1. Spring

1.1 请谈一下你对Spring的理解？说一下Spring的核心是什么？请谈一下你对Spring IOC和AOP的理解？

Spring框架是一个轻量级的JavaSE/JavaEE应用开发框架，是构建企业级应用程序的一站式解决方案。

Spring是模块化的，并被分为大约20个模块(core、beans、context、web等)，允许我们只使用需要的部分，而不需要引入其他部分。

Spring的两大核心内容是IOC和AOP(控制翻转和面向切面编程)；

谈一下对IOC（Inversion Of Control）的理解

IOC的意思是控制反转，它是一种设计思想，是一个重要的面向对象编程的法则；

在Java开发中，Ioc可以让我们把设计好的对象交给容器控制，而不是在对象内部直接控制；

对于Spring框架来说，就是由Spring来负责控制对象的生命周期和对象间的关系；

谈一下对AOP的理解

AOP被称为面向切面编程，是一种编程范式，是对面向对象编程(OOP)的一种完善。

OOP最大问题就是无法解耦组件进行开发，而AOP就是为了克服这个问题而出现的。

AOP将整个系统分为“核心业务逻辑”和“非核心的服务”;

AOP的关注点是系统中的“非核心服务”【权限;事务;安全;异常;日志等】;

Spring将非核心服务封装成一个AOP组件，然后通过配置信息形成“核心业务和AOP组件”之间的调用关系，当执行核心业务时，AOP组件会在合适的时机进行调用；

附：理解IOC或者DI

- 谁控制谁？控制什么？

  传统Java程序设计，我们直接在对象内部通过new进行创建对象，是程序主动去创建依赖对象；

  而IoC有专门的容器来创建这些对象，即由Ioc容器来控制对象的创建；
- 为什么是反转？

  由于是容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反

转；

- 哪些方面反转了？

  依赖对象的获取被反转了。

- 谁依赖于谁？ 应用程序依赖于IoC容器；
- 为什么需要依赖？ 应用程序需要IoC容器来提供对象；
- 谁注入谁？ IoC容器注入应用程序中的某个对象（应用程序依赖的对象）；
- 注入了什么？ 注入了某个对象所需要的外部资源（包括对象、资源等）。

1.2 请说一下Spring的Bean作用域

Spring提供"singleton"和"prototype"两种基本作用域；

另外Spring提供"request"、"session"、"global session"三种web作用域；

如果不指定Bean的作用域，Spring默认使用singleton作用域；

- singleton：在Spring IOC容器中仅存在一个Bean的实例，Bean以单例的方式存在；
- prototype：每次从容器中调用Bean时，都返回一个新的实例，也就是每次调用getBean()方法时，相当于执行了new对象的操作；
- request：每次http请求都会创建一个新的Bean，该作用域仅适合WebApplicationContext环境；
- session：同一个http session共享一个Bean实例，不同session使用不同的Bean实例，该作用域仅适用WebApplicationContext环境；
- global session：这种作用域类似于标准的HTTP Session作用域，不过仅仅在基于portlet的web应用中才有意义；

附：简单理解Portlet

Portlet规范定义了全局Session的概念，它被所有构成某个portlet web应用的各种不同的portlet所共享；

在global session作用域中定义的bean被限定于全局portlet Session的生命周期范围内；

1.3 请谈一下Spring中bean对象的生命周期

Spring Bean的生命周期主要分为四个阶段，也就是：Bean的实例化、Bean属性赋值、初始化和Bean的销毁；

其中前三个阶段主要实现在AbstractAutowireCapableBeanFactory类中doCreateBean()方法中；

而"Bean的销毁"则是容器关闭时；

1. Spring启动，查找并加载需要被Spring管理的bean，进行Bean的实例化
2. Bean实例化后对将Bean的引入和值注入到Bean的属性中
3. 如果Bean实现了BeanNameAware接口的话，Spring将Bean的Id传递给setBeanName()方法
4. 如果Bean实现了BeanFactoryAware接口的话，Spring将调用setBeanFactory()方法，将BeanFactory容器实例传入
5. 如果Bean实现了ApplicationContextAware接口的话，Spring将调用Bean的setApplicationContext()方法，将bean所在应用上下文引用传入进来。
6. 如果Bean实现了BeanPostProcessor接口，Spring就将调用他们的postProcessBeforeInitialization()方法。
7. 如果Bean 实现了InitializingBean接口，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
8. 如果Bean 实现了BeanPostProcessor接口，Spring就将调用他们的postProcessAfterInitialization()方法。
9. 此时，Bean已经准备就绪，可以被应用程序使用了。他们将一直驻留在应用上下文中，直到应用上下文被销毁。
10. 如果bean实现了DisposableBean接口，Spring将调用它的destory()接口方法，同样，如果bean使用了destory-method 声明销毁方法，该方法也会被调用。

附：理解Spring bean生命周期

在Bean实例化前后分别执行InstantiationAwareBeanPostProcessor类的postProcessBeforeInstantiation()方法和postProcessAfterInstantiation()方法；

其中postProcessBeforeInstantiation()方法的返回值作为原来Bean的代理；

而postProcessAfterInstantiation()方法作为"Bean属性赋值阶段"前的检查条件，在属性赋值前执行，可以影响是否进行"属性赋值"；

如果检查通过，然后进行"Bean属性赋值"；

"属性赋值"完成之后，"Bean初始化之前"，执行两组Aware接口；

- 第一组：BeanNameAware、BeanClassLoaderAware和BeanFactoryAware
- 第二组：EnvironmentAware、EmbeddedValueResolverAware、ResourceLoaderAware、ApplicationEventPublisherAware MessageSourceAware、ApplicationContextAware

其中第一组Aware接口的执行是在invokeAwareMethods()方法中直接执行，这个方法有三个判断：

- 如果bean是BeanNameAware接口的实现类会调用setBeanName方法；
- 如果bean是BeanClassLoaderAware接口的实现类会调用setBeanClassLoader方法，将BeanFactory容器实例传入；
- 如果是BeanFactoryAware接口的实现类会调用setBeanFactory方法，将bean所在应用上下文引用传入进来；

而第二组Aware接口的执行是在BeanPostProcessor类的postProcessBeforeInitialization()方法依次按顺序判断进行调用的，
完成Aware接口的执行之后，执行InitializingBean类的afterPropertiesSet()方法；
进而执行BeanPostProcessor类的postProcessAfterInitialization()方法，完成"Bean的初始化"；

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean接口，会调用其实现的destroy方法。
最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其自定义的销毁方法。

AbstractAutowireCapableBeanFactory类中doCreateBean()方法中的调用顺序：

createBeanInstance() -> 实例化
populateBean() -> 属性赋值
initializeBean() -> 初始化

1.4 Spring的事务相关面试题

1.4.1 请谈一下Spring的两种事务形式？

1. Spring提供了 "编程式事务" 和 "基于AOP方式的声明式事务"；
2. Spring编程式事务管理高层的抽象 主要包括三个接口

- PlatformTransactionManager：事务管理器；
- TransactionDefinition：事务定义信息（包括事务的隔离、传播机制等）；
- TransactionStatus：事务具体运行状态；
  其中Spring为不同的持久化框架提供了不同事务管理器PlatformTransactionManager的接口实现；
  比如
  - 使用Spring JDBC或Mybatis进行持久化数据时的DataSourceTransactionManager；
  - 使用Hibernate进行持久化数据时的HibernateTransactionManager；

- 使用JPA进行持久化数据时的JpaTransactionManager；

  同时，Spring可以使用TransactionTemplate进行编程式的事务控制；

3. Spring基于AOP的声明式事务又有三种方式，分别是：
   - 基于TransactionProxyFactoryBean的方式；（不常用）
   - 基于AspectJ的方式；

- 基于注解方式；

3.1 第一种方式需要为每个事务管理的类配置一个TransactionProxyFactoryBean进行管理，使用时还需要在类中注入该代理类。

3.2 第二种方式直接在配置文件配置好之后，可以按照方法的名字进行管理，无需在类中添加任何东西；

3.3 第三种方式比较简单，配置好事务管理器之后，开启注解事务，然后在业务层类上添加注解@Transactional即可；

4. **附：三种声明式事务配置方式**
   - 基于TransactionProxyFactoryBean的方式 （不常用）

```
 <bean id="baseTransactionProxy"
   class="org.Springframework.transaction.interceptor.Transacti
onProxyFactoryBean"
   abstract="true">
   <property name="transactionManager"
ref="transactionManager"/>
   <property name="transactionAttributes">
     <props>
       <prop key="insert*">PROPAGATION_REQUIRED</prop>
       <prop key="update*">PROPAGATION_REQUIRED</prop>
       <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
     </props>
   </property>
 </bean>
 <bean id="myProxy" parent="baseTransactionProxy">
   <property name="target" ref="myTarget"/>
 </bean>

 <bean id="yourProxy" parent="baseTransactionProxy">
   <property name="target" ref="yourTarget"/>
 </bean>
```

- **基于AspectJ的方式（常用）**

```
<bean id="txManager" class="DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" />
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="save*" propagation="REQUIRED" />
    <tx:method name="add*" propagation="REQUIRED" />
    <tx:method name="insert*" propagation="REQUIRED" />
    <tx:method name="update*" propagation="REQUIRED" />
    <tx:method name="delete*" propagation="REQUIRED" />
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="txPointcut" expression="execution(*
com.etoak..service.*.*(..))" />
  <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"
/>
</aop:config>
```

- **基于注解方式**

```
<bean id="tx"
class="org.Springframework.jdbc.datasource.DataSourceTransactio
nManager">
  <property name="dataSource" ref="dataSource" />
</bean>

<tx:annotation-driven transaction-manager="tx" />
```

1.4.2 对Spring事务的传播行为了解吗？Spring提供了几种事务的传播行为？

事务传播行为主要在多个事务方法间调用时，事务是如何在这些方法间传播的；

Spring共支持7种传播行为：

1. REQUIRED：表示如果当前存在一个事务，则加入该事务，否则将新建一个事务；
2. REQUIRES_NEW：表示不管是否存在事务，都创建一个新的事务，原来的挂起，新的执行完毕，继续执行老的事务；
3. SUPPORTS：表示如果当前存在事务，就加入该事务；如果当前没有事务，那就不使用事务；
4. NOT_SUPPORTED： 表示不使用事务；如果当前存在事务，就把当前事务暂停，以非事务方式执行；
5. MANDATORY：表示必须在一个已有的事务中执行，如果当前没有事务，则抛出异常；
6. NEVER：表示以非事务方式执行，如果当前存在事务，则抛出异常；
7. NESTED：这个是嵌套事务；
   如果当前存在事务，则在嵌套事务内执行；
   如果当前不存在事务，则创建一个新的事务；
   嵌套事务使用数据库中的保存点来实现，即嵌套事务回滚不影响外部事务，但外部事务回滚将导致嵌套事务回滚；

1.4.3 请谈一下Spring事务的隔离机制？

隔离级别是指若干个并发的事务之间的隔离程度。TransactionDefinition接口中定义了五个表示隔离级别的常量：

1. ISOLATION_DEFAULT：默认的
   这是默认值，表示使用底层数据库的默认隔离级别。对大部分数据库而言，通常这值就是ISOLATION_READ_COMMITTED。
2. ISOLATION_READ_UNCOMMITTED：未提交读
   该隔离级别表示一个事务可以读取另一个事务修改但还没有提交的数据。
   该级别不能防止脏读、不可重复读和幻读，因此很少使用该隔离级别。
3. ISOLATION_READ_COMMITTED：已提交读
   该隔离级别表示一个事务只能读取另一个事务已经提交的数据。
   该级别可以防止脏读，这也是大多数情况下的推荐值。
4. ISOLATION_REPEATABLE_READ：可重复读
   该隔离级别表示一个事务在整个过程中可以多次重复执行某个查询，并且每次返回的记录都相同。
   该级别可以防止脏读、不可重复读。
5. ISOLATION_SERIALIZABLE：序列化
   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。

1.4.4 请谈一下Spring事务回滚机制

默认情况下，Spring只有在抛出的异常是运行时异常("非检查型")时才回滚该事务；

也就是抛出的异常为RuntimeException的子类(Errors也会导致事务回滚)；

而抛出非运行时异常(检查型)则不会导致事务回滚；

但是，我们可以明确的配置抛出哪些异常时回滚事务，包括checked异常。也可以定义哪些异常抛出时不回滚事务。

1.5 说一下过滤器和Spring拦截器的区别?

拦截器和过滤器都是AOP编程思想的体现，都能实现权限检查、日志记录等；

他们的区别主要有：

- 拦截器是基于反射实现，更准确的说是通过jdk的动态代理实现； 过滤器是基于函数回调；
- 拦截器不依赖于Servlet容器； 过滤器依赖于Servlet容器，它属于Servlet规范规定的；
- 拦截器只能对Controller请求起作用； 过滤器则可以对几乎所有的请求起作用；
- 拦截器可以访问controller上下文的对象(如service对象、数据源等)；过滤器则不可以访问；
- 拦截器可以深入的方法前后、异常抛出前后等，并且可以重复调用； 过滤器只在Servlet前后起作用，并且只在初始化时被调用一次；

1.6 拦截器的实现原理是什么？简单说一下拦截器用场景？

Java中的拦截器是基于Java反射机制和动态代理实现的；

它依赖于具体的接口，在运行期间动态生成字节码。

2. SpringMVC

2.1 介绍一下SpringMVC流程、执行流程、运行流程

2.2 谈一下你对SpringMVC的理解

针对2.1 和 2.2 的问题

- 首先，回答什么是SpringMVC(官网介绍)
- 然后，SpringMVC的优点
- 然后，SpringMVC的几个重要组件
- 最后，回答SpringMVC执行流程

1. SpringMVC是构建在Servlet api基础之上的MVC框架，它是Spring框架的一个模块。
2. SpringMVC是实现了请求-驱动类型的轻量级web框架；
    - SpringMVC天然能够和Spring进行整合，比如Spring的ioc和aop；
    - 支持与其它视图技术集成，如Freemarker、Thymeleaf等；
    - 更加简单的异常处理；
    - 提供了对静态资源的支持；
    - 支持Restful风格请求；
    - ....

3. SpringMVC主要的组件

    前端控制器、处理器映射器、处理器适配器、视图解析器、处理器、拦截器等

4. SpringMVC执行流程
    1. 用户发送请求到前端控制DispatcherServlet，但是前端控制器并不处理请求，而是将请求转发给处理器映射器；
    2. 处理器映射器使用请求的url(/user.do)去查找(xml、注解)处理器，查找到处理器之后，将HandlerExecutionChain返回DispatcherServlet；
    3. DispatcherServlet请求处理器适配器执行处理器；
    4. 处理器适配器执行处理器；
    5. 处理器执行完成之后，将逻辑视图ModelAndView返回给处理器适配器；
    6. 处理器适配器将ModelAndView返回给DispatcherServlet；
    7. DispatcherServlet请求视图解析器解析逻辑视图ModelAndView，解析完成之后，返回View对象给DispatcherServlet
    8. DispatcherServlet根据View和模型数据进行视图渲染，响应用户请求；

2.3 说一下Springmvc的核心类、说一下SpringMVC的核心？

SpringMVC提供了很多组件

1. DispatcherServlet：前端控制器，作为统一访问点，进行全局的流程控制；
2. HandlerMapping：处理器映射器，用于查找处理器；
3. HandlerAdapter：处理器适配器，用于执行处理器；
4. ViewResolver：视图解析器，用于解析逻辑视图ModelAndView，返回View视图；
5. ModelAndView：逻辑视图；
6. Controller：处理器，用于处理用于请求；

2.4 说一下SpringMVC或Spring或SpringBoot或SpringCloud的常用注解有哪些？

@Component、@Controller、@RestController、@Service、@Repository、
@ControllerAdvice、@RestControllerAdvice、@ExceptionHandler
@Autowired @RequestMapping、@PostMapping、@GetMapping、@PutMapping、@DeleteMapping...
@RequestParam、@RequestBody、@ResponseBody、@PathVariable

@Configuration、@Bean、@Qualifier、@Value、@ComponentScan、@EnableWebMvc
@PropertySource、@ConfigurationProperties
@ConditionalOnClass、@ConditionalOnBean... 、@ConditionalOnXxx类型的注解有一大堆
@SpringBootApplication、@EnableAutoConfiguration、@SpringBootConfiguration、@EnableTransactionManagement

@EnableEurekaServer、@EnableDiscoveryClient、@EnableFeignClients
...

2.5 说一下@RequestMapping注解的作用？

1. @RequestMapping相当于RequestMappingHandlerMapping和RequestMappingHandlerAdapter组合；
2. @RequestMapping可以使用value(path)属性将请求的url映射到处理器的方法之上；
3. @RequestMapping默认可以接收所有类型的http请求方法(如POST、GET、put、delete..)，可以使用它的method属性来指定某种具体的http请求方法；

4. 在Spring4.3之后，Spring提供了@GetMapping、@PostMapping、@PutMapping、@DeleteMapping等来处理http请求方法的注解；

2.7 说一下SpringMVC和struts2的区别？

1. 底层使用的框架实现不同

   Struts2采用Filter（StrutsPrepareAndExecuteFilter）实现；

   SpringMVC则采用Servlet（DispatcherServlet）实现；

2. 请求的拦截机制不同

   Struts2是类级别的请求拦截方式，每次请求就会创建一个Action

   SpringMVC是方法级别的请求拦截方式，处理器默认是单例的，如果Spring和Struts2整合时，Action的作用域要设置prototype

3. 性能方面

   由于SpringMVC基于方法的拦截，Controller默认是单例模式，只加载一次；

   而Struts2是类级别的拦截，每次请求对应一个新的Action实例，并且每次都需要加载所有的属性值注入；

   所以，SpringMVC效率和性能高于Struts2；

2.8 SpringMvc怎么将url映射到Controller

1. SpringMVC主要通过处理器映射器来映射url和Controller；
2. SpringMVC提供了常用的几种处理器映射器，如：

   BeanNameUrlHandlerMapping

   SimpleUrlHandlerMapping

   RequestMappingHandlerMapping

   BeanNameUrlHandlerMapping使用Spring bean的name属性作为url，将url映射到Controller；

   SimpleUrlHandlerMapping使用自己的mappings属性将url映射到Controller，这里的url对应了Spring bean的id；

   RequestMappingHandlerMapping可以通过@RequestMapping注解将url映射Controller的方法上，但它必须配合ReuqestMappingHandlerAdapter使用；

3.SpringBoot

3.1 接触过SpringBoot吗？有没有用过SpringBoot？说一下你对SpringBoot的理解？你怎么理解SpringBoot的？SpringBoot是什么？SpringBoot有哪些优点？

如果问前边两个问题，直接说使用过，然后说一下什么项目中用的即可；

后边三个问题的回答，直接回答SpringBoot是什么，SpringBoot的优点即可；

- SpringBoot主要用来简化Spring开发过程；
- 它不是一个全新的MVC框架，自动配置了Spring框架和第三方类库；
- 它的核心思想"约定大于配置"；
- SpringBoot内置Servlet容器，如Tomcat、Jetty或者Undertow(不需要部署war包)；
- 提供了"starter"依赖(maven依赖)来简化构建配置；
- 提供了准生产级功能：如监控、健康检查等一些额外的配置；
- 完全没有代码生成和XML配置；

3.2 请谈一下SpringBoot的自动配置原理？

- SpringBoot借助Spring提供的java config方式将Bean加载到容器中；

  在SpringBoot的启动类上有个@SpringBootApplication注解，这个注解由@SpringBootConfiguration、@EnableAutoConfiguration和@ComponentScan组成；

  SpringBoot利用@EnableAutoConfiguration注解引入的AutoConfigurationImportSelector类，在这个类中，使用SpringFactoriesLoader类，将classpath下所有jar包中的META-INF/Spring.factories文件中所有符合自动配置条件的类型进行加载；

  最后通过反射，利用Java Config，将bean加载Spring的ioc容器中，完成自动配置；

3.3 你在项目中SpringBoot配置文件用的什么格式的？

在项目中目前使用yaml文件格式(.yml结尾)；

紧着可能会问为什么用这种格式，这个时候直接回答YAML的优点

- YAML是一个类似 XML、JSON 的标记性语言，YAML 强调以数据为中心；
- 容易阅读；

- 可用于不同程序间的数据交换；
- 适合描述程序所使用的数据结构，特别是脚本语言；
- 丰富的表达能力与可扩展性；
- 易于使用；

3.4 请说一下SpringBoot和Spring MVC的区别？

- Spring MVC是建立在Servlet基础之上的一个MVC框架，主要解决WEB开发的问题；
- 由于在使用Spring MVC开发WEB应用过程中的配置非常复杂，如XML配置或者JavaConfig方式处理起来比较繁琐；
- Spring boot为了简化开发者的使用，采用约定大于配置的思想，简化了Spring MVC的配置流程；
- Spring Boot内嵌了Servlet容器，很容易创建一个独立运行、准生产级别的Spring项目。

3.5 SpringBoot如何处理跨域？

在使用SpringBoot的情况下可以使用下面几种方式处理跨域请求

1. 实现WebMvcConfigurer接口的addCorsMappings()方法；
   但是使用此方法后，再使用自定义拦截器时，处理跨域就会失效。
   原因是当请求到达时会先进入拦截器中，而不是进入Mapping映射中，所以响应头信息中并没有配置的跨域信息，浏览器就会报跨域异常；
2. 使用Spring web自带的CorsFilter处理；
3. 使用自定义的过滤器（或者拦截器）实现，但是这个过滤器（或者拦截器）一定要配置所有过滤器（或者拦截器）的最前面；

附一：什么是跨域？

当一个请求url的协议、域名、端口三者之间任意一个与当前页面url不同即为跨域

当前页面url	被请求页面url	是否跨域	原因
http://www.et.com/	http://www.et.com/index.html	否	同源（协议、域名、端口号相同）
http://www.et.com/	https://www.et.com/index.html	跨域	协议不同（http/https）
http://www.et.co	http://www.etoak.com/	跨域	主域名不同（et/etoak）

当前页面url	被请求页面url	是否跨域	原因
http://www.et.com/	http://wiki.et.com/	跨域	子域名不同（www/wiki）
http://www.et.com:8080/	http://www.et.com:9090/	跨域	端口号不同（8080/9090）

附二：CORS跨域

1. CORS是一个W3C标准，全称是"跨域资源共享"（Cross-Origin Resource Sharing）；
2. CORS允许浏览器向跨域服务器发出AJAX请求，从而克服了AJAX只能访问同域资源的限制；
3. CORS需要浏览器和服务器同时支持；
4. 整个CORS通信过程，都是浏览器自动完成，不需要用户参与。
5. 对于开发者来说，CORS通信与同源的AJAX通信没有差别，代码完全一样。

附三：SpringBoot配置跨域方式

- 方式一：实现WebMvcConfigurer接口的addCorsMappings()方法 -- 不推荐

```
public void addCorsMappings(CorsRegistry registry) {
  registry.addMapping("/**")      // 配置可以被跨域的路径，可以任意配置，可以具体到直接请求路径。
  .allowedOrigins("*") //允许所有的请求域名访问我们的跨域资源，可以固定单个或者多个内容
  .allowedMethods("POST", "GET", "PUT", "OPTIONS", "DELETE") // 允许何种请求方法访问该跨域资源服务器
  .allowCredentials(true)    // 是否允许用户发送、处理 cookie
  .allowedHeaders("*")       // 允许所有的请求header访问，可以自定义设置任意请求头信息
  .maxAge(3600);             // 预检请求的有效期，单位为秒。有效期内，不会重复发送预检请求
}
```

- 方式二：使用Spring-web提供的CorsFilter

```
@Configuration
public class CorsConfig {
  private CorsConfiguration addcorsConfig() {
```

```
    CorsConfiguration corsConfiguration = new CorsConfiguration();
    List<String> list = new ArrayList<>();
    list.add("*");
    corsConfiguration.setAllowedOrigins(list);
    corsConfiguration.addAllowedOrigin("*");
    corsConfiguration.addAllowedHeader("*");
    corsConfiguration.addAllowedMethod("*");
    corsConfiguration.setAllowCredentials(true);
    corsConfiguration.setMaxAge(3600L);
    return corsConfiguration;
  }

  @Bean
  public CorsFilter corsFilter() {
    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", addcorsConfig());
    return new CorsFilter(source);
  }
}
```

- 方式三：自定义过滤器（或者拦截器），要注意，这个拦截器一定要放在所有的拦截器的第一个位置

```
public class CrossDomainFilter implements Filter {
  @Override
  public void doFilter(ServletRequest request, ServletResponse
response, FilterChain chain)
    throws IOException, ServletException {
    HttpServletResponse httpResponse = (HttpServletResponse)
response;
    httpResponse.setHeader("Access-Control-Allow-Origin", "*");
    httpResponse.setHeader("Access-Control-Allow-Methods", "POST,
GET, OPTIONS, DELETE");
    httpResponse.setHeader("Access-Control-Max-Age", "3600");
    httpResponse.addHeader("Access-Control-Allow-Headers",
                           "Accept, Origin, No-Cache, X-Requested-
With, If-Modified-Since, Pragma, Last-Modified, Cache-Control,
Expires, Content-Type, X-E4M-With");
    httpResponse.setHeader("Access-Control-Allow-Credentials",
"true");
```

```
    chain.doFilter(request, httpResponse);
  }
}
```

```
public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
  if (request.getHeader(HttpHeaders.ORIGIN) != null) {
    response.addHeader("Access-Control-Allow-Origin", "*");
    response.addHeader("Access-Control-Allow-Credentials", "true");
    response.addHeader("Access-Control-Allow-Methods", "POST, GET,
OPTIONS, DELETE, PUT, HEAD");
    response.addHeader("Access-Control-Allow-Headers", "*");
    response.addHeader("Access-Control-Max-Age", "3600");
  }
  return true;
}
```

• 3.6 说一下项目中Springboot和dubbo是怎么搭建的？

1. Springboot整合Dubbo非常简单，直接引入Dubbo官方提供的starter(dubbo-Spring-boot-starter)即可。
2. 然后在配置文件配置应用名称、注册中心、接口暴露方式等；
3. 在SpringBoot的启动类上加上@EnableDubbo注解，启动Dubbo配置；
4. 服务提供者发布服务时使用@Service注解；
5. 服务消费者消费服务时使用@Reference注解；

4. 微服务、SpringCloud

• 4.1 微服务的框架了解过嘛？谈谈你对微服务的理解？还了解哪些微服务框架？

了解过，在xx项目使用过Dubbo；在xx项目使用过SpringCloud；

总体来说，微服务是一种架构风格，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合；
对于一个大型复杂的业务系统：

- 它的业务功能可以拆分为多个独立的服务；
- 各个服务之间是松耦合的；
- 通过远程协议进行通信（异步、同步）；
- 各个微服务均可被独立部署、扩容、升降级。

其它微服务框架还有Motan、GRPC、Thrift等；
接下来可能会让你对比一下Dubbo和SpringCloud。

4.2 说一下Dubbo和SpringCloud的区别

- SpringCloud借助SpringBoot提供一套完整的微服务解决方案，而Dubbo的定义是一款高性能的RPC框架；
- 如果非要对比的话，捡几个重要的点进行对比一下：

1. 在服务调用方式方面，Dubbo采用RPC方式，而SpringCloud使用REST方式；
2. 在注册中心方面，Dubbo推荐使用Zookeeper，而SpringCloud推荐使用Eureka；
3. 在服务网关方面，Dubbo没有服务网关，而SpringCloud提供了Zuul和Spring cloud gateway；
4. 在断路器方面，Dubbo不太完善，而SpringCloud整合了Hystrix；
5. 在分布式配置方面，Dubbo没有提供，而SpringCloud提供了Spring cloud config；
6. 在服务跟踪方面，Dubbo没有提供，而SpringCloud提供了Spring cloud sleuth；
   等等

附：对比列表

组件	Dubbo	SpringCloud
服务注册中心	Zookeeper	Eureka
服务调用方式	RPC	REST API
服务网关	无	Zuul、GateWay
断路器	不完善	Spring Cloud Netflix Hystrix
分布式配置	无	Spring Cloud Config
服务跟踪	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
.....		

4.3 对SpringCloud了解吗？

了解；

- SpringCloud是在2014年底由Spring团队推出的，它的目标是使其成为Java领域的微服务架构落地标准；
- SpringCloud基于SpringBoot开发，提供了一套完整的微服务解决方案，包括：服务注册与发现、配置中心、Api网关、断路器、全链路监控、控制总线、智能路由等，并可以根据需要进行扩展和替换；

4.4 说一下SpringCloud都有哪些组件？

这里提供组件列表，捡主要的说几个

组件名称	所属项目	组件分类
Eureka	spring-cloud-netflix	注册中心
Zuul	spring-cloud-netflix	第一代网关
Ribbon	spring-cloud-netflix	负载均衡
Hystrix	spring-cloud-netflix	熔断器
Turbine	spring-cloud-netflix	集群监控
Sidecar	spring-cloud-netflix	多语言
Feign	spring-cloud-openfeign	声明式Http客户端
Consul	spring-cloud-consul	注册中心
GateWay	spring-cloud-gateway	第二代网关
Sleuth	spring-cloud-seluth	链路追踪
Config	spring-cloud-config	配置中心
Bus	spring-cloud-bus	总线
Popeline	spring-cloud-pipeline	部署管道
Dataflow	spring-cloud-dataflow	数据处理

5. Dubbo相关面试

5.1 有没有用过dubbo，请介绍一下

有，直接说什么项目中有使用；

Dubbo是一款高性能、轻量级的开源Java RPC框架，

它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

5.2 请说一下Dubbo由哪些角色组成？

1. Provider: 暴露服务的服务提供方。
2. Consumer: 调用远程服务的服务消费方。
3. Registry: 服务注册与发现的注册中心。
4. Monitor: 统计服务的调用次数和调用时间的监控中心。
5. Container: Dubbo容器。

5.3 为什么要用Dubbo？

- 国内很多互联网公司都在用，已经经过很多线上考验。它内部使用了 Netty、Zookeeper，保证了高性能高可用性；
- Dubbo是阿里开源项目，目前已经贡献给Apache，在2019年5月20日升级为Apache的顶级项目；
- 使用Dubbo可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求；

5.4 Dubbo的直连用过吗？

在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，这时候可能需要点对点直连，

这种直连方式以服务接口为单位，忽略注册中心的提供者列表，接口配置点对点，不影响其它接口从注册中心获取列表。

具体配置如下

```
<dubbo:reference id="xxxService"
                 interface="com.xxx.XxxService"
                 url="dubbo://服务ip:20880" />
```

5.5 你们调用dubbo接口出现过冲突问题吗？是怎么解决的？

这个应该是考察"服务分组"

没有出现过;

- 如果是一个接口有多个实现的时候，可以用group属性来分组，服务提供方和消费方都指定同一个 group 即可。

5.6 dubbo消费者怎么获取连接地址？ dubbo消费端是怎么获取服务端发布的地址的？其中注册中心起了什么作用？

服务消费者在启动时，向注册中心订阅自己所需的服务。

注册中心返回服务提供者地址列表给消费者，如果有变更（新服务上线、服务下线），注册中心将基于长连接推送变更数据给消费者。

5.7 你在项目使用Dubbo的什么协议？还有其他协议吗？

在项目中主要使用dubbo协议，Dubbo官方也是推荐这种协议；
除了dubbo协议之外，Dubbo还提供了

- rmi://
- hessian://
- http://
- webservice://
- thrift://
- memcached://
- redis://
- rest://

5.8 你们使用Dubbo时使用什么注册中心？还有了解其他注册中心吗？

我们在项目主要使用Zookeeper注册中心，Dubbo官方也推荐使用Zookeeper作为注册中心；

因为Zookeeper是Apacahe Hadoop的子项目，是一个树型的目录服务，支持变更推送，适合作为Dubbo服务的注册中心；

其他注册中心还有Multicast、Redis、Simple、Nacos；

5.9 Dubbo有哪几种负载均衡策略，默认是哪种？

- Random LoadBalance 随机，按权重设置随机概率。
- RoundRobin LoadBalance 轮询，按公约后的权重设置轮询比率。
- LeastActive LoadBalance 最少活跃调用数，相同活跃数的随机，活跃数指调用前后计数差。
- ConsistentHash LoadBalance 一致性 Hash，相同参数的请求总是发到同一提供者。
- 默认的负载均衡策略是"随机"策略；

5.10 服务提供者能实现失效踢出是什么原理？

服务失效剔出基于Zookeeper的临时节点原理。

附：ZK提供了"持久化节点"和"临时节点"两种；临时节点失效的情况：

- 当创建该节点客户端会话因超时或主动关闭时；
- 当某个客户端(不一定是创建者)主动删除该节点时；

5.11 Dubbo的管理控制台能做什么？

Dubbo管理控制台主要用来进行服务治理，其包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡等管理功能。

5.12 Dubbo启动时如果依赖的服务或者注册中心不可用会怎样？

Dubbo默认会在启动时检查依赖的服务是否可用或者注册中心是否启动，不可用时会抛出异常，阻止初始化完成；

Dubbo在配置dubbo服务引用时可以使用check属性配置检查服务是否可用，check默认是true，表示检查服务是否启动，我们可以将其设置为false；

Dubbo在配置注册中心时，也可以设置check属性的值来检查注册中心是否启动；

6. Redis（全称：Remote Dictionary Server）

6.1 说一下Redis有什么特点？

1. Redis基于内存的K-V数据库，访问速度快；
2. 支持数据的持久化(可以将数据保存到硬盘，重启Redis之后可以重新写入内存)；
3. 支持丰富数据类型，主要包括string、list、hash、set、zset(sorted set)
   Redis5.0新增了Stream类型
4. 支持主从数据备份；
5. 支持事务；
6. 支持集群；

6.2 说一下Redis都有什么数据类型？

Redis常用的数据类型有string、list、set、zset（sorted set）和hash类型；

Redis5.0以后添加了新的数据类型stream；

6.3 说一下Redis和MemCached的区别

1. 在存储方式方面

   Memecached把数据全部存在内存之中，不能持久化数据；

   Redis支持数据的持久化（RDB和AOF两种）；

2. 在数据类型支持方面

   Redis在数据支持上要比memecached多的多。

3. 在底层模型方面：

   Redis在2.0版本后增加了自己的VM特性，突破物理内存的限制；

   Memcached可以修改最大可用内存，采用LRU算法

4. 数据一致性方面

   Memcache 在并发场景下，用CAS保证一致性；

   Redis事务支持比较弱，只能保证事务中的每个操作连续执行；

附：Memcached的介绍

1. Memcached的优点：

Memcached可以利用多核优势，单实例吞吐量极高，可以达到几十万QPS（取决于key、value的字节大小以及服务器硬件性能，日常环境中QPS高峰大约在4-6w左右）。

2. Memcached的局限性：

   只支持简单的key/value数据结构，不像Redis可以支持丰富的数据类型。

   无法进行持久化，数据不能备份，只能用于缓存使用，且重启后数据全部丢失。

   无法进行数据同步，不能将Memcached中的数据迁移到其他Memcached实例中。

   Memcached内存分配采用Slab Allocation机制管理内存，value大小分布差异较大时会造成内存利用率降低，并引发低利用率时依然出现踢出等问题。

   需要用户注重value设计

6.4 关于Redis的回收策略，过期机制的相关问题

- 问题1：redis回收策略、过期机制？Redis内存满了怎么办？有什么替换策略？
- 回答：

  1. Redis内存数据集大小上升到一定大小的时候，会实行数据淘汰策略（回收策略）
  2. Redis key过期的方式有三种

     被动删除：当读/写一个已经过期的key时，会触发惰性删除策略，直接删除掉这个过期key；

     主动删除：由于惰性删除策略无法保证冷数据被及时删掉，所以Redis会定期主动淘汰一批已过期的key；

     当前已用内存超过maxmeory限定时，触发主动清理策略；
  3. 当前已用内存超过maxmemory限定时，触发主动清理策略，主动清理策略有6种：

     `volitile-lru` ：从已设置过期时间的数据集中挑选"最近最少使用"的数据进行淘汰；

     `volitile-ttl` ：从已设置过期时间的数据集中挑选"将要过期"的数据进行淘汰；

     `volitile-random` ：从已设置过期时间的数据集中"任意"挑选数据进行淘汰；

     `allkeys-lru` ：从数据集中挑选"最近最少使用"的数据进行淘汰；

     `allkeys-random` ：从数据集中"任意"选择数据进行淘汰；

`no-envicition` ：设置永不过期，禁止驱逐数据；

• 6.5 Redis缓存怎么与数据库数据保持一致？（缓存与数据双写一致）

1. 如果数据要求强一致性，最好不要将数据放到缓存；
2. 什么是缓存与数据库双写一致问题？

   如果仅仅查询的话，缓存的数据和数据库的数据是没问题的。

   但是，当我们要更新时候呢？各种情况很可能就造成数据库和缓存的数据不一致了。

   比如：数据库中存储的某篇文章的点赞数为10，而Redis中存储的点赞数为11；

3. 对于读操作

   如果数据在Redis缓存里，那么就直接取Redis缓存的数据。

   如果数据不在Redis缓存里，那么先去查询数据库，然后将数据库查出来的数据写到Redis缓存中，最后将数据返回；

4. 从理论上说，只要我们设置了key的过期时间，我们就能保证缓存和数据库的数据最终是一致的。因为只要缓存数据过期了，就会被删除。之后再读的时候，如果缓存里没有，可以查数据库的数据，然后将查出来的数据写入到缓存中。

5. 除了设置过期时间，我们还需要做更多的措施来尽量避免数据库与缓存处于不一致的情况发生。
   这个时候，跟面试官说一下几种处理策略：

   5.1 先更新数据库，再删除缓存

   这种策略会出现如下情况

   情况1：如果更新数据库就失败了，程序可以直接返回错误(Exception)，不会出现数据不一致。

   情况2：更新数据库成功，删除缓存失败；这就会导致数据库里是新数据，而缓存里是旧数据。

   解决情形2思路：

先删除缓存，再更新数据库。如果数据库更新失败了，那么数据库中是旧数据，缓存中是空的，那么数据不会不一致。这样读的时候缓存没有，所以去读了数据库中的旧数据，然后更新到缓存中。

将需要删除的key发送到消息队列中，自己消费消息，获得需要删除的key，不断重试删除操作，直到成功

5.2 先删除缓存，再更新数据库

这种策略会出现如下情况

情况1：删除缓存成功，更新数据库失败，这样的话，数据库和缓存的数据最终是一致的。

情况2：删除缓存失败了，我们可以直接返回错误(Exception)，数据库和缓存的数据还是一致的。

但是在并发场景下，就会下面的问题：

- 线程1删除了缓存；
- 线程2查询，发现缓存已不存在；
- 线程2去数据库查询得到旧值；
- 线程2将旧值写入缓存；
- 线程1将新值写入数据库；
- 所以也会导致数据库和缓存不一致的问题；
- 解决思路：将删除缓存、修改数据库、读取缓存等的操作积压到队列里边，实现串行化。

贴两篇相关文章文章：

`https://github.com/doocs/advanced-java/blob/master/docs/high-concurrency/redis-consistence.md`

`https://zhuanlan.zhihu.com/p/48334686`

6.6 Redis负载均衡了解吗？怎么实现的？

- 利用Redis官方提供的Redis集群方案实现Redis数据的负载均衡；
- Redis集群自动分割数据到不同的节点上，整个集群的部分节点失败或者不可达的情况下能够继续处理命令。
- Redis集群引入了哈希槽的概念，集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

6.7 你们Redis做读写分离了么

- 不做读写分离，我们用的是Redis集群架构，是属于分片集群的架构；
- Redis本身在内存上操作，不会涉及IO吞吐，即使读写分离也不会提升太多性能；
- Redis在生产上的主要问题是考虑容量，单机最多10-20G，key太多降低Redis性能，因此采用分片集群结构，已经能保证了我们的性能。
- 如果使用了读写分离后，还要考虑主从一致性，主从延迟等问题，会增加务复杂度。

6.8 Redis集群机制中，你觉得有什么不足的地方吗？

1. 默认不支持批量操作，如不支持mset、mget等命令；
2. 如果一个key对应的value是Hash类型的，若Hash对象非常大，是不支持映射到不同节点的，只能映射到集群中的一个节点上！
3. 不支持事务

6.9 了解Redis的多数据库机制吗？了解多少？

1. 单机下的Redis默认支持16个数据库；但是：

   不支持自定义数据库名称。

   不支持为每个数据库设置访问密码。

   数据库之间不是完全隔离的，FLUSHALL命令会清空所有数据库的数据。

   多数据库不适用存储不同应用的数据。

2. 我们生产环境中使用Redis集群，集群架构下只有一个数据库空间，即db0。

   因此，我们没有使用Redis的多数据库功能！

6.10 说一下Redis事务？

1. Redis对事务的支持比较简单，它是一组命令的集合，命令被依次顺序的执行；
2. 我们在生产上采用的是Redis集群，不同的key是可能分到不同的Redis节点上的，在这种情况下Redis的事务机制是不生效的；
3. Redis事务不支持回滚操作，所以基本没有使用过；

6.11 Redis持久化的两种方式及应用场景？

1. Redis持久化分成两种方式：RDB（redis database）、AOF（append only file）

2. RDB是在不同的时间点，将Redis某一时刻的数据生成快照并存储到磁盘上。
3. AOF是只允许追加不允许改写文件，是将Redis执行过的所有写指令记录下来，在下次redis重启的时候，只要把这些写指令从前到后重复执行一遍，就可以实现数据恢复。
4. RDB持久化的触发方式：自动触发和手动触发，自动触发方式在redis.conf文件配置

   `save 900 1` 900秒内至少有一个key被改变就自动触发RDB持久化

   `save 300 10` 300秒内至少有10个key被改变就自动触发RDB持久化

   `save 60 10000` 60秒内至少有10000个key被改变就自动触发RDB持久化

   手动触发方式

   `save命令` ：执行此命令会阻塞Redis服务器，执行命令期间，Redis不能处理其它命令，直到RDB过程完成为止。

   `bgsave命令` ：执行该命令时，Redis会在后台异步进行快照操作，做快照的同时还可以响应客户端请求；此时Redis进程执行"fork操作"创建子进程，持久化过程由子进程负责，完成后自动结束。阻塞只发生在fork阶段，一般时间很短。
5. AOF持久化

   默认的AOF持久化策略是每秒钟一次同步策略；

   AOF的同步策略还有always和no两种，加上everysec，共三种；
6. 说完这些之后，可能还会问：RDB和AOF可以同时使用吗？

   RDB和AOF两种方式可以同时使用，这时如果Redis重启，则会优先采用AOF方式进行数据恢复，因为AOF方式的数据恢复完整度更高；

7 定时任务相关

7.1 定时任务怎么实现的

1. jdk自带的库中,有两种技术可以实现定时任务：
   - 一种是使用Timer；
   - 另外一个则是ScheduledThreadPoolExecutor；
2. 使用Quartz框架、当当网的Elastic-job、XXL-JOB、Spring task

- 附1：Timer方式

```
public class TimerDemo {
```

```
  public static void main(String[] args) {
    //创建定时器对象
    Timer timer = new Timer();

    //在2秒后执行EtoakTask类中的run方法，后面每5秒执行一次
    timer.schedule(new EtoakTask(), 2000, 5000);
  }
}

class EtoakTask extends TimerTask {
  @Override
  public void run() {
    System.out.println("hello world");
  }
}
```

- 附2：ScheduledThreadPoolExecutor方式

```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
public class Task3 {
  public static void main(String[] args) {
    Runnable runnable = new Runnable() {
      public void run() {
        System.out.println("Hello !!");
      }
    };
    ScheduledExecutorService service =
Executors.newSingleThreadScheduledExecutor();
    service.scheduleAtFixedRate(runnable, 0, 1, TimeUnit.SECONDS);
  }
}
```

7.2 Spring整合Quartz定时任务的配置

```
<!-- JobDetailFactoryBean -->
```

```
<bean id="jobDetail"
  class="org.Springframework.scheduling.quartz.JobDetailFactoryBean">
  <property name="jobClass" value="com.etoak.job.ClusterJob" />
  <!-- 持久化任务 -->
  <property name="durability" value="true" />
  <property name="name" value="clusterJob" />
  <property name="group" value="group1" />
</bean>

<!-- CronTriggerFactoryBean -->
<bean id="trigger"
  class="org.Springframework.scheduling.quartz.CronTriggerFactoryBean"
>
  <property name="jobDetail" ref="jobDetail" />
  <property name="cronExpression" value="0/5 * * * * ?" />
  <property name="name" value="clusterTrigger" />
  <property name="group" value="group1" />
</bean>

<!-- SchedulerFactoryBean -->
<bean id="scheduler"
class="org.Springframework.scheduling.quartz.SchedulerFactoryBean">
  <property name="triggers">
    <list>
      <ref bean="trigger" />
    </list>
  </property>
  <property name="configLocation" value="classpath:quartz.properties"
/>
  <property name="dataSource" ref="dataSource" />
  <property name="transactionManager" ref="tx" />
  <!-- 解决Spring管理的bean无法注入任务类中
       解决方式一 ：把Spring ioc容器放到调度器的容器中(context中) -->
  <property name="applicationContextSchedulerContextKey" value="ioc"
/>
</bean>
```

- **7.3 cron表达式怎么写**

```
cron表达式的七个字段：
秒　分　时　天　月　周　[年]
```

7.4 Quartz定时任务每天0点15分执行任务式子怎么写？

8. 消息队列

8.1 消息队列有没有接触过？简单介绍一下？

了解，消息队列中间件是分布式系统中重要的组件，主要解决应用解耦，异步消息，流量削峰等问题，实现高性能，高可用，可伸缩和最终一致性架构；

目前使用较多的消息队列有ActiveMQ，RabbitMQ，ZeroMQ，Kafka，MetaMQ，RocketMQ；

8.2 说一下消息的类型？

- 点对点Point To Point（P2P）
- 发布/订阅Publish-Subscribe（Pub\Sub）类型

8.3 Spring整合Activemq如何配置？

- 生产者和消费者公共配置

```
<!-- ActiveMQConnectionFactory -->
<bean id="mqConnectionFactory"
class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="userName" value="" />
  <property name="password" value="" />
  <property name="brokerURL" value="tcp://localhost:61616" />
  <!-- 是否进行异步发送 -->
  <property name="useAsyncSend" value="true" />
</bean>

<!-- CachingConnectionFactory -->
<bean id="connectionFactory"
class="org.Springframework.jms.connection.CachingConnectionFactory"
>
```

```
  <constructor-arg name="targetConnectionFactory"
ref="pooledConnectionFactory" />
  <property name="sessionCacheSize" value="10" />
</bean>

<!-- JmsTemplate -->
<bean id="jmsTemplate"
class="org.Springframework.jms.core.JmsTemplate">
  <property name="connectionFactory" ref="connectionFactory" />
  <!-- 表示开启持久化、优先级、消息生存时间 -->
  <property name="explicitQosEnabled" value="true" />
  <!-- 是否持久化，默认就是持久化（值为2） -->
  <property name="deliveryMode" value="2" />
  <!-- 客户端签收方式:默认是1(表示自动签收)，2表示客户端手动签收 -->
  <property name="sessionAcknowledgeMode" value="2" />
</bean>
```

- 消费者需要增加的配置

```
<!-- 配置消费消息的来源(email队列) -->
<bean id="emailQueue"
class="org.apache.activemq.command.ActiveMQQueue">
  <!-- 邮件队列的名称 -->
  <constructor-arg name="name" value="email-queue" />
</bean>

<!-- 自定义的监听器：用于监听邮件队列 -->
<bean id="emailListener" class="com.etoak.listener.EmailListener"
/>

<!-- 配置监听器容器 -->
<bean
class="org.Springframework.jms.listener.DefaultMessageListenerConta
iner">
  <property name="connectionFactory" ref="connectionFactory" />
  <property name="messageListener" ref="emailListener" />
  <!-- 消费者的消息来源 -->
  <property name="destination" ref="emailQueue" />
</bean>
```

9. Web Service

9.1 说一下WSDL指的是什么？

Web服务描述语言，用来描述Web服务和说明如何与Web服务通信的XML语言，由于是基于XML的，所以WSDL具有非常好的可阅读性。

9.2 Web Service用的什么框架怎么生成客户端？

- 目前使用web service的框架是CXF；
- CXF提供了wsdl2java工具用于生成web service的客户端；
- 执行命令：wsdl2java –d d:/client http://localhost:9090/hello?wsdl

9.3 Web Service三要素

1. SOAP (Simple Object Access Protocol)：简易对象访问协议，基于HTTP协议，采用XML格式，用来传递信息的格式。
2. WSDL (Web Services Description Language)：Web服务描述语言，用来描述如何访问具体的服务。
3. UDDI (Universal Description Discovery and Integration)：通用描述、发现及整合，用来管理、分发、查询web Service。

10. 用过linux吗？知道简单的一些命令吗？

用过；列举几个常用命令，咱们上课都讲过了

- mv：重命名、移动文件；
- rm：删除；
- cp：拷贝；
- scp：远程拷贝；
- ps：查看进程；　查看java进程：ps -ef|grep java、ps aux|grep java
- netstat：用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况；netstat -tpln|grep 8080
- chmod：修改文件权限；
- tar：压缩、解压缩tar包

- zip、unzip：压缩、解压缩zip包
- less、more、cat、tail、head：操作文件相关的命令

11. Mybatis缓存问题

一级缓存

SqlSession

二级缓存

- 开启二级缓存，在mapper的配置文件中 `<cache></cache>` ；
- 如果有的sql不需要二级缓存，那么在 `<select>` 标签配置 `useCache="false"` ；
- 如果使用二级缓存，缓存自定义对象时，一定要将bean进行序列化(实现Serializable)

12. Spring Bean Scope

面试题：Controller默认是单例的吗？如果是，会不会有线程问题？怎么解决？

- Controller默认是单例；
- 会；
- 不要将参数写到成员变量上，所有的参数都要写在方法参数上；

Struts2的Action不能写在方法参数上，只能写在成员变量，所以要将Action配置成prototype；