的开发

<aop:aspectj-autoproxy/>

### @Annotations

1. **@Before**
2. **@AfterReturning**
3. **@Around**
4. **@AfterThrowing**
5. **@After**

Example:

