Spring

Spring 框架

Spring 框架指的都是 Spring Framework，它是很多模块的集合，使用这些模块可以很方便地协助我们进行开发。这些模块是：核心容器、数据访问/集成,、Web、AOP（面向切面编程）、工具、消息和测试模块。

Spring主要模块

- **Spring Core：** 基础,可以说 Spring 其他所有的功能都需要依赖于该类库。主要提供 IoC 依赖注入功能。
- **Spring Aspects** ： 该模块为与AspectJ的集成提供支持。
- **Spring AOP** ： 提供了面向切面的编程实现。
- **Spring JDBC** : Java数据库连接。
- **Spring JMS** ： Java消息服务。
- **Spring ORM** : 用于支持Hibernate等ORM工具。
- **Spring Web** : 为创建Web应用程序提供支持。
- **Spring Test** : 提供了对 JUnit 和 TestNG 测试的支持。
- **Spring WebFlux** : 5.x版本引进异步响应式处理组件

Spring MVC

Spring MVC是一种设计模式

Spring MVC 可以帮助我们进行更简洁的Web层的开发，并且它天生与 Spring 框架集成。Spring MVC 下我们一般把后端项目分为 Service层（处理业务）、Dao层（数据库操作）、Entity层（实体类）、Controller层(控制层，返回数据给前台页面)

Spring MVC原理

Spring MVC 的入口函数也就是前端控制器 `DispatcherServlet` 的作用是接收请求，响应结果。

流程说明（重要）：

1. 客户端（浏览器）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`（也就是我们平常说的 `Controller` 控制器）后，开始由 `HandlerAdapter` 适配器处理。
4. `HandlerAdapter` 会根据 `Handler` 来调用真正的处理器开处理请求，并处理相应的业务逻辑。
5. 处理器处理完业务后，会返回一个 `ModelAndView` 对象，`Model` 是返回的数据对象，`View` 是个逻辑上的 `View`。
6. `ViewResolver` 会根据逻辑 `View` 查找实际的 `View`。
7. `DispaterServlet` 把返回的 `Model` 传给 `View`（视图渲染）。
8. 把 `View` 返回给请求者（浏览器）

Spring IOC & AOP

IOC

Ioc（Inverse of Control:控制反转）是一种**设计思想**，就是 **将原本在程序中手动创建对象的控制权，交由Spring框架来管理。** IoC 在其他语言中也有应用，并非 Spirng 特有。 **IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个Map（key，value）,Map 中存放的是各种对象。**

将对象之间的相互依赖关系交给 IoC 容器来管理，并由 IoC 容器完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。

Spring Ioc初始化过程

xml -> Resource -> BeanDefinition -> BeanFactory

AOP

AOP(Aspect-Oriented Programming:面向切面编程)，**业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来**，便于**减少系统的重复代码**，**降低模块间的耦合度**，并**有利于未来的可拓展性和可维护性。**

> Spring AOP指什么，JDK proxy实现动态代理，cglib实现有啥区别，Aspectj又是啥

Spring AOP就是基于动态代理的，使用JDK Proxy(基于接口)，Cglib(基于类)，也可以使用AspectJ，Spring AOP 已经集成了AspectJ，AspectJ是 Java 生态系统中最完整的 AOP 框架。

Spring AOP 属于运行时增强，而 AspectJ 是编译时增强。 Spring AOP 基于代理(Proxying)，而 AspectJ 基于字节码操作(Bytecode Manipulation)

基础概念

> AOP
>
> AOP(Aspect Orient Programming) 作为面向对象的一种补充，广泛用于处理具有横切性质的系统级服务，如 事务，安全检查，缓存，对象池管理等。
>
> AOP 实现的关键就在于 AOP 框架自动创建代理对象，AOP 代理可分为 静态代理 和 动态代理 两大类，静态代理在编译阶段就可以生成代理类，因此也称为 编译时增强；动态代理 在运行时 借助 JDK 动态代理或者 CGLIB 等在内存中临时生成代理类，因此也被称为运行时增强
>
> Spring AOP:
>
> Spring 提供的 AOP 框架，底层使用 JDK 动态代理或者 CGLIB 来实现，支持了 Aspectj 的注解标签
>
> CGLIB:
>
> CGLIB 是一个功能强大，高性能的代码生成包，它可以为没有实现接口的类提供代理，对 JDK 基于接口的动态代理做了很好的补充。
>
> AspectJ:
>
> AspectJ 是一个 AOP 框架，是事实上的 AOP 标准。
>
> 一句话概括:
>
> Spring AOP 最终使用的是 JDK 或者 CGLIB 来实现的, 只是 Spring AOP 支持了 Aspectj 的注解标签
> https://blog.csdn.net/itguangit/article/details/108420366

Spring bean

bean作用域

- singleton : 唯一 bean 实例，Spring 中的 bean 默认都是单例的。
- prototype : 每次请求都会创建一个新的 bean 实例。
- request : 每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。
- session : 每一次HTTP请求都会产生一个新的 bean，该bean仅在当前 HTTP session 内有效。
- global-session： 全局session作用域，仅仅在基于portlet的web应用中才有意义，Spring5已经没有了。Portlet是能够生成语义代码(例如：HTML)片段的小型Java Web插件。它们基于portlet容器，可以像servlet一样处理HTTP请求。但是，与 servlet 不同，每个 portlet 都有不同的会话

bean生命周期

- Bean 容器找到配置文件中 Spring Bean 的定义。
- Bean 容器利用 Java Reflection API **创建一个Bean的实例。**
- 如果涉及到一些属性值 利用 `set()` 方法**设置属性值。**
- **检查Aware相关接口，并设置相关依赖**
  - 如果 Bean 实现了 `BeanNameAware` 接口，调用 `setBeanName()` 方法，传入Bean的名字。
  - 如果 Bean 实现了 `BeanClassLoaderAware` 接口，调用 `setBeanClassLoader()` 方法，传入 `ClassLoader` 对象的实例。
  - 如果Bean实现了 `BeanFactoryAware` 接口，调用 `setBeanClassLoader()` 方法，传入 `ClassLoade` r对象的实例。
  - 与上面的类似，如果实现了其他 `*.Aware` 接口，就调用相应的方法。
- 如果有和加载这个 Bean 的 Spring 容器相关的 `BeanPostProcessor` 对象，执行**前置处理** `postProcessBeforeInitialization()` 方法
- 如果Bean实现了 `InitializingBean` 接口，执行 `afterPropertiesSet()` 方法。
- 如果 Bean 在配置文件中的定义包含 init-method 属性，执行指定的方法。
- 如果有和加载这个 Bean的 Spring 容器相关的 `BeanPostProcessor` 对象，执行**后置处理** `postProcessAfterInitialization()` 方法
- 当要**销毁** Bean 的时候，如果 Bean 实现了 `DisposableBean` 接口，执行 `destroy()` 方法。
- 当要销毁 Bean 的时候，如果 Bean 在配置文件中的定义包含 destroy-method 属性，执行指定的方法。

注解

@Controller vs @RestController

单独使用 `@Controller` 不加 `@ResponseBody` 的话一般使用在要返回一个视图的情况，这种情况属于比较传统的Spring MVC 的应用，对应于前后端不分离的情况。

`@RestController` (Spring 4.x新加的注解)只返回对象，对象数据直接以 JSON 或 XML 形式写入 HTTP 响应(Response)中，这种情况属于 RESTful Web服务，这也是目前日常开发所接触的最常用的情况（前后端分离）

@Component vs @Bean

1. 作用对象不同: `@Component` 注解作用于类，而 `@Bean` 注解作用于方法。
2. `@Component` 通常是通过类路径扫描来自动侦测以及自动装配到Spring容器中（我们可以使用 `@ComponentScan` 注解定义要扫描的路径从中找出标识了需要装配的类自动装配到 Spring 的 bean 容器中）。 `@Bean` 注解通常是我们在标有该注解的方法中定义产生这个 bean, `@Bean` 告诉了Spring这是某个类的示例，当我需要用它的时候还给我。

3. `@Bean` 注解比 `Component` 注解的自定义性更强，而且很多地方我们只能通过 `@Bean` 注解来注册bean。比如当我们引用第三方库中的类需要装配到 `Spring`容器时，则只能通过 `@Bean`来实现。

@Resource vs @Autowired

1. 两者都可以写在字段和setter方法上。两者如果都写在字段上，那么就不需要再写setter方法
2. @Autowired是Spring提供的注解，默认按照byType注入；如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。
3. @Resource是J2EE提供，默认按照ByName自动注入；如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略

@Transient

使用JPA在数据库中非持久化一个字段，也就是不保存到数据库中，推荐使用这个注解，也可以用关键字transient

@Transactional(rollbackFor = Exception.class)

当`@Transactional`注解作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。如果类或者方法加了这个注解，那么这个类里面的方法抛出异常，就会回滚，数据库里面的数据也会回滚。

> Exception分为运行时异常RuntimeException和非运行时异常。

在`@Transactional`注解中如果不配置`rollbackFor`属性,那么事物只会在遇到`RuntimeException`的时候才会回滚,加上`rollbackFor=Exception.class`,可以让事物在遇到非运行时异常时也回滚。

Spring框架中使用的设计模式

- **工厂设计模式** : Spring使用工厂模式通过 `BeanFactory`、`ApplicationContext` 创建 bean 对象。
- **代理设计模式** : Spring AOP 功能的实现。
- **单例设计模式** : Spring 中的 Bean 默认都是单例的。
- **模板方法模式** : Spring 中 `jdbcTemplate`、`hibernateTemplate` 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。
- **包装器设计模式** : 我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
- **观察者模式:** Spring 事件驱动模型就是观察者模式很经典的一个应用。
- **适配器模式** :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配`Controller`。

设计模式，策略设计模式

根据入参，选择对应的实现类，抽象接口Strategy，具体策略实现类StrategyImpl多个

相比动态代理来说，把策略选择的逻辑是封装了一层StrategyContext，具体的实现作为成员属性传递

https://www.cnblogs.com/yefengyu/p/10527350.html

Spring事务

1. 编程式事务，在代码中硬编码。(不推荐使用)
2. 声明式事务，在配置文件中配置（推荐使用）

1. 基于XML（常用）
2. 基于注解（常用）

Spring事务中隔离级别（一般采用默认配置）

- **TransactionDefinition.ISOLATION_DEFAULT:** 使用后端数据库默认的隔离级别，Mysql 默认采用的 REPEATABLE_READ隔离级别 Oracle 默认采用的 READ_COMMITTED隔离级别.
- **TransactionDefinition.ISOLATION_READ_UNCOMMITTED:** 最低的隔离级别，允许读取尚未提交的数据变更，**可能会导致脏读、幻读或不可重复读**
- **TransactionDefinition.ISOLATION_READ_COMMITTED:** 允许读取并发事务已经提交的数据，**可以阻止脏读，但是幻读或不可重复读仍有可能发生**
- **TransactionDefinition.ISOLATION_REPEATABLE_READ:** 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，**可以阻止脏读和不可重复读，但幻读仍有可能发生。**
- **TransactionDefinition.ISOLATION_SERIALIZABLE:** 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，**该级别可以防止脏读、不可重复读以及幻读**。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

Spring 事务传播行为

支持当前事务的情况：

- **TransactionDefinition.PROPAGATION_REQUIRED：** 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。
- **TransactionDefinition.PROPAGATION_SUPPORTS：** 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。
- **TransactionDefinition.PROPAGATION_MANDATORY：** 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。（mandatory：强制性）

不支持当前事务的情况：

- **TransactionDefinition.PROPAGATION_REQUIRES_NEW：** 创建一个新的事务，如果当前存在事务，则把当前事务挂起。
- **TransactionDefinition.PROPAGATION_NOT_SUPPORTED：** 以非事务方式运行，如果当前存在事务，则把当前事务挂起。
- **TransactionDefinition.PROPAGATION_NEVER：** 以非事务方式运行，如果当前存在事务，则抛出异常。

其他情况：

- **TransactionDefinition.PROPAGATION_NESTED：** 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于 TransactionDefinition.PROPAGATION_REQUIRED。

其他

- 单例 bean 存在**线程问题**，主要是因为当多个线程操作同一个对象的时候，对这个对象的非静态成员变量的写操作会存在线程安全问题。
    - 在类中定义一个ThreadLocal成员变量，将需要的可变成员变量保存在 ThreadLocal 中（推荐的一种方式）