Spring

优良特性：

1）非入侵式

2）依赖注入：DI，控制反转（ioc)最经典的实现

3）面向切面编程：AOP

4）容器

5）组件化：Spring实现了使用简单的组件配置组合了复杂的应用

配置文件 applicationContext.xml

<!--配置bean-->

配置方式：基于XML的方式，使用时全类名的方式。

<bean> :受Spring管理的一个Javabean对象

id：<bean>的唯一标识，在整个IOC容器中唯一不重复

class：指定javabean的全类名，目的是通过反射创建对象。、

Class c = Class.forName(""com.wang.class.Main");

Object obj = c.newInstance();//必须提供无参数构造器

<property>：给对象的属性赋值

name：指定属性名 -->对应的setter方法不是成员变量

value：指定属性值

```
<bean id ="main" class ="com.wang.class.Main">
    <property name="name" value="value"></property>
</bean>
```

IOC容器和Bean的配置

1.IOC和DI

1）IOC（inversion of control）：控制反转

反转控制的思想颠覆了应用程序组件获取资源的传统方式：反转了资源的获取方向--改由容器主动的将资源推送给需要的组件，开发人员不需要知道容器是如何创建资源对象的，只需要提供接收资源的方式即可，极大的降低了学习成本，提高了开发效率。这种行为也称为查找的被动方式。

2）DI（dependency injection）：依赖注入

IOC的另一种表述方式：即组件以一些预先定义好的方式（例如：setter方法）接受来自容器的资源注入。相对于IOC而言，这种表述更直接。

IOC描述的是一种思想，而DI是对IOC思想的具体实现。

3）IOC容器在Spring中实现

获取IOC容器：

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
```

获取bean方法：

```
1.Main main = (Main)ctx.getBean("main");//利用id获取，得到的是Object类型，强转
```

```
2.Main main = ctx.getBean(Main.Class);//利用class获取，不唯一
```

```
3.Main main= ctx.getBean("main",Main.class);//以上两种的结合，推荐使用
```

2.给bean的属性赋值

DI依赖注入的方式

1.通过bean的setXxx方法

property标签

2.通过bean的构造器

前提需要定义有参构造

value:构造器参数的实参值

index:构造器参数的下标，从0开始；

type:指定参数的类型

<constructor-arg value="" index="" type= “” ></constructor-arg>

p命名空间

通过bean的setXxx方法通过bean的setXxx方法

<bean id ="main" class = "com.wang.class"

p:属性名="" >

</bean>

可以使用的值

1.字面量

可以用value=" “,也可以用value标签，不能同时使用

特殊字符可以在value标签中使用<![CDATA[]]>把字面值包裹起来

2.引用其他的bean

ref:引用其他bean的id，只能是IOC容器中的；

<property name ="main" ref="main"></property>

3.内部bean

//不需要id属性，内部bean只能在内部使用

<property name = "car">

<property name value ></property>

</property>

4.给bean的级联属性赋值

<property name = "car"></property>

<!--设置级联属性-->

<property name = "car.name" value="奥迪"></property>

5.null值

1.可以不写，默认是null

2.<property> *<null/>* </property>，可以使用专门的空值标签

3.集合属性

1.数组、List和Set、Map

<bean id ="mainList" class = "com.wang.mainList">

<property name = “cars” >

<!--构造List集合-->

<list>

<ref bean = "car"/>

<ref bean = "car1"/>

</list>

<!--Map集合-->

<map>

<entry key="" value-ref=""></entry>

</map>

</property>

</bean>

数组 List Set Map

<array>

<list> <list>

<set>

<Map>

2.集合bean

当bean会被多次引用，可以util标签

```
<util:list id = "listBean">
    <ref bean = "main">
    <bean></bean>...
</util:list>
<bean id ="mainList" class = "com.wang.mainList">
    <property name = “cars” ref = "listBean">
    </property>
</bean>
```

4.FactoryBean

ApplicationContext的getBean方法返回不是CarFactoryBean，而是CarFactoryBean重写的getObject方法决定的。

```
public class CarFactoryBean implements FactoryBean<Car>{
 //工厂bean具体创建的bean对象是由getObject方法返回的
 @Override
 public Car getObject() throws Exception{
 return new Car(参数);
 }
 //返回具体的bean对象的类型
 @Override
 public Class<?> getObjectType(){
  return Car.class;
  }
  //是否单例的
  @Override
  public boolean isSingleton(){
  return true;
  }
}
```

```
<bean id ="mainList" class = "com.wang.CarFactoryBean">
</bean>
```

5.bean的高级配置

bean的继承关系

```
<bean id ="address1" class="..">
    <property name="city" value="beijng"></property>
    <property name="street" value="changanjie"></property>
</bean>
<bean id ="address2"  <!--class=".."--> parent="address1" >
```

<!--<property name="city" value="beijng"></property>-->
<property name="street" value="wudaokou"></property>
</bean>
bean标签有个abstract属性 = true就是一个抽象bean；不能被创建对象，class可以 · 忽略不配置
bean之间的依赖
有的时候创建一个bean的时候需要保证另一个bean也被创建，这个时候我们称前
面的bean对后面的bean有依赖。
<!--依赖关系-->
<bean id = "address3" parent="address1" depends-on="address4">
</bean>
<bean id = "address4" class=".."></bean>

6.bean的作用域☆

1.bean的作用域：

scope的属性

1）“singleton”；

singleton：单例的（默认值），整个IOC容器中只能存在一个bean对象，而且在IOC容器对象被创建死，就创建的bean对象，后序每次通过getBean方法获取时，返回的都是同一个对象。

2）” prototype “

prototype：原型的/多例的 在整个IOC容器中可有多个bean对象，在IOC容器对象被创建时，不会创建原型的bean的对象，而是等到每次通过getBean方法获取bean对象时，才会创建一个新的bean返回。

3）” request “

request：一次请求对应一个bean对象。该作用域仅适用于WebApplicationContext环境。

4）“session”

session：同一个HTTP Session共享一个bean对象，不同的HTTP Session使用不同的bean，该作用域仅适用于WebApplicationContext环境。

7.bean的生命周期

1）Spring IOC容器可以管理bean的生命周期，Spring允许在bean生命周期特定的时间执行特定的任务。

2）Spring IOC容器对bean的生命周期进行管理的过程：

1>通过构造器或工厂方法创建bean实例

2>位bean的属性设置值和对其他bean的引用

3>调用bean的初始化方法

4>bean可以使用了

5>当容器关闭时，调用bean的销毁方法

3）在配置bean时，通过init-method和destroy-method属性为bean指定初始化和销毁方法

4）bean的后置处理器

1>bean后置处理器：对IOC容器所有的bean都起作用

```
public class MyBeanPostProcessor implements BeanPostProcessor{
 //在bean的生命周期的初始化方法之前执行
 //bean：正在被创建的bean对象
 //name：bean对象的id值
 @Override
 public Object postProcessBeforeInitialization(Object bean,String beanNam
e)throws BeansException{
 return bean;
 }
 //在bean的生命周期的初始化方法之后执行
  //bean：正在被创建的bean对象
  //name：bean对象的id值
  @Override
  public Object postProcessAfterInitialization(Object bean,String beanNam
e)throws BeansException{
  return bean;
  }
}
```

<!--配置后置处理器　Spring能自动识别是一个后置处理器-->

<bean class = ".....MyBeanPostProcessor"></bean>

8.引用外部属性文件

1>直接配置

```
<!-- 直接配置c3p0连接池  ComboPooledDataSource -->
<bean id ="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/mysql"></property>
    <property name="user" value="root"></property>
    <property name="password" value="1234"></property>
    <property name="initialPoolSize" value="5"></property>
    <property name="maxPoolSize" value="10"></property>
</bean>
```

2>使用外部的属性文件

db.properties文件

```
# k = v
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mysql
jdbc.username=root
jdbc.password=1234
```

引用属性文件

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="classpath:db.properties"></property>
</bean>
```