1.Spring是一个从实际开发中抽取出来的框架，它完成了大量开发中的通用步骤，留给开发者的仅仅是与特定应用相关的部分，从而大大地提高了企业应用的开发效率。

Spring 具有如下优点：

> 低侵入式设计， 代码的污染极低。

> 独立于各种应用服务器， 基于Spring 框架的应用，可以真正实现Write Once 、Run Anywhere的承诺。

> Spring 的DI 容器降低了业务对象替换的复杂性，提高了组件之间的解耦。

> Spring 的**AOP 支持允许将一些通用任务如安全、事务、日志等进行集中式处理，从而提供**

**了更好的复用。**

> Spring 的ORM 和DAO 提供了与第三方持久层框架的良好整合， 并简化了底层的数据库访间。

> Spring 的高度开放性，并不强制应用完全依赖于Spring， 开发者可自由选用Spring 框架的部分或全部。

2. **Spring框架的下载地址：---到2017.1.3最新版为4.3.5**

<http://repo.springsource.org/libs-release-local/org/springframework/spring/>

**Spring API 的在线地址：**

<http://docs.spring.io/spring/docs/current/javadoc-api/>

**Spring 涉及的相关jar包下载：**

<http://commons.apache.org/proper/commons-logging/download_logging.cgi>

**在Eclispe中安装Spring tools 套件插件：**

Spring Tool Suite(STS) for Eclispe 3.8.3.RELEASE

安装此插件后可以方便的创建和修改Spring的配置文件，并且支持提示的功能。

3.Spring项目的开发步骤：

A.创建一个动态Web项目或者Java项目；

B.导入Spring框架的相关jar包，若是Java项目需要加入到build path中，若是Web项目需要复制到WEB-INF下的lib文件夹中。

C.创建spring配置文件applicationContext.xml，以及其他的配置文件，比如log4j的配置文件。

4.当我们使用Spring 框架时， 必须使用Spring Core Container （即Spring 容器），它代表了Spring 框架的核心机制， Spring Core Container 主要由org.springframework.**core** 、

orgspringfamework**.beans** 和org.springframework.**context**、org.springframework.**expression** 四个包及其子包组成，主要提供Spring I oC 容器支持。

**控制反转(Inversion of Control, IoC)**：**###从调用者角度看**

使用Spring框架之后，调用者无需主动获取被依赖的对象，调用者只要被动接受Spring容器为调用者的成员变量赋值即可(只要配置一个<property.../> 子元素，Spring就会执行对应的Setter方法为调用者的成员变量赋值)。使用Spring框架之后，调用者获取被依赖对象的方式，由原来的**主动**获取变成了**被动**接受----Rod Johnson将这种方式称为控制反转！

**依赖注入(Denpendency Injection, DI)**：**###从Spring容器看**

Spring容器负责将被依赖对象赋值给调用者的成员变量----相当于为调用者注入它依赖的实例，Martine Fowler 称之为依赖注入

IoC和DI，其实意义完全相同，是同一个行为的两种表达，只是描述的角度不同而已！

5.Spring容器，创建、管理Java类，

要被管理的Bean配置有哪些配置方式？

---XML的配置文件和注解方式；

如何初始化容器？

---使用ApplicationContext接口的两个实现类

如何从容器中获取Bean？

---使用ApplicationContext接口的getBean()方法。

6.将Java类放入Spring容器中有两种方式：

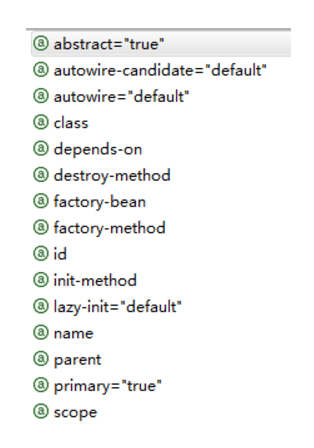
A.使用XML配置文件的方式

B.使用注解的配置方式

7.使用XML配置文件的方式：在applicationContext.xml中

* 1. 使用<bean>标签在<beans>标签中定义/声明一个bean；支持的属性：
  2. 使用classs属性指定要实例化的对象是什么类型，值通常为：包名.类名；
  3. 使用id属性，指定对象实例化后叫什么名字，以便后面获取时引用；

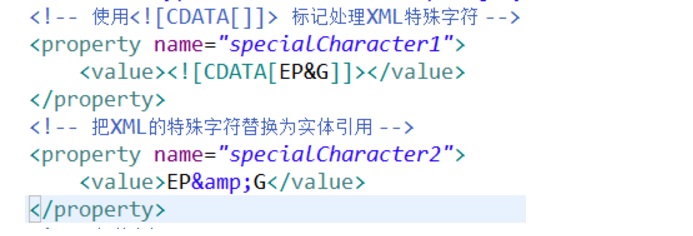
<bean id="dao" class="dao.impl.UserDaoImpl" />

* 1. 使用<property>标签为bean对象的属性赋值；**基本类型和String类型使用value，引用类型使用ref**！

<bean id="service" class="service.impl.UserServiceImpl">

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

</bean>



对于内部bean的



对于list、set、数组的<property>标签的配置是一样的,如：

<!-- 注入List类型 -->

<property name="list">

<list>

<!-- 定义List中的元素 -->

<value>乒乓球</value>

</list>

</property>



* 1. 可以使用<constructor-arg ref="...">,在对象初始化时调用含有参数的构造函数，完成属性的赋值

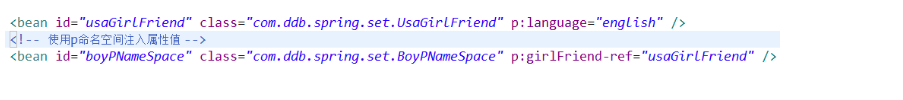
<bean id="boyConstructor" class="com.ddb.spring.set.BoyConstructor">

<!-- 通过构造方法注入对象 -->

<constructor-arg ref="chinaGirlFriend" />

</bean>

f.可以使用p命名空间，来为属性赋值：



8.使用注解的配置方式：

**使用注解标注要放入Spring容器的类；**使用的注解可以是：

@Component 标注一个普通的Spring Bean类

@Controller 标注一个控制器组件类

@Service 标注一个业务逻辑组件类

@Repository 标注一个DAO组件类

**使用@Autowired为Java类的引用属性自动注入值**；

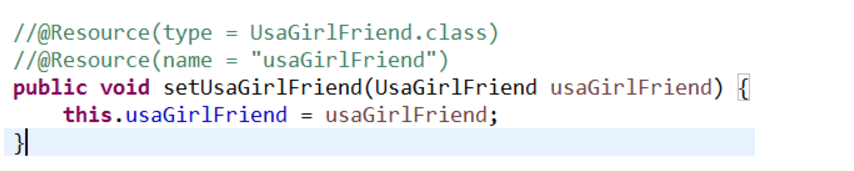
@Autowired

private UsaGirlFriend usaGirlFriend;

**使用@Resource为方法或属性注入需要的组件；**

---此注解并不属于Spring框架，而是属于javax的

@Resource注解，支持name和type属性，例如：



**XML的配置文件中配置组件自动扫描，将使用注解的Java类加入Spring容器中**：

---这一步必须有！

<!-- 自动扫描包含注解的包 -->

<context:component-scan base-package="com.ddb.spring.auto" />

9.如何在Java中初始化Spring容器

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

//以applicationContext.xml为参数初始化Spring容器上下文

说明：ApplicationContext 是Spring容器最常用的接口，有两个实现类如下：

**ClassPathXmlApplicationContext**：从类加载路径下搜索配置文件，并根据配置文件来创建Spring容器； ---这种方式使用更改普遍一些。

**FileSystemXmlApplicationContext**：从文件系统的相对路径或绝对路径，去搜索配置文件，并根据...

10.获取Spring容器中的Bean：

//获取容器中指定的id的Bean实例

UserService userService = (UserService) ctx.getBean("service");

主要有两个方法：

Object getBean(String id) :根据容器中Bean的id来获取指定的Bean，获取Bean**之后需要进行强制类型转换**；

T getBean(String id，Class<T> requiredType) :根据容器中Bean的id来获取指定的Bean，但是方法带了一个泛型参数，获取Bean之后**不需要进行强制类型转换**；

11..AOP ( Aspect Orient Programming），也就是面向切面编程,在Java EE应用中，常常通过AOP 来处理一些具有横切性质的系统级服务，如**事务管理、安全检查、缓存、对象池管理等**， AOP 已经成为一种非常常用的解决方案。

12.如何理解AOP：

A.在没有AOP的时候，可能会有这样的场景，在N多个方法中有相同功能代码，比如安全检查，但是存在问题是---重复，修改困难容易出现不一致，新方法还需要这些重复的代码！

B.重复的问题，我可以将相同的代码抽取出来，统一调用抽取出来的方法来解决，但是存在的问题是---每个方法都显示地知道了自己调用抽出来的方法，并且如果我们需要再增加日志的操作等，就要再增加显示调用日志的函数，再增加权限检查，事务检查，你会发现显示调用就会越来越多。

C.AOP就可以解决上面的问题，通过切面和切入点织入需要处理的操作，而原来的业务组件却毫无察觉。

AOP的原理和Listener有些类似，可以理解监听的方法被调用，从而触发了一些前置操作或者后置操作等！

13.AspectJ 的作用：**开发者无须修改源码，但又可以为组件的方法添加新的功能！**

14.AOP的实现分为两类：

静态AOP实现：编译阶段对程序进行修改，即实现对目标对象的增强，以AspectJ为代表；

动态AOP实现：在运行阶段动态生成AOP代理，以实现对目标对象的增强，以SpringAOP为代表。

15.AOP 框架具有如下两个特征：

> 各步骤之间的良好隔离性。

> 源代码无关性。

16.AOP编程，其中需要程序员参与的只有三个部分：

> 定义普通业务组件。

> 定义切入点， 一个切入点可能横切多个业务组件。

> 定义增强处理，增强处理就是在AOP 框架为普通业务组件织入的处理动作。

17.Spring 有如下两种选择来定义切入点和增强处理。

> 基于XML 配置文件的管理方式：使用Spring 配置文件来定义切入点和增强处理。

> 基于注解的“零配置”方式：使用@Aspect 、＠Pointcut 等注解来标注切入点和增强处理。

18.Spring AOP基于XML配置的方式：

在Spring 的配置文件中， 所有切面、切入点和增强处理都必须定义在**<aop:config>标签**元素内部。<beans ... ／＞元素下可以包含多个＜aop:config>元素， 一个<aop:config＞可以包含pointcut 、advisor 和aspect 元素， 且这三个元素必须按照此顺序来定义。

**A .定义切面：---使用<aop:aspect>标签**

**B .配置增强处理：**

使用的标签有：

<aop:before>、<aop：after>、<aop：after-returning>、

<aop:after-throwing>、<aop:around>

上面这些元素都不支持使用子元素， 但通常可指定如下属性。

> pointcut： 指定一个切入表达式， Spring 将在匹配该表达式的连接点时织入该增强处理。

> pointcut-ref： 指定一个已经存在的切入点名称， 通常pointcut 和pointcut-ref 两个属性

只需使用其中之一。

> method ： 该属性指定一个方法名， 指定切面Bean 的该方法将作为增强处理。

> throwing ： 只对<aop:after-throwing>元素有效， 用于指定一个形参名,AfteThrowing 增强处理方法可通过该形参访问目标方法所抛出的异常。

> returning ： 该属性只对<aop：after-returning>元素有效，用于指定一个形参名， AfterReturning增强处理方法可通过该形参访问目标方法的返回值。

**C.配置切入点：---使用<aop:pointcut>标签**

标签中可以使用的子标签，可以通过自动提示列出来！

例如：

<bean id="dao" class="dao.impl.UserDaoImpl" />

<bean id="service" class="service.impl.UserServiceImpl">

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

</bean>

<bean id="loggerBefore" class="aop.LoggerBefore" />

<bean id="loggerAfterReturning" class="aop.LoggerAfterReturning" />

<aop:config>

<aop:pointcut id="pointcut"

expression="execution(public void addNewUser(entity.User))" />

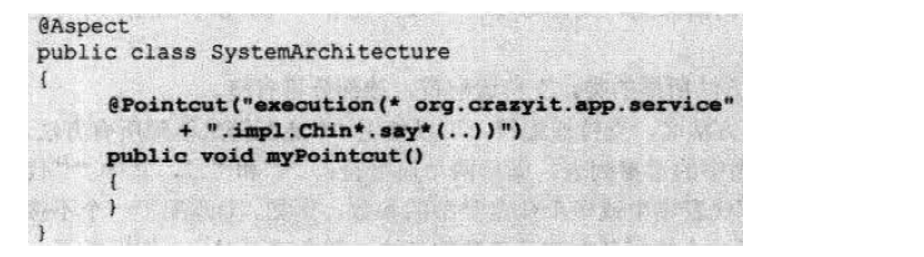
<aop:advisor pointcut-ref="pointcut" advice-ref="loggerBefore"/>

<aop:advisor pointcut-ref="pointcut" advice-ref="loggerAfterReturning"/>

</aop:config>

19.Spring AOP**基于注解**的配置方式：

为了启用Spring 对注解@AspetJ 切面配置的支持



**方式一：使用Spring 的XML Schema 配置方式**，在Spring的applicationContext.xml文件中的<beans>中增加：

<!-- 启动＠AspectJ 支持 -->

<aop : aspectj-autoproxy/>

**方式二：使用一个Bean后处理器**，在Spring 配置文件中增加如下片段来启用＠AspecJ 支持。

<!-- 启动＠AspectJ 支持 -->

<bean class="org.springframework.aop.aspectj.annotation.

AnnotationAwareAspectJAutoProxyCreator" />

上面的配置文件中的 AnnotationAwareAspecUAutoProxyCreator是一个Bean 后处理器，该Bean 后处理旅将会为容器中Bean 生成AOP 代理，为了在Spring 应用中启动@AspectJ 支持，还需要在应用的类加载路径中增加两个AspectJ 库：aspectjweaver.jar 和aspectjrt.jar 。

**A.定义切面：使用注解：@AspectJ** ，直接加到类的上一行

使用@Aspect 标注一个Java 类，该Java 类将会作为切面Bean，当我们使用@Aspect 来修饰一个Java 类之后， Spring 将不会把该Bean 当成级件Bean 处理，因此负责自动熠强的后处理Bean 将会略过该Bean，不会对该Bean 进行任何增强处理。

**B.定义增强处理：注解都是加在方法之上**

**@Before --只能在目标方法执行之前织入增强**

通常需要指定一个value 属性值，该属性值指定一个切入点表达式〈既可以是一个

己有的切入点， 也可以直接定义切入点表达式），用于指定该增强处理将被织入哪些切入点。

Before 处理无法阻止目标方法的执行， Before 增强处理执行时，目标方法还未获得执行的机会，所以Before 增强处理无法访问目标方法的返回值． 若要阻止目标方法执行，可以通过抛异常来实现。

例如：

@Before("execution(\* org.crazyit.app.service.impl .\*.\*(..))")

＠Before 注解 直接指定了切入点表达式，指定匹org.crazyit.app.service.impl 包下所有类的所有方法的执行作为切入点。

**@AfterReturning 在目标方法正常完成后被织入。**

例如：

@AfterReturning(returning="**rvt**",pointcut="execution(\* org.crazyit.app.service.impl .\*.\*(..))") //value属性与pointcut属性作用一样

public void log(Object **rvt**)

如果上面的log()方法中定义rvt 形参的类型是String，则该切入点只匹配lee 包下返回值类型为String 的所有方法．

可以访问目标方法的返回值，但是不可以改变返回值！

**@AfterThrowing 主要用于处理程序中未处理的异常。**

@AfterThrowing( throwing="**ex**"

,pointcut="execution(\* org.crazyit.app.service.impl .\*.\*(..))") //value属性与pointcut属性作用一样

public void doRecoveryActions (Throwable **ex**)

使用throwing 属性还有一个额外的作用：它可用于限定切入点只匹配指定类型的异

常--假如如在上面的doRecoveryActions()方法中定义了ex 形参的类型是NullPointer-

Exception ，则该切入点只匹配抛出NullPointerException 异常的情况． 上面doRecoveryActions()方法的ex 形参类型是Throwable ，这表明该切入点可匹配抛出任何异常的情况． AfterThrowing 处理虽然处理了该异常，但它不能完全处理异常，异常依然会传播到上一级调用者

**@After 不管目标方法如何结束，成功完成或者异常终止，它都会被织入；这种增强通常用于释放资源。**

@After("execution(\* org.crazyit.app.service.impl .\*.\*(..))")

After 增强处理的作用非常类似于异常处理中finally 块的作用一一无论如何，它总会在方法执行结束之后被织入，因此特别适用于进行资源回收。

**@Around 增强的超级大Boss**

@Around("execution(\* org.crazyit.app.service.impl .\*.\*(..))")

public void save(ProceedingJoinPoint pjp)

Around 增强处理甚至可以决定目标方法在什么时候执行，如何执行，甚至可以完全阻止目标方法的执行;可以改变执行目标方法的参数值;也可改变执行目标方法之后的返回值。

Around 增强处理的功能虽然强大，但通常需要在线程安全的环境下使用.

当定义一个Around 增强处理方法时，该方法的第一个形参必须是ProceedingJoinPoint 类型（至少包含一个形参〉， 在增强处理方法体内，调用ProceedingJoinPoint 的proceed（）方法才会执行目标方法一一这就是Around 增强处理可以完全控制目标方法执行时机、如何执行的关键； 如果程序没有调用ProceedingJoinPoint 的proceed()方法，则目标方法不会被执行。

调用ProceedingJoinPoint 的proceed（）方法时，还可以传入一个Object［］对象，该数组中的值将被传入目标方法作为执行方法的实参。

ProceedingJoinPoint 使用下面这些方法就可**访问到目标方法的相关信息**

> Object[] getArgs（）： 返回执行目标方法时的参数。

> Signature getSignature（），返回被增强的方法的相关信息。

> Object getTarget（）： 返回被织入增强处理的目标对象。

> Object getTh is（）， 返回AOP 框架为目标对象生成的代理对象。

配置applicationContext.xml，自动扫描Bean组件类与切面类：

<!-- 指定自动搜索Bean组件，自动搜索切面类 -->

<context:component-scan base-package="org.crazyit.app.service,org.crazyit.app.aspect">

<context:include-filter type="annotation" expression ="org.aspectj.lang.annotation.Aspect"/>

</context:component-scan>

<!-- 启动＠AspectJ 支持 -->

<aop : aspectj-autoproxy/>

**C.定义切入点：**

---上面的每个增强注解都含有【execution(\* org.crazyit.app.service.impl .\*.\*(..))"】完全可以提取出来，使用切入点代替。

**C1.定义切面内范围的切入点**

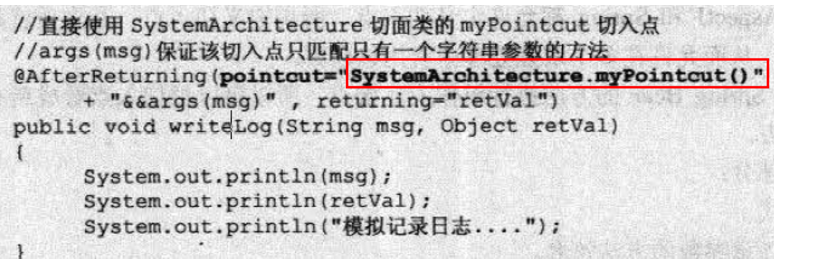
当myPointcut()的修饰符为private时，则为切面范围的切入点，

引用方式：@AfterReturning(pointcut="myPointcut()”,returning="retVal")

**C2.定义可以共享的切入点**

当myPointcut()的修饰符为public时，则为可共享范围的切入点，

引用方式：



附：

关于面向切面编程的一些术语。

> 切面（Aspect) ：业务流程运行的某个特定步骤，也就是应用运行过程的关注点， 关注点可能横切多个对象，所以常常也称为横切关注点。

> 连接点（ Joinpoint) ： 程序执行过程中明确的点，如方法的调用，或者异常的抛出。**Spring AOP**

**中，连接点总是方法的调用，或者说仅支持将方法调用作为连接点。**

> 增强处理（Advice ) : AOP 框架在特定的切入点执行的增强处理。处理有"around "," before "和“ after”等类型。

> 切入点（ Pointcut) ：可以插入增强处理的连接点。简而言之， 当某个连接点满足指定要求

时，该连接点将被添加增强处理，该连接点也就变成了切入点。

> 引入：将方法或字段添加到被处理的类中。Spring 允许引入新的接口到任何被处理的对象。

例如，你可以使用一个引入，使任何对象实现lsModified 接口，以此来简化缓存。

> 目标对象：被AOP 框架进行增强处理的对象，也被称为被增强的对象。如果AOP 框架是通

过运行时代理来实现的，那么这个对象将是一个被代理的对象。

> AOP 代理： AOP 框架创建的对象，简单地说，代理就是对目标对象的加强。Spring 中的AOP

代理可以是JDK 动态代理，也可以是CGL旧代理。前者为实现接口的目标对象的代理，后者为不实现接口的目标对象的代理。

> 织入（ Weaving ） ： 将增强处理添加到目标对象中，并创建一个被增强的对象（ AOP 代理）

的过程就是织入。织入有两种实现方式：编译时增强（例如AspectJ ）和运行时增强（例如

CGLIB ） 。Spring 和其他纯JavaAOP 框架一样，在运行时完成织入．