Spring

1.IOC

spring中的控制反转（IOC）

将对象间的依赖关系交给Spring容器，使用配置文件来创建所依赖的对象，由主动创建对象改为了被动方式，实现解耦合。

- Spring的**依赖注入**将对象之间的依赖关系交给了框架来处理，减小了各个组件之间的耦合性；
- **AOP面向切面编程，可以将通用的任务抽取出来，复用性更高；**
- Spring对于其余**主流框架都提供了很好的支持**，代码的侵入性很低。

2.AOP

AOP，面向切面编程是指当需要在某一个方法之前或者之后做一些额外的操作，比如说日志记录，权限判断，异常统计等，可以利用AOP将功能代码从业务逻辑代码中分离出来

日志记录,事务控制,权限管理

动态代理

1.动态代理的特点

字节码随用随创建，随用随加载

它与静态代理的区别也在于此。因为静态代理是字节码一上来就创建好，并完成加载。

装饰者模式就是静态代理的一种体现

2.基于接口的动态代理

提供者是JDK官方的Proxy类

要求被代理对象至少实现一个接口，将代理接口中的方法

通过*Proxy.newProxyInstance*创建代理对象，三个参数

ClassLoader：和被代理对象使用相同的类加载器。

Interfaces：和被代理对象具有相同的行为。实现相同的接口。

InvocationHandler：如何代理通过重写

public Object invoke(Object proxy, Method method, Object[] args)来实现方法增强

Object proxy 代理对象的引用

Method method 当前执行的对象方法

Object[] args 执行方法所需的参数

3.基于子类的动态代理

提供者是Cglib的Enhancer类

被代理对象不能是最终类,生成一个被代理对象的子类来作为代理

通过*Enhancer.create*创建代理对象，两个参数

create(Class, Callback)

Class：被代理对象的字节码

Callback：如何代理

public Object intercept(Object proxy, Method method, Object[] args,MethodProxy methodProxy)

Spring中的常用注解

@Controller 作用于类,标识为控制层对象

@Service 作用于类,标识为服务层对象

@Repository 作用于类,标识为持久层对象

@Component 作用于类,登记对象到IOC容器

@Autowired 作用于变量,表示依赖注入

BeanFactory和ApplicationContext

ApplicationContext 的三种实现方式

FileSystemXmlApplicationContext

ClassPathXmlApplicationContext

WebXmlApplicationContext

BeanFactory和ApplicationContext的优缺点

BeanFactory采用的是延迟加载形式来注入Bean的

ApplicationContext，它是在容器启动时，一次性创建了所有的Bean。

BeanFactory的优缺点：

优点：应用启动的时候占用资源很少，对资源要求较高的应用，比较有优势；

缺点：运行速度会相对来说慢一些。而且有可能会出现空指针异常的错误，而且通过Bean工厂创建的Bean生命周期会简单一些。

ApplicationContext的优缺点：

优点：所有的Bean在启动的时候都进行了加载，系统运行的速度快；在系统启动的时候，可以发现系统中的配置问题。

缺点：把费时的操作放到系统启动中完成，所有的对象都可以预加载，缺点就是内存占用较大。

Spring中Bean的作用域

singleton ：Bean在每个Spring IOC 容器中只有一个实例，默认作用域。

prototype：一个Bean的定义可以有多个实例。

request：每次http请求都会创建一个Bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。

session：在一个HTTP Session中，一个Bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

global-session：在一个全局的HTTP Session中，一个Bean对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

SpringMVC

1.MVC

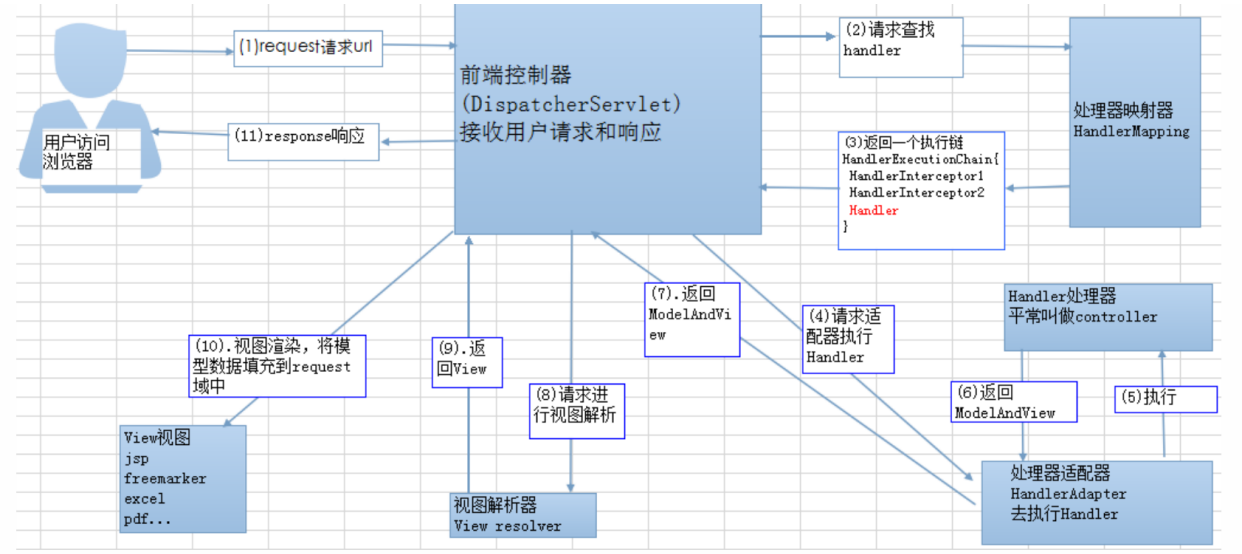
MVC是一个架构模式，它分离了表现与交互。它被分为三个核心部件：模型、视图、控制器。下面是每一个部件的分工：

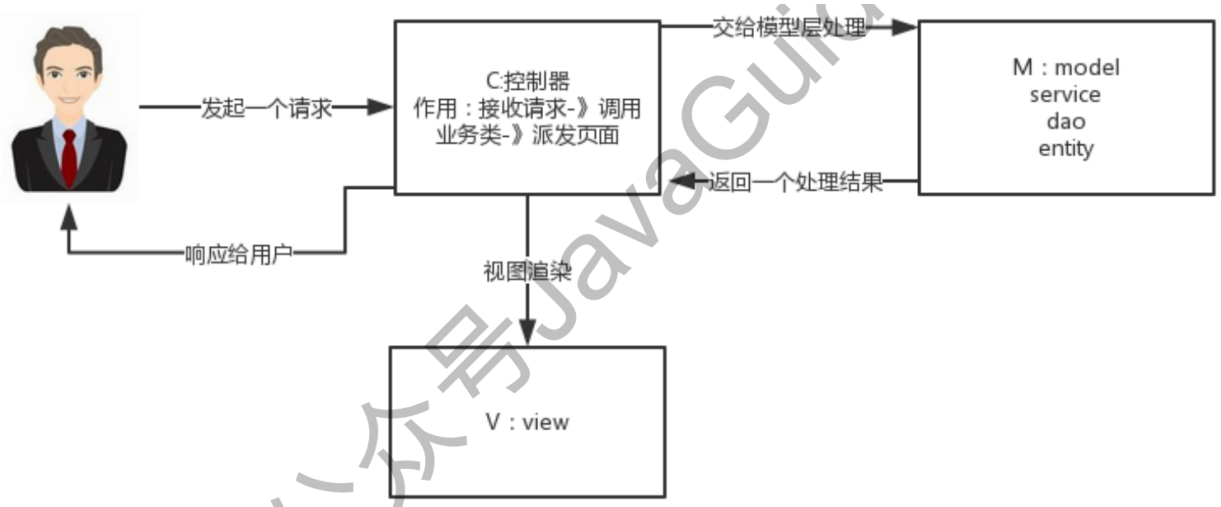
- 视图是用户看到并与之交互的界面。
- 模型表示业务数据，并提供数据给视图。
- 控制器接受用户的输入并调用模型和视图去完成用户的需求。

下面是MVC（模型、视图、控制器）架构的控制流程：

- 所有的终端用户请求被发送到控制器。
- 控制器依赖请求去选择加载哪个模型，并把模型附加到对应的视图。
- 附加了模型数据的最终视图做为响应发送给终端用户。

2.SpringMVC工作原理





流程说明:

1.客户端发送请求,直接请求到DispatcherServlet.

2.DispatcherServlet根据请求信息调用HandlerMapping,解析请求对应的Handler.

3.解析到对应的Handler(Controller)控制器,毫影有HandlerAdapter适配器处理

4.HandlerAdapter根据Handler来调用真正的处理器处理相应的业务逻辑

5.处理完返回一个ModelView对象,Model是返回的数据对象,View是逻辑上的View

6.ViewResolver根据逻辑view查找实际view

7.DispatcherServlet把返回的model传给view

8.把view返回给请求者