

# 1.注解不需要再配置容器文件中，也可以通过注解的方法来进行，但是不好改变

//自己定义的中间的方法 这样就不要 init-method

@PostConstruct

**public** **void** zhangInit(){

System.*out*.println("这个是我自己的方法");

}

//自己定义的

@PreDestroy

**public** **void** mydestory(){

System.*out*.println("这个是我自己的销毁方法 释放各种资源，也许看不到");

}

2.

# 2.一般我们不使用接口的InitializingBean,DisposableBean，都是自己配置



# 3.注入集合类型的属性

## 3.1设计两个bean，配置文件中会建立他们之间的联系

**public** **class** Employee {

**private** String name;

**private** **int** id;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**public** **class** Department {

**private** String name;

**private** String [] empName;//����

**private** List<Employee> empList;//list����

**private** Set<Employee> empsets;//set����

**private** Map<String,Employee> empMaps;//map����

**private** Properties pp;//Properties��ʹ��

## 3.2配置容器中进行配置信息

### 3.2.1给数组进行注入

**private** String [] empName;//����

<!-- 给数组注入值 -->

<property name=*"empName"*>

<list>

<value>小明</value>

<value>小明小明</value>

<value>小明小明小明小明</value>

</list>

</property>

#### 使用数组

**for**(String emName:department.getEmpName()){

System.*out*.println(emName);

}

### 3.2.2给List注入对象

**private** List<Employee> empList;//list����

<!-- 给list注入值 list 中可以有相当的对象 -->

<property name=*"empList"*>

<list>

<ref bean=*"emp2"* />

<ref bean=*"emp1"*/>

<ref bean=*"emp1"*/>

<ref bean=*"emp1"*/>

<ref bean=*"emp1"*/>

<ref bean=*"emp1"*/>

<ref bean=*"emp1"*/>

</list>

</property>

#### 使用list

System.*out*.println("\*\*\*\*\*\*\*\*\*\*ͨ��list����ȡ�����\*\*\*\*\*");

**for**(Employee e : department.getEmpList()){

System.*out*.println("name="+e.getName()+" "+e.getId());

}

#### 3.2.2.2 Employee对象的配置

<bean id=*"emp1"* class=*"com.hsp.collection.Employee"*>

<property name=*"name"* value=*"北京"*/>

<property name=*"id"* value=*"1"*/>

</bean>

<bean id=*"emp2"* class=*"com.hsp.collection.Employee"*>

<property name=*"name"* value=*"天津"*/>

<property name=*"id"* value=*"2"*/>

</bean>

### 3.2.3给Set注入对象，不能有相同的对象，相同就会覆盖

**private** Set<Employee> empsets;//set����

<!-- 给set注入值 set不能有相同的对象 -->

<property name=*"empsets"*>

<set>

<ref bean=*"emp1"* />

<ref bean=*"emp2"*/>

<ref bean=*"emp2"*/>

<ref bean=*"emp2"*/>

<ref bean=*"emp2"*/>

</set>

</property>

#### 使用Set

System.*out*.println("\*\*\*\*\*\*\*\*\*\*ͨ��set����ȡ�����\*\*\*\*\*");

**for**(Employee e : department.getEmpsets()){

System.*out*.println("name="+e.getName());

}

### 3.2.4给Map注入对象

**private** Map<String,Employee> empMaps;//map����

<!-- 给map注入值 map只有key不一样，就可以装配value -->

<property name=*"empMaps"*>

<map>

<entry key=*"11"* value-ref=*"emp1"* />

<entry key=*"22"* value-ref=*"emp2"*/>

<entry key=*"22"* value-ref=*"emp1"*/>

</map>

</property>

#### 使用 Map

Map<String,Employee> empmaps=department.getEmpMaps();

Iterator it=empmaps.keySet().iterator();

**while**(it.hasNext()){

String key=(String) it.next();

Employee emp=empmaps.get(key);

System.*out*.println("key="+key+" "+emp.getName());

}

System.*out*.println("\*\*\*\*\*\*\*ͨ��map����ȡ����� ��෽��\*\*\*\*");

//2.��෽��

**for**(Entry<String,Employee> entry1:department.getEmpMaps().entrySet()){

System.*out*.println(entry1.getKey()+" "+entry1.getValue().getName());

}

### 3.2.5给Properties注入

**private** Properties pp;//Properties��ʹ��

<!-- 给属性集合配置 -->

<property name=*"pp"*>

<props>

<prop key=*"pp1"*>abcd</prop>

<prop key=*"pp2"*>hello</prop>

</props>

</property>

#### 使用Properties

Properties pp=department.getPp();

//System.out.println(pp.get("pp1").toString());

**for**(Entry<Object,Object> entry:pp.entrySet()){

System.*out*.println(entry.getKey().toString()+" "+entry.getValue().toString());

}

System.*out*.println("\*\*\*\*\*ͨ��Enumerationȡ��\*\*\*\*\*");

Enumeration en= pp.keys();

**while**(en.hasMoreElements()){

// Entry<Object,Object> elment= (Entry<Object, Object>) en.nextElement();

// System.out.println(elment.getKey()+" "+elment.getValue());

String key=(String) en.nextElement();

System.*out*.println(key+" "+pp.getProperty(key));

}

}

# 4.内部bean

<bean id=”foo” class=”....Foo”>

<property name=”属性”>

<!—第一方法引用-->

<ref bean=’bean对象名’/>

<!—内部bean-->

<bean>

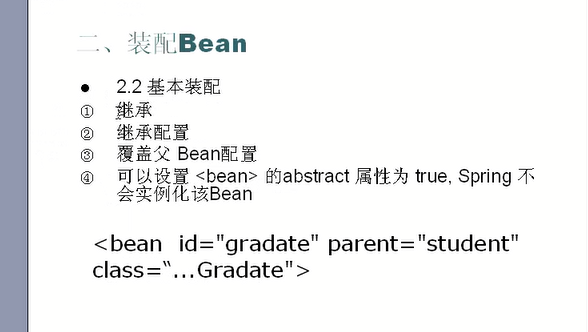
<properyt></property>

</bean>

</property>

</bean>

# 5.继承配置



## 1.创建两个bean，一个继承另一个

### 1.1 Student

**public** **class** Student {

**protected** String name;

**protected** **int** age;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

### 1.2 Gradate 继承 Student

**public** **class** Gradate **extends** Student {

**private** String degree;

**public** String getDegree() {

**return** degree;

}

**public** **void** setDegree(String degree) {

**this**.degree = degree;

}

}

## 2.注入

<!-- 配置一个学生对象 -->

<bean id=*"student"* class=*"com.hsp.inherit.Student"*>

<property name=*"name"* value=*"顺平"* />

<property name=*"age"* value=*"30"*/>

</bean>

<!-- 配置Grdate对象 -->

<bean id=*"grdate"* parent=*"student"* class=*"com.hsp.inherit.Gradate"*>

<!-- 如果自己配置属性name,age,则会替换从父对象继承的数据 -->

<property name=*"name"* value=*"小明"*/>

<property name=*"degree"* value=*"学士"*/>

</bean>

## 3.测试

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ac=**new** ClassPathXmlApplicationContext("com/hsp/inherit/beans.xml");

Gradate gradate=(Gradate) ac.getBean("grdate");

System.*out*.println(gradate.getName()+" "+gradate.getAge()+" "+gradate.getDegree());

}

# 6. 通过构造函数注入值

## 6.1创建一个bean

**public** **class** Employee {

**private** String name;

**private** **int** age;

**public** Employee(String name, **int** age) {

System.*out*.println("Employee(String name, int age) ");

**this**.name = name;

**this**.age = age;

}

**public** Employee() {

**super**();

}

## 6.2.注入

<!-- 配置一个雇员对象 -->

<bean id=*"employee"* class=*"com.hsp.constructor.Employee"*>

<!-- 通过构造函数来注入属性值 index 代表第几个属行值，-->

<constructor-arg index=*"0"* type=*"java.lang.String"* value=*"大明"* />

<constructor-arg index=*"1"* type=*"int"* value=*"23"* />

</bean>

## 6.3.测试

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ac=**new** ClassPathXmlApplicationContext("com/hsp/constructor/beans.xml");

Employee ee=(Employee) ac.getBean("employee");

System.*out*.println(ee.getName() +""+ ee.getAge());

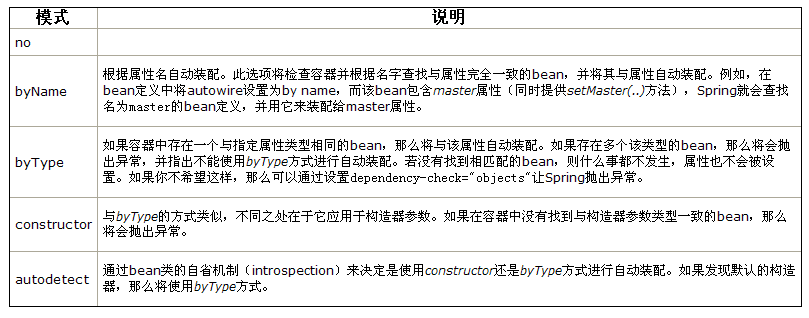
}

## 6.4注意：一定要让配置文件中的东西与构造函数对应，否则不会完成的，会报错

Set注入的缺点，是无法确定哪些是必须注入的，

构造函数 可以保证值是全的，只要有了构造函数，就会强制将数据注入

# 7.自动装配bean autowire的属性值



## 7.1 byName的用法: autowire

### 1.1创建一个两个bean，一个中包含另一个bean

**public** **class** Dog {

**private** String name;

**private** **int** age;

**public** String getName() {

**return** name;

}

**public** **class** Master {

**private** String name;

**private** Dog dog;

### 1.2进行注入，根据属行名字自动装配，这个名字和dog中的是一样的不能讲下面的id的dog变成dog1123，只能是dog因为id不能重复，所以这个就是唯一的

<!-- 配置一个master对象 -->

<bean id=*"master"* class=*"com.hsp.autowire.Master"* autowire=*"byName"* >

<property name=*"name"*>

<value>顺平</value>

</property>

<!--

传统的方法

<property name="dog" ref="dog111"></property>

--></bean>

<!-- 配置dog对象 -->

<bean id=*"dog"* class=*"com.hsp.autowire.Dog"*>

<property name=*"name"* value=*"小黄"*/>

<property name=*"age"* value=*"3"*/>

</bean>

### 1.3测试

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ac=**new** ClassPathXmlApplicationContext("com/hsp/autowire/beans.xml");

//��ȡ

Master master=(Master) ac.getBean("master");

System.*out*.println(master.getName()+" �� "+master.getDog().getName());

}

## 7.2 byType:寻找和属性类型相同的bean,找不到,装不上,找到多个抛异常。

注意：1.这个是判断容器中有没有一个dog的属行，不管名字是不是一样的

2.但是如果有了多个dog的类型，就会报错

## 7.3 constructor: autowire="constructor"

解释：如果有这个construstor 就会先有了 通过构造中有了这个东西，然后再去执行容器中的东西，试图通过构造函数将这个dog接收到，通过构造函数接收到

**public** Master(Dog dog) {

**this**.dog = dog;

}

## 7.4 autodetect

说明 ： autowire="autodetect"这个是在 bytype和construstor中选择一个，不确定

Construstor级别会稍微高一点

## 7.5 defualt <beans defualt-autorwire=“指定” />

<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"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

*http://www.springframework.org/schema/context/spring-context-2.5.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd"*

default-autowire=*"byName"*

>

这个需要在<beans defualt-autorwire=“指定” />

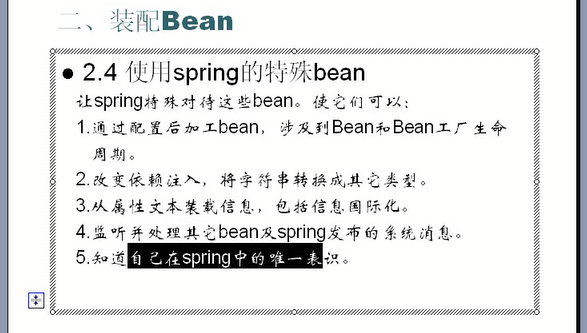
当你在<beans >指定了 default-atuowrite后， 所有的bean的 默认的autowire就是 指定的装配方法;

如果没有在<beans defualt-autorwire=“指定” /> 没有 defualt-autorwire=“指定” ，则默认是

defualt-autorwire=”no”

## 7.6 no: 不自动装配

# 8. 分散配置:使用spring的特殊bean,完成



## 1.建立一个工具 bean

**public** **class** DBUtil {

**private** String drivername;

**private** String url;

**private** String name;

**private** String pwd;

## 2. db.properties

name=scott

drivername=oracle:jdbc:driver:OracleDirver

url=jdbc:oracle:thin:@127.0.0.1:1521:hsp

pwd=tiger

## 3.进行联合注入，首先需要的是引入. db.properties

<!-- 引入我们的db.properties文件 -->

<context:property-placeholder location=*"classpath:com/hsp/dispatch/db.properties,classpath:com/hsp/dispatch/db2.properties"*/>

<!-- 配置一DBUtil对象 $占位符号 -->

<bean id=*"dbutil"* class=*"com.hsp.dispatch.DBUtil"*>

<property name=*"name"* value=*"${name}"* />

<property name=*"drivername"* value=*"${drivername}"* />

<property name=*"url"* value=*"${url}"* />

<property name=*"pwd"* value=*"${pwd}"* />

</bean>

## 4.测试

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ApplicationContext ac=**new** ClassPathXmlApplicationContext("com/hsp/dispatch/beans.xml");

DBUtil dbUtil=(DBUtil) ac.getBean("dbutil2");

System.*out*.println(dbUtil.getDrivername()+" "+dbUtil.getName());

}