# **spring三大核心学习**

spring是一个轻型容器(light-weight Container)，其核心是Bean工厂(Bean Factory)，用以构造我们所需要的M(Model)

## **IOC（控制反转）**

下面是多个针对此理解的表达。

* 一个对象A依赖另一个对象B就要自己去new 这是高度耦合的 IOC容器的使用。   
  比如在B中使用A很多，哪一天A大量更改，那么B中就要修改好多代码。
* 通俗的理解是：平常我们new一个实例，这个实例的控制权是我们程序员，而控制反转是指new实例工作不由我们程序员来做而是交给spring容器来做。
* 针对一个接口，我们可能会写多个实现类，如果在代码中、程序中对实现类的对象进行创建，当想更换实现类时(使用其他的实现类)，就需要对代码进行更改。

一个使用实例   
通过spring的IOC功能，在xml配置文件中，给接口的实现类起一个名字“XXX”，代码中创建对象时，使用以下方式创建：

## **DI（Dependency Injection，依赖注入）**

首先应该明白两个问题：1，谁依赖谁；2，谁注入，注入什么？

* 利用xml的配置信息，在客户端代码中不用具体new任何的java对象了，java对象的创建工作，和对象中元素的赋值工作可以交给xml（spring）处理。
* 回答文中开头两个问题：1.客户端代码中，具体对象的创建依赖于xml文件（spring，即IOC容器）；2.是IOC容器注入，在运行期，根据xml的配置信息，将具体的对象注入到相应的bean中。

JavaBean：为了写出方便他人使用的类，于是规定，必须有一个零参的构造函数，同时还要用get/set方法，以便隐藏内部细节，方便使用和之后的代码更新。

* 针对一个JavaBean，为了使用它，首先需要new一个对象，之后需要对其中的set方法进行调用进而赋值。代码之间的联系变得很大，封装的特性渐渐变小。这样在修改代码时，就麻烦了。要成堆的更改，尤其是在不同团队分工开发的过程中，代码变更影响巨大。
* 通过控制反转（IOC）、依赖注入，new的同一种对象，在xml文件中都给他起一个小名，这样更改时只需要在xml文件中，将小名对应的类的具体路径更改了。不需要一个个.java文件替换。
* 在使用set方法传值时，如果针对具体的属性值，进行填写，更改起来也会麻烦一些，通过Spring来进行赋值，更改起来更加方便。

## **面向切面AOP**

可以通过预编译方式和运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术。

# **AOP（Aspect-Oriented Programming,面向切面编程）**

切面（Aepect）：横切关注点（跨越应用程序多个模块的功能）被模块化的对象；

通知（Advice）：切面必须要完成的工作；

目标（Target）：被通知的对象；

代理（Proxy）：像目标对象应用通知之后创建的对象；

连接点（Joinpoint）：程序执行的某个特殊位置，如类某个方法调用前、调用后、方法抛出异常后等。连接点由两个信息确定：方法表示的程序执行点；想对点表示的方位。

切点（pointcut）：每个类都拥有多个连接点，即连接点是程序类中客观存在的事务；

AOP通过切点定位到特定的连接点。

AOP的主要编程对象是切面（aopect）,而切面模块化**横切关注点**。

在应用AOP编程时，仍需要定义公共功能，但可以明确这个功能在哪里，以什么方式应用，并且不必修改受影响的类，这样的话横切关注点就被模块化到特殊的对象（切面）里。

AOP的好处：

　　每个事物逻辑位于一个位置，代码不分散，便于维护和升级；

　　业务模块更简洁，值包含核心业务代码