IoC（Inversion of Control）

  (1). IoC（Inversion of Control）是指容器控制程序对象之间的关系，而不是传统实现中，由程序代码直接操控。控制权由应用代码中转到了外部容器，控制权的转移是所谓反转。 对于Spring而言，就是由Spring来控制对象的生命周期和对象之间的关系；IoC还有另外一个名字——“依赖注入（Dependency Injection）”。从名字上理解，所谓依赖注入，即组件之间的依赖关系由容器在运行期决定，即由容器动态地将某种依赖关系注入到组件之中。

(2). 在Spring的工作方式中，所有的类都会在spring容器中登记，告诉spring这是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

(3). 在系统运行中，动态的向某个对象提供它所需要的其他对象。

(4). 依赖注入的思想是通过反射机制实现的，在实例化一个类时，它通过反射调用类中set方法将事先保存在HashMap中的类属性注入到类中。 总而言之，在传统的对象创建方式中，通常由调用者来创建被调用者的实例，而在Spring中创建被调用者的工作由Spring来完成，然后注入调用者，即所谓的依赖注入or控制反转。 注入方式有两种：依赖注入和设置注入； IoC的优点：降低了组件之间的耦合，降低了业务对象之间替换的复杂性，使之能够灵活的管理对象。

AOP（Aspect Oriented Programming）

(1). AOP面向方面编程基于IoC，是对OOP的有益补充；

(2). AOP利用一种称为“横切”的技术，剖解开封装的对象内部，并将那些影响了 多个类的公共行为封装到一个可重用模块，并将其名为“Aspect”，即方面。所谓“方面”，简单地说，就是将那些与业务无关，却为业务模块所共同调用的 逻辑或责任封装起来，比如日志记录，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。

(3). AOP代表的是一个横向的关 系，将“对象”比作一个空心的圆柱体，其中封装的是对象的属性和行为；则面向方面编程的方法，就是将这个圆柱体以切面形式剖开，选择性的提供业务逻辑。而 剖开的切面，也就是所谓的“方面”了。然后它又以巧夺天功的妙手将这些剖开的切面复原，不留痕迹，但完成了效果。

(4). 实现AOP的技术，主要分为两大类：一是采用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；二是采用静态织入的方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码。

(5). Spring实现AOP：JDK动态代理和CGLIB代理 JDK动态代理：其代理对象必须是某个接口的实现，它是通过在运行期间创建一个接口的实现类来完成对目标对象的代理；其核心的两个类是InvocationHandler和Proxy。 CGLIB代理：实现原理类似于JDK动态代理，只是它在运行期间生成的代理对象是针对目标类扩展的子类。CGLIB是高效的代码生成包，底层是依靠ASM（开源的java字节码编辑类库）操作字节码实现的，性能比JDK强；需要引入包asm.jar和cglib.jar。     使用AspectJ注入式切面和@AspectJ注解驱动的切面实际上底层也是通过动态代理实现的。

(6). AOP使用场景：

Authentication 权限检查

Caching 缓存

Context passing 内容传递

Error handling 错误处理

Lazy loading　延迟加载

Debugging　　调试

logging, tracing, profiling and monitoring　日志记录，跟踪，优化，校准

Performance optimization　性能优化，效率检查

Persistence　　持久化

Resource pooling　资源池

Synchronization　同步

Transactions 事务管理

另外Filter的实现和struts2的拦截器的实现都是AOP思想的体现。