**Spring的两大特征（IOC和AOP）**

众所周知，Spring的核心特性就是IOC和AOP，IOC（Inversion of Control），即“控制反转”；AOP（Aspect-OrientedProgramming），即“面向切面编程”。参考书《Spring In Action》，下面分享一下我对这两大特性的个人理解。

**IOC:控制反转也叫依赖注入。利用了工厂模式**

**将对象交给容器管理，你只需要在spring配置文件总配置相应的bean，以及设置相关的属性，让spring容器来生成类的实例对象以及管理对象。在spring容器启动的时候，spring会把你在配置文件中配置的bean都初始化好，然后在你需要调用的时候，就把它已经初始化好的那些bean分配给你需要调用这些bean的类（假设这个类名是A），分配的方法就是调用A的setter方法来注入，而不需要你在A里面new这些bean了。**

**AOP，即面向切面编程，面向切面编程的目标就是分离关注点。我们以系统中常用到的事务管控举例子。在系统操作数据库的过程中，不可避免地要考虑到事务相关的内容。如果在每一个方法中都新建一个事务管理器，那么无疑是对代码严重的耦合和侵入。为了简化我们的开发过程（实际上spring所做的一切实现都是为了简化开发过程），需要把事务相关的代码抽成出来做为一个独立的模块。通过AOP，确认每一个操作数据库方法为一个连接点，这些连接点组成了一个切面。当程序运行到其中某个一个切点时，我们将事务管理模块顺势织入对象中，通过通知功能，完成整个事务管控的实现。这样一来，所有的操作数据库的方法中不需要再单独关心事务管理的内容，只需要关注自身的业务代码的实现即可。所有的事务管控相关的内容都通过AOP的方式进行了实现。简化了代码的内容，将目标对象复杂的内容进行解耦，分离业务逻辑与横切关注点。**

**从Spring的角度看，AOP最大的用途就在于提供了事务管理的能力。事务管理就是一个关注点，你的正事就是去访问数据库，而你不想管事务（太烦），所以，Spring在你访问数据库之前，自动帮你开启事务，当你访问数据库结束之后，自动帮你提交/回滚事务！**

**IOC**：IOC，另外一种说法叫DI（Dependency Injection），即依赖注入。它并不是一种技术实现，而是一种设计思想。在任何一个有实际开发意义的程序项目中，我们会使用很多类来描述它们特有的功能，并且通过类与类之间的相互协作来完成特定的业务逻辑。这个时候，每个类都需要负责管理与自己有交互的类的引用和依赖，代码将会变的异常难以维护和极度的高耦合。而IOC的出现正是用来解决这个问题，我们**通过IOC将这些相互依赖对象的创建、协调工作交给Spring容器去处理，每个对象只需要关注其自身的业务逻辑关系就可以了**。在这样的角度上来看，获得依赖的对象的方式，进行了反转，变成了**由spring容器控制对象如何获取外部资源**（包括其他对象和文件资料等等）。

举例：某一天，你生病了，但是你不清楚自己到底得了什么病，你只知道自己头疼，咳嗽，全身无力。这个时候你决定去药店买药，药店有很多种药，仅仅是治疗头疼就有好几十种，还有西药中药等区别。然后你自己看了看说明书，选择了一盒你自己觉得最能治疗自己病症的药，付钱吃药，期待可以早点好起来。

但是这个过程，对于一个病人来说，太辛苦了。头疼，咳嗽，全身无力，还要一个个的看药品说明书，一个个的比较哪个药比较好，简直是太累了。这个时候，你决定直接去医院看医生。

医生给你做了检查，知道你的病症是什么，有什么原因引起的；同时医生非常了解有哪些药能治疗你的病痛，并且能根据你的自身情况进行筛选。只需要短短的十几分钟，你就能拿到对症下药的药品，即省时又省力。

**在上面这个例子中，IOC起到的就是医生的作用，它收集你的需求要求，并且对症下药，直接把药开给你。你就是对象，药品就是你所需要的外部资源。通过医生，你不用再去找药品，而是通过医生把药品开给你。这就是整个IOC的精髓所在。**

**AOP**：面向切面编程，往往被定义为**促使软件系统实现关注点的分离的技术**。系统是由许多不同的组件所组成的，每一个组件各负责一块特定功能。除了实现自身核心功能之外，这些组件还经常承担着额外的职责。例如日志、事务管理和安全这样的核心服务经常融入到自身具有核心业务逻辑的组件中去。这些系统服务经常被称为横切关注点，因为它们会跨越系统的多个组件。

AOP的概念不好像IOC一样实例化举例，现在我们以一个系统中的具体实现来讲讲AOP具体是个什么技术。

**AOP，即面向切面编程，面向切面编程的目标就是分离关注点。**我们以系统中常用到的事务管控举例子。在系统操作数据库的过程中，不可避免地要考虑到事务相关的内容。如果在每一个方法中都新建一个事务管理器，那么无疑是对代码严重的耦合和侵入。为了简化我们的开发过程（实际上spring所做的一切实现都是为了简化开发过程），**需要把事务相关的代码抽成出来做为一个独立的模块。通过AOP，确认每一个操作数据库方法为一个连接点，这些连接点组成了一个切面。当程序运行到其中某个一个切点时，我们将事务管理模块顺势织入对象中，通过通知功能，完成整个事务管控的实现。这样一来，所有的操作数据库的方法中不需要再单独关心事务管理的内容，只需要关注自身的业务代码的实现即可。所有的事务管控相关的内容都通过AOP的方式进行了实现。简化了代码的内容，将目标对象复杂的内容进行解耦，分离业务逻辑与横切关注点。**

**从Spring的角度看，AOP最大的用途就在于提供了事务管理的能力。事务管理就是一个关注点，你的正事就是去访问数据库，而你不想管事务（太烦），所以，Spring在你访问数据库之前，自动帮你开启事务，当你访问数据库结束之后，自动帮你提交/回滚事务！**

下面介绍一下AOP相关的术语：

通知： 通知定义了切面是什么以及何时使用的概念。Spring 切面可以应用5种类型的通知：

前置通知（Before）：在目标方法被调用之前调用通知功能。

后置通知（After）：在目标方法完成之后调用通知，此时不会关心方法的输出是什么。

返回通知（After-returning）：在目标方法成功执行之后调用通知。

异常通知（After-throwing）：在目标方法抛出异常后调用通知。

环绕通知（Around）：通知包裹了被通知的方法，在被通知的方法调用之前和调用之后执行自定义的行为。

连接点：是在应用执行过程中能够插入切面的一个点。

切点： 切点定义了切面在何处要织入的一个或者多个连接点。

切面：是通知和切点的结合。**通知和切点共同定义了切面的全部内容**。

引入：引入允许我们向现有类添加新方法或属性。

织入：是把切面应用到目标对象，并创建新的代理对象的过程。切面在指定的连接点被织入到目标对象中。在目标对象的生命周期中有多个点可以进行织入：

编译期： 在目标类编译时，切面被织入。这种方式需要特殊的编译器。AspectJ的织入编译器就是以这种方式织入切面的。

类加载期：切面在目标加载到JVM时被织入。这种方式需要特殊的类加载器(class loader)它可以在目标类被引入应用之前增强该目标类的字节码。

运行期： 切面在应用运行到某个时刻时被织入。一般情况下，在织入切面时，AOP容器会为目标对象动态地创建一个代理对象。SpringAOP就是以这种方式织入切面的。