 引言 在J2EE的整个发展历程中，现在正是一个非常时刻。从很多方面来说，J2EE都是一个伟大的成功：它成功地在从前没有标准的地方建立了标准；大大提升了企业级软件的开放程度，并且得到了整个行业和开发者的广泛认可。然而，J2EE在一些方面已经开始捉襟见肘。J2EE应用开发的成本通常很高。J2EE应用项目至少和从前的非J2EE项目一样容易失败如果不是更容易失败的话。这样的失败率高得让人难以接受。在这样的失败率之下，软件开发几乎变成了碰运气。而在J2EE遭遇失败的场景中，EJB通常都扮演着重要的角色。因此，J2EE社群不断地向着更简单的解决方案、更少使用EJB的方向发展[1]。然而，每个应用程序都需要一些基础设施，拒绝使用EJB并不意味着拒绝EJB所采用的基础设施解决方案。那么，如何利用现有的框架来提供这些基础设施服务呢，伴随着这个问题的提出，一个轻量级的J2EE解决方案出现了，这就是Spring Framework。 Spring是为简化企业级系统开发而诞生的，Spring框架为J2EE应用常见的问题提供了简单、有效的解决方案，使用Spring，你可以用简单的POJO(Plain Old Java Object)来实现那些以前只有EJB才能实现的功能。这样不只是能简化服务器端开发，任何Java系统开发都能从Spring的简单、可测试和松耦合特征中受益。可以简单的说，Spring是一个轻量级的反向控制（IoC）和面向切面编程（AOP）容器框架[3]。Spring IoC，借助于依赖注入设计模式，使得开发者不用理会对象自身的生命周期及其关

系，而且能够改善开发者对J2EE模式的使用；Spring AOP，借助于Spring实现的拦截器，开发者能够实现以声明的方式使用企业级服务，比如安全性服务、事务服务等。Spring IoC和Spring . AOP组合，一起形成了Spring，这样一个有机整体，使得构建轻量级的J2EE架构成为可能，而且事实证明，非常有效。没有Spring IoC的Spring AOP是不完善的，没有Spring AOP的Spring IoC是不健壮的。本文是以Spring架构的成功的实际商务系统项目为背景，阐述了反向控制原理和面向切面的编程技术在Spring框架中的应用，同时抽取适量代码示意具体应用，并和传统开发模式进行对比，展示了Spring framework的简单，高效，可维护等优点。 1、Spring IoC 1.1 反向控制原理 反向控制是Spring框架的核心。但是，反向控制是什么意思？到底控制的什么方面被反向了呢？2004年美国专家Martin Fowler发表了一篇论文《Inversion of Control Containers and the Dependency Injection pattern》阐述了这个问题，他总结说是获得依赖对象的方式反向了，根据这个启示，他还为反向控制提出了一个更贴切的名字：Dependency Injection(DI 依赖注入)。 通常，应用代码需要告知容器或框架,让它们找到自身所需要的类,然后再由应用代码创建待使用的对象实例。因此，应用代码在使用实例之前，需要创建对象实例。然而，IoC模式中,创建对象实例的任务交给IoC容器或框架(实现了IoC设计模式的框架也被称为IoC容器)，使得应用代码只需要直接使用实例，这就是IoC。相对IoC 而言，“ 依赖注入 ”的确更加准确的描述了这种设计理念。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，形象的来说，即由容器动态的将某种依赖关系注入到组件之中。

1.2 IoC在Spring中的实现 任何重要的系统都需要至少两个相互合作的类来完成业务逻辑。通常，每个对象都要自己负责得到它的合作（依赖）对象。你会发现，这样会导致代码耦合度高而且难于测试。使用IoC，对象的依赖都是在对象创建时由负责协调系统中各个对象的外部实体提供的，这样使软件组件松散连接成为可能。下面示意了Spring IoC 应用，步骤如下：（1）定义Action接口，并为其定义一个execute方法，以完成目标逻辑。多年前，GoF在《Design Pattern：Elements of Reusable Object-Oriented Software》一书中提出“ Programming to an Interface，not an implementation ”的原则，这里首先将业务对象抽象成接口，正是为了实施这个原则。（2）类UpperAction实现Action接口，在此类中，定义一个String型的域message，并提供相应的setter和getter方法，实现的execute方法如下： public String execute (String str) { return (getMessage () str).toUpperCase () . }（3）编写Spring配置文件（bean.xml） ＜beans＞＜bean id="TheAction" class="net.chen.spring.qs.UpperAction"＞＜property name="message"＞＜value＞HeLLo＜/value＞＜/property＞＜/bean＞＜/beans＞（4）测试代码 public void testQuickStart () { ApplicationContext ctx=new FileSystemXmlApplicationContext ("bean.xml"). Action a= (Action) ctx.getBean ("TheAction"). System.out.println (a. execute ("Rod Johnson")).}上面的测试代码中，我们根据"bean.xml"创建了一个ApplicationContext实例，并从此实例中获取我们所需的Action实现，运行测试代码，我们看到控制台输出：…… HELLO ROD JOHNSON 仔细观察一下上面的代码，可以看到：（1）我们的组件并不需要

实现框架指定的接口，因此可以轻松的将组件从Spring中脱离，甚至不需要任何修改，这在基于EJB框架实现的应用中是难以想象的。（2）组件间的依赖关系减少，极大改善了代码的可重用性。Spring的依赖注入机制，可以在运行期为组件配置所需资源，而无需在编写组件代码时就加以指定，从而在相当程度上降低了组件之间的耦合。 Spring给我们带来了如此这般的好处，那么，反过来，让我们试想一下，如果不使用Spring框架，回到我们传统的编码模式，情况会是怎样呢？首先，我们必须编写一个配置文件读取类，以实现Message属性的可配置化。 