IOC/DI

上面的简单案例，简单了解了一下IOC的魔力。现在说什么是IOC。

IOC : 思想反转了资源获取的方向。[由传统的主动获取变成了当下的被动查找]

传统模式下 ——> 传统的资源查找方式要求组件(B)向容器(A)发起请求查找资源（抛出set方法），作为回应，容器(A)适时的返回资源(get到的内容)。

ioc ——> 应用了ioc以后，则是容器(spring)主动将资源(bean)推送给它所管理的组件（User user = (User) applicationContext.getBean("user");)的时候，spring就自动进行了User的构造方法和set方法），组件所要做的仅仅是选择一种合适的方式来接受资源（例子中选取get方法去获取资源）。这种行为也别称作查找的被动形式，学术上叫IOC。就是将资源的创建初始化这种原先由调用组件所做工作的控制权转交给spring。

DI （Dependency Injection）：依赖注入

IoC过程中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的。

比如：对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的，依赖注入的名字就这么来的.

DI是 IOC更好的一个诠释，更直接的解释就是DI就是IOC。IOC是宏观看到的，而DI是IOC的运行机制。

IOC的前世今生

1. 分离接口与实现： A=>B{CHtmlStyle(iml),DPdfStyle(iml)}==========>输出自己想要的格式信息 【原始社会】
2. 工厂模式：比原先的降低了耦合度。 出现E去充当工厂的角色，A去查找工厂，同时A里面有B接口的定义，A给E所需类型，E去做转换。new出相应的类返回.从而A去调取完成实现。【封建社会】
3. 控制反转：当下spring使用的.... ，一个巨大的变更 【现代社会】

在spring中可以配置各层的组件（bean），并维护bean与bean的关系。

applicationContext内部就像一个hashMap一样 内存中 id 对象 id名字 Object

id = Class.forName("Object");

测试过程中applicationContext的创建的重量级的，所以创建成单态的。项目中应为都在spring中操作，所以通过依赖注入的方式实现业务逻辑功能。