# Spring笔记

Spring优势：

1、**方便解耦，简化开发**

2、**AOP编程的支持**

**3、声明式事务的支持**

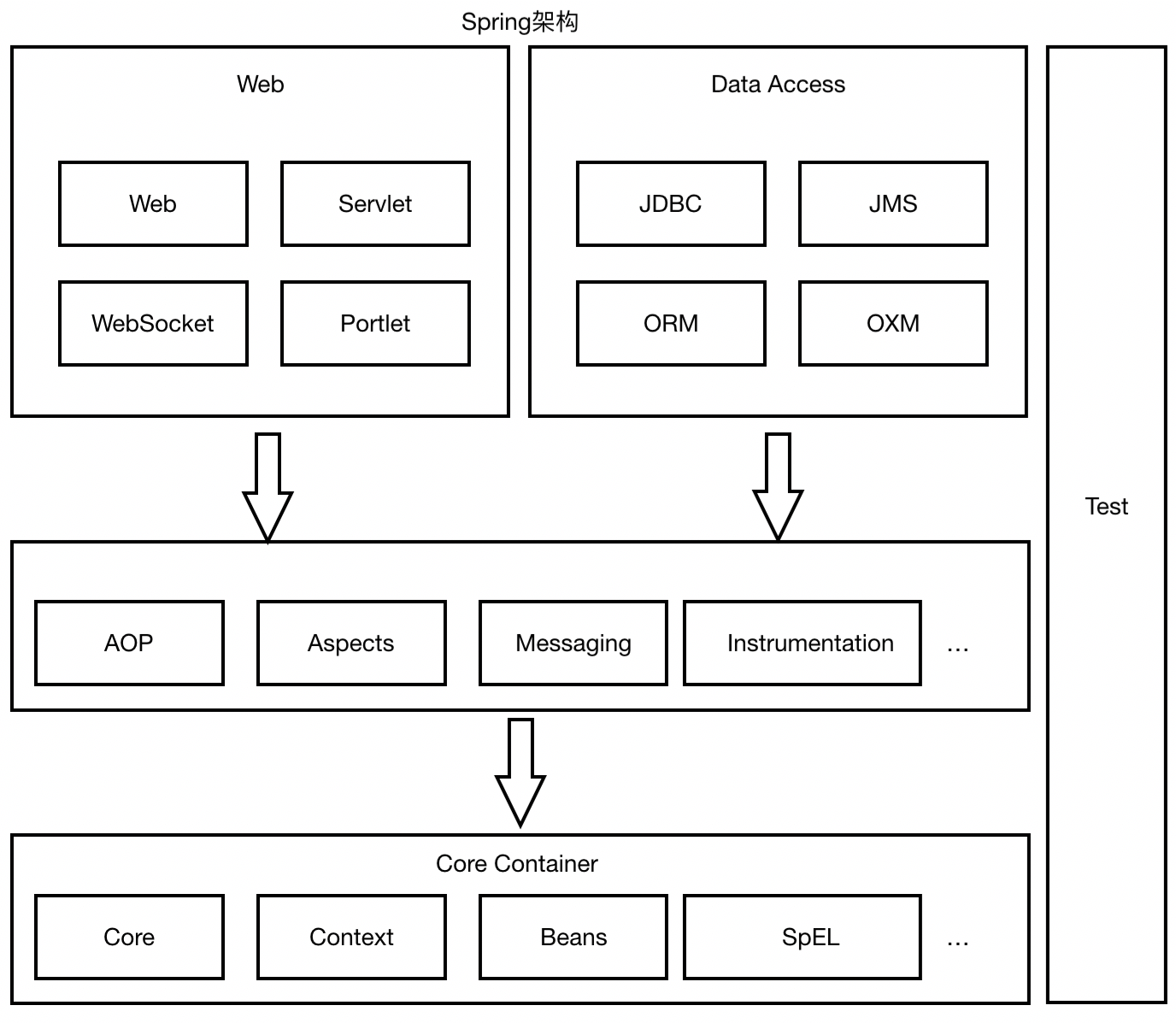
**4、方便程序的测试**

**5、方便集成各种优秀框架**

**6、降低JavaEE API的使用难度**

**7、源码是经典的 Java 学习范例**

**Spring核心结构**



IOC相关概念：

IoC Inversion of Control (控制反转/反转控制)，注意它是一个技术思想，不是一个技术实现 。描述的事情:Java开发领域对象的创建，管理的问题 。

控制:指的是对象创建(实例化、管理)的权利

反转:控制权交给外部环境了(spring框架、IoC容器)

解决问题：IoC解决对象之间的耦合问题

AOP相关概念：

AOP: Aspect oriented Programming 面向切面编程/面向方面编程

横切逻辑代码存在问题：

横切代码重复问题

横切逻辑代码和业务代码混杂在一起，代码臃肿，维护不方便

AOP解决问题：在不改变原有业务逻辑情况下，增强横切逻辑代码，根本上解耦合，避免横切逻辑代码重复

面向切面编程:

「切」:指的是横切逻辑，原有业务逻辑代码我们不能动，只能操作横切逻辑代码，所以面向横切逻辑

「面」:横切逻辑代码往往要影响的是很多个方法，每一个方法都如同一个点，多个点构成面，有一个

面的概念在里面

银行转账2个问题解决方案：

问题一：service和dao层耦合在一起

解决方案：1、使用工程模式进行解耦

2、使用反射自动注入

问题二：service层没有添加事务控制

解决方案：1、让两次update使用同一个connection连接

2、把事务控制添加到service层的方法上

Spring IOC配置方法：

纯xml模式：

实例化Bean的三种方法：

1、使用无惨构造函数 2、使用静态方法创建 3、使用实例化方法创建

xml + 注解组合模式：

|  |  |
| --- | --- |
| **xml形式** | **对应的注解形式** |
| 标签 | @Component("accountDao")，注解加在类上 bean的id属性内容直接配置在注解后面如果不配置，默认定义个这个bean的id为类 的类名首字母小写; 另外，针对分层代码开发提供了@Componenet的三种别名@Controller、 @Service、@Repository分别用于控制层类、服务层类、dao层类的bean定义，这 四个注解的用法完全一样，只是为了更清晰的区分而已 |
| 标签的 scope属 性 | @Scope("prototype")，默认单例，注解加在类上 |

DI依赖注入的注解实现方式：

@Autowired : 采取的策略为按照类型注入

纯注解模式：

将xml中遗留的内容全部以注解的形式迁移出去，最终删除xml，从Java配置类启动

lazy-init延迟加载

设置 lazy-init 为 true 的 bean 将不会在 ApplicationContext 启动时提前被实例化，而是第一次向容器 通过 getBean 索取 bean 时实例化的。

应用场景:

1、开启延迟加载一定程度提高容器启动和运转性能

2、对于不常使用的 Bean 设置延迟加载，这样偶尔使用的时候再加载，不必要从一开始该 Bean 就占 用资源

循环依赖：

两个或者两个以上的 Bean 互相持有对方，最终形成闭环。比如A 依赖于B，B依赖于C，C又依赖于A。

Spring中循环依赖场景有:

1、构造器的循环依赖(构造器注入)

2、Field 属性的循环依赖(set注入)

构造器的循环依赖问题无法解决，只能拋出 BeanCurrentlyInCreationException 异常，在解决 属性循环依赖时，spring采用的是提前暴露对象的方法。

Spring Aop：

连接点:方法开始时、结束时、正常运行完毕时、方法异常时等这些特殊的时机点，我们称之为连接点，项目中每个方法都有连接点，连接点是一种候选点

切入点:指定AOP思想想要影响的具体方法是哪些，描述感兴趣的方法

**众多的概念，目的就是为了锁定要在哪个地方插入什么横切逻辑代码**

代理选择：

默认情况下，Spring会根据被代理对象是否实现接口来选择使用JDK还是CGLIB。当被代理对象没有实现 任何接口时，Spring会选择CGLIB。当被代理对象实现了接口，Spring会选择JDK官方的代理技术，不过 我们可以通过配置的方式，让Spring强制使用CGLIB。

配置方法：

1. 纯xml 2、xml + 注解 3、纯注解