Spring

Springboot与Spring的关系

Springboot简化了Spring的配置

Springboot内嵌Tomcat

Spring Boot 的主要优点

- 简化了Spring的配置
- 内嵌Tomcat，可以打成jar包独立运行

Spring Boot 支持哪些内嵌 Servlet 容器

- tomcat
- jetty

如何在 Spring Boot 应用程序中使用 Jetty 而不是 Tomcat

1. 排除掉tomcat的依赖
2. 添加jetty的依赖

Spring怎么解决循环依赖的

Bean

Bean的注入方式@Autowire或@Resource的区别

@Autowired是Spring的注解，注解是按类型装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它required属性为false

@Resource是Java的注解，默认按照名字进行注入。有两个属性name和type，表示使用名字或类型进行注入

Spring 中 beanFactory 和 ApplicationContext 的联系和区别

两者都可作为容器

BeanFactory：

是Spring里面最低层的接口，提供了最简单的容器的功能，只提供了实例化对象和拿对象的功能；

ApplicationContext：

应用上下文，继承BeanFactory接口，它是Spring的一各更高级的容器，提供了更多的有用的功能

总体区别如下：

1） 使用 ApplicationContext，配置 bean 默认配置是 singleton，无论是否使用，都会被实例化。优点是预先加载，缺点是浪费内存；

2） 使用 BeanFactory 实例化对象时，配置的 bean 等到使用的时候才会被实例化。优点是节约内存，缺点是速度比较慢，多用于移动设备的开发；

3） 没有特殊要求的情况下，应该使用 ApplicationContext 完成，ApplicationContext 可以实现

BeanFactory 所有可实现的功能，还具备其他更多的功能。

bean 的作用域

作用域	描述
singleton	bean 默认都是单例的
prototype	IOC容器可以创建多个Bean实例，每次返回的都是一个新的实例
request	每个请求创建一个
session	每个会话创建一个
global-session	所有的Session共享一个Bean实例，Spring5已经没有了

Spring bean 的生命周期

1. 如果Bean实现了BeanPostProcessor接口，实例化之前Spring就将调用他们的postProcessBeforeInitialization()方法。
2. 如果Bean 实现了InitializingBean接口，初始化时Spring将调用他们的afterPropertiesSet()方法。类似的，如果bean使用init-method声明了初始化方法，该方法也会被调用
3. 如果Bean 实现了BeanPostProcessor接口，Spring就将调用他们的postProcessAfterInitialization()方法。
4. bean实现了DisposableBean接口，Spring将调用它的destory()接口

Spring中的Bean为什么用单例

- 减少实例的创建提高性能
- 减少GC
- 快速获得

SpringBean是线程安全的吗

单例的，但不是线程安全的

BeanFactory 和FactoryBean的区别

@Component 和 @Bean 的区别是什么?

1. 作用范围不同

IOC

什么是控制反转

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。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。

什么是IOC容器

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

AOP

什么是面向切面编程

面向对象编程的补充，不再使用继承的方式来增强而是使用切面的方式进行增强

名词解释

切面：增强的功能类

连接点：

切入点：

通知：增强的方法

AOP 有哪些实现方式

- 预编译：AspectJ
- 运行期动态代理（JDK动态代理、CGLib动态代理）：SpringAOP

Spring AOP 和 AspectJ AOP 有什么区别?

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)。

springIOC、AOP、静态代理和动态代理的区别

静态代理时在编译的时候生成一个代理类，动态代理是在运行的时候生成一个代理对象

项目中SpringAOP的使用

日志采集

有哪几种增强

- 运行前
- 运行后
- 返回
- 环绕
- 异常

```
try{
      try{
           doBefore();//对应@Before注解的方法切面逻辑
           method.invoke();
       }finally{
           doAfter();//对应@After注解的方法切面逻辑
       }
       doAfterReturning();//对应@AfterReturning注解的方法切面逻辑
   }catch(Exception e){
        doAfterThrowing();//对应@AfterThrowing注解的方法切面逻辑
   }
```

拦截器和过滤器底层原理

过滤器 和 拦截器 均体现了AOP的编程思想，都可以实现诸如日志记录、登录鉴权等功能

实现原理：过滤器是基于Servlet的函数回调的，依赖tomcat容器，拦截器 则是基于Java的反射机制（动态代理）实现的。

触发时机：Filter是进入Tomcat后servlet前触发，拦截器是进入servlet后，controller前触发

```
@Order(1)
@WebFilter(filterName="firstFilter", urlPatterns="/*")
public class FirstFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {
        System.out.println("first filter 1");
        chain.doFilter(request, response);
        System.out.println("first filter 2");
```

```
    }

    @Override
    public void destroy() {

    }
}
```

事务

Spring的事务传播机制

Service接口方法可能会在内部调用其它的Service接口方法以共同完成一个完整的业务操作，因此就会产生服务接口方法嵌套调用的情况， Spring通过事务传播行为控制当前的事务如何传播到被嵌套调用的目标服务接口方法中。即当前的事务(调用的方法)遇到了另一个事务(方法调用了另一个方法)，另一个事务怎么办。

分为支持当前事务和不支持当前事务

传播行为	描述
REQUIRED	默认的，如果当前没有事务，就新建一个事务，如果有，就加入当前事务
PROPAGATION_SUPPORTS	以当前事务为准，当前没有事务，就以非事务的方式运行
PROPAGATION_MANDATORY	以当前事务为准，当前没有事务，就抛出异常
PROPAGATION_REQUIRES_NEW	新建事务，把当前事务挂起
PROPAGATION_NOT_SUPPORTED	不要事务，把当前事务挂起
PROPAGATION_NEVER	非事务方式运行，当前存在事务时抛出异常
NESTED	与REQUIRED类似，只不过它是嵌套的事务，嵌套事务出错是不会全部回滚

@Transactional

加上`rollbackFor=Exception.class`,可以让事物在遇到非运行时异常时也回滚

事务失效的原因

1. @Transactional没有放在public方法上，Spring Framework 默认使用 AOP 代理，在代码运行时生成一个代理对象来执行事务，而直接通过service调用方法，执行的不是代理对象，所以索引失效了。
2. **在同一个类之中，方法互相调用，切面无效**

总结一句话，**调用了自身而没有经过 Spring 的代理类**

解决方案：事务代码用一个类单独去处理

2. 异常被吃掉了

SpringMVC

SpringMVC工作流程

1. 发送请求到**DispatchServlet**，DispatchServlet调用**HandlerMapping**通过url找到对应的处理器及拦截器生成执行链返回给DispatchServlet
2. DispatchServlet调用**HandlerAdapter**，HandlerAdapter进行参数校验，格式转换等后执行Handler，返回ModelAndView
3. DispatchServlet将ModelAndView调用**View Resoler**，
4. DispatchServlet将结果返回用户

SpringBoot

SpringBoot的核心注解，主要有哪几个注解组成，Spring Boot 的自动配置是如何实现的? **

@SpringBootApplication，主要由@SpringBootConfiguration，@EnableAutoConfiguration，@ComponentScan三个注解组成

@SpringBootConfiguration：等同于@Configuration,说明它是一个配置类，可以用来注入bean

@ComponentScan：自动扫描本包和子包路径下的 @Component(以及@Service等) 注解进行注册 bean 实例到 context 中

@EnableAutoConfiguration：通过判断类路径下有没有相应配置的jar包来确定是否加载和配置这个功能，读取jar包下的配置文件，将需要的bean注入到spring容器中。

bootstrap.yml与application.yml的区别

bootstrap.yml用来加载远程配置文件，优先于application，一般在SpringCloudConfig或者Nacos中用到。

bootstrap主要用于从额外的资源来加载配置信息，

启动SpringBoot后运行一些代码

- ApplicationRunner接口
- CommandLineRunner接口

Spring Boot 常用的读取配置文件的方法有哪些

配置文件的优先级

config>当前目录，工作目录>classPath

1. 工作目录/config
2. 工作目录
3. classpath/config
4. classpath

Spring Boot 如何做请求参数校验

实体类属性上添加@xxx注解和错误消息

接口中的参数前添加@Valid，参数后进阶BindResult接收错误消息

判断BindResult中有没有错误

如何使用 Spring Boot 实现全局异常处理?

类名上添加 @ControllerAdvice

定义方法，方法上添加 @ExceptionHandler(value =Exception.class)，方法的参数接受异常

如何在服务端使用 Cookie 呢?

- @CookieValue(value="key")
- 从HttpRequest中读取

如何读取Header

@RequestHeader("key")

其他

Spring 框架中都用到了哪些设计模式

- 工厂模式
- 代理模式
- 单例模式
-

怎么防止定时任务重复值行多次

Oauth2协议授权流程简单介绍

Oauth2中后台token是存在服务JVM内存中，如果服务崩了的话，token失效了怎么处理(可以用Redis去实现持久化)

Spring线程池

怎么进行限流

SpringMVC不同用户登录的信息怎么保证线程安全的

- 不要定义类变量
- bean设为原型模式

Spring怎么解决循环依赖的