**Java框架**

# Mybatis

## 1.1、什么是Mybatis

​ (1) MyBatis 是一个**半ORM(对象关系映射)框架**，它内部封装了JDBC,开发时只需要关注SQL语句本事，**不需要花费精力去处理加载驱动，创建连接，创建statement等复杂的过程**，程序员直接编写原生态sql,可以严格控制sql执行性能，灵活度高。

​ （2）MyBatis 可以使用XML或者注解来配置和映射原生信息，**将POJO映射成数据库中的记录**，避免了几乎所有的JDBC代码和手动设置参数以及获取结果集。

​ （3）通过xml 文件或注解方式将要执行的各种statement 配置起来，并通过java 对象和statement中SQL的动态参数进行映射生成最终执行的sql语句，最后由mybatis 框架执行SQL并将结果映射为java对象并返回。

## 1.2、Mybatis 的优点

​ （1）基于SQL语句编程，相当**灵活**，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在xml里，**解除了SQL与程序代码的耦合**，便于统一管理，提供xml标签，支持编写SQL语句，并可以重用。

​ （2）与 JDBC相比，减少了50%以上的代码量，**消除了JDBC大量冗余的代码**，不需要手动开关连接

​ （3）很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库， 所以只要JDBC支持的数据库MyBatis都支持）。

​ （4）能够与Spring很好的集成；

​ （5）**提供映射标签**，支持对象与数据库的ORM字段关系映射；提供对象 关系映射标签，支持对象关系组件维护

1.3 MyBatis与Hibernate有哪些不同？

​ （1）Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis 需要程序员自己编写Sql语句。

​ （2）Mybatis直接编写原生态SQL，可以严格控制SQL执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频 繁，一但需求变化要求迅速输出成果。但是灵活的前提是Mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套SQL映 射文件，工作量大。

​ （3）Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用Hibernate开发可以节省很多代码，提高效率

## 1.4、#{}和${}的区别

​ #{}是预编译处理，${}是字符串替换。

​ Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用 PreparedStatement 的set方法来赋值Mybatis在处理时 ，就是把 {}时，就是把时，就是把{}替换成变量的值

​ 使用#{}可以有效的防止SQL注入，提高系统安全性。

## 1.5、Mybatis是如何进行分页的？分页插件的原理是什么？

​ Mybatis使用**RowBounds**对象进行分页，它是针对ResultSet结果集执行的**内存分页，而非物理分页**。可以在SQL内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

​ 分页插件的基本原理是**使用Mybatis提供的插件接口**，实现自定义插件，在插件的**拦截方法内拦截待执行的SQL，然后重写SQL**，根据dialect方言，添加对应的物理分页语句和物理分页参数。

## 1.6、Mybatis动态SQL有什么用？执行原理？有哪些动态 sql标签?

**Mybatis动态SQL**可以在Xml映射文件内，以标签的形式编写动态sql

**执行原理**是根据表达式的值完成逻辑判断并动态拼接sql的功能。

**Mybatis提供了9种动态sql标签**：trim | where | set | foreach | if | choose | when | otherwise | bind。

## 1.7、MyBatis实现一对一有几种方式?具体怎么操作的?

​ 有**联合查询和嵌套查询**,联合查询是几个表联合查询,只查询一次, 通过在 resultMap里面配置association节点配置一对一的类就可以完成；

​ 嵌套查询是先查一个表，根据这个表里面的结果的外键id，去再另外一个表里面查询数据,也是通过association配置，但另外一个表的查询通过select 属性配置。

## 1.8、Mybatis是否支持延迟加载？如果支持，它的实现原理是什么?

**​ Mybatis仅支持association关联对象和collection关联集合对象的延迟加 载**，association指的就是一对一，collection指的就是一对多查询。在Mybatis 配置文件中，可以配置是否启用延迟加载**lazyLoadingEnabled=true|false**。

​ 它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时， 进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现 a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把 B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。

## 1.9、Mybatis的一级、二级缓存

​ 1）一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其**存储作用域为 Session**，当 Session **flush 或 close** 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存且不能关闭。

2）二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache， HashMap 存储，不同在于其**存储作用域为 Mapper(Namespace)**，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要手动开启二级缓存，**使用二级缓存属性类需要实现Serializable序列化接口**(可用来保存对象的状态),可在它的映射文件中配置。

​ 3）对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存 Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存 将被 clear。

# 2、SpringBoot

## 2.1、什么是SpringBoot

1）用来简化Spring应用的初始搭建以及开发过程，使用特定的方式来进行配置

2）创建独立的Spring引用程序main方法运行

3）嵌入的tomcat无需部署war文件

4）简化maven配置

5）自动配置Spring添加对应的功能starter自动化配置

## 2.2、SpringBoot优缺点

### 2.2.1、优点

1）独立运行

Spring Boot 内嵌了各种 servlet 容器，Tomcat、Jetty 等，现在不再需要打成war 包部署到容器中，Spring Boot 只要打成一个可执行的 jar 包就能独立运行，所有的依赖包都在一个 jar 包内。

2）简化配置

spring-boot-starter-web 启动器自动依赖其他组件，简少了 maven 的配置。

3）自动配置

Spring Boot 能根据当前类路径下的类、jar 包来自动配置 bean，如添加一个 spring

boot-starter-web 启动器就能拥有 web 的功能，无需其他配置。

4）无代码生成和XML配置

Spring Boot 配置过程中无代码生成，也无需 XML 配置文件就能完成所有配置工作，这一切都是借助于条件注解完成的，这也是 Spring4.x 的核心功能之一。

5）应用监控

Spring Boot 提供一系列端点可以监控服务及应用，做健康检测。

### 2.2.2、缺点

Spring Boot 虽然上手很容易，但如果你不了解其核心技术及流程，所以一旦遇到问题就很棘手，而且现在的解决方案也不是很多，需要一个完善的过程。

## 2.3、SpringBoot、Spring MVC和Spring有什么区别

### 2.3.1、Spring

Spring最重要的特征是依赖注入。所有Spring Modules不是依赖注入就是IOC控制反转。

当我们恰当的使用DI或者是IOC的时候，可以开发松耦合应用。

### 2.3.2、Spring MVC

Spring MVC提供了一种分离式的方法来开发Web应用。通过运用像DispatcherServelet，MoudlAndView 和 ViewResolver 等一些简单的概念，开发 Web 应用将会变的非常简单。

### 2.3.3、SpringBoot

Spring和Spring MVC的问题在于需要配置大量的参数。

SpringBoot通过一个自动配置和启动的项来解决这个问题。

## 2.4、什么是Spring Boot Starter

**启动器是一套方便的依赖描述符**，它可以放在自己的程序中。可以一站式的获取你所需要的Spring和相关技术，而不需要依赖描述符的通过示例代码搜索和复制粘贴的负载。

例如，如果想使用Spring和JPA访问数据库，只需要项目中包含spring-boot-starter-data-jpa 依赖项，你就可以正产是用。

## 2.5、为什么需要spring-boot-maven-plugin

**spring-boot-maven-plugin提供了一些像jar一样打包或者运行应用程序的命令。**

spring-boot:run 运行SpringBoot应用程序；

spring-boot:repackage 重新打包你的jar包或者是war包使其可执行

spring-boot:start和spring-boot:stop管理Spring Boot应用程序的生命周期

spring-boot:build-info生成执行器可以使用的构造信息

## 2.6、什么是YAML

**YAML是一种人类可读的数据序列化语言**。它通常用于配置文件。

与属性文件相比，如果我们想要在配置文件中添加复杂的属性，**YAML文件就更加结构化，而且更少混淆。可以看出YAML具有分层配置数据。**

## 2.7、SpringBoot自动配置的原理

在Spring程序main方法中，添加@SpringBootApplication或者@EnableAutoConfiguration会**自动去maven中读取每个starter中的spring.gfactories文件**，改文件里配置了所有需要被创建的Spring容器中的bean

## 2.8、RequestMapping和GetMapping的不同之处

**RequestMapping具有类属性的**，可以进行GET、POST、PUT或者其他的注释中具有的请求方法。

GetMapping是Get请求方法中的一个特例，它只是RequestMapping的一个延伸，目的是为了提高清晰度。

## 2.9、spring-boot-starter-parent有什么作用

新建一个SpringBoot项目，默认都是有parent的，这个parent就是spring-boot-starter-parent，spring-boot-starter-parent主要有如下作用：

定义了Java编译版本为1.8

使用UTF-8格式编码

继承自spring-boot-dependencies，这里面定义了依赖的版本，也正是因为继承了这个依赖，所以我们在写依赖时才不需要写版本号

执行打包操作的配置

自动化的资源过滤

自动化的插件配置

针对application.peoperties和application.yuml的资源过滤，包括通过profile定义的不同环境的配置文件，例如application-dev.properties和application-dev.yuml。

## 2.10、SpringBoot打成jar和普通的jar区别

Spring Boot 项目最终打包成的 jar 是可执行 jar，这种 jar **可以直接通过java -jar xxx.jar命令来运行，这种 jar 不可以作为普通的 jar 被其他项目依赖**，即使依赖了也无法使用其中的类。

Spring Boot 的 jar 无法被其他项目依赖，主要还是他和普通 jar 的结构不同。**普通的 jar 包，解压后直接就是包名，包里就是我们的代码，而 Spring Boot 打包成的可执行 jar 解压后，在 \BOOT-INF\classes目录下才是我们的代码**，因此无法被直接引用。如果非要引用，可以在 pom.xml 文件中增加配置，将 Spring Boot 项目打包成两个 jar ，一个可执行，一个可引用。

## 2.11、运行SpringBoot有几种方式

打包用命令或者放到容器中运行

用Maven或Gradle插件运行

直接执行main方法运行

## 2.12、开启Spring Boot特性有哪几种方式

继承spring-boot-starter-parent项目

导入spring-boot-dependencies项目依赖

## 2.13、什么是Spring Data

Spring Data 是 Spring 的一个子项目。用于简化数据库访问，支持NoSQL和关系数据存储。其主要目标是使数据库的访问变得方便快捷。Spring Data 具有如下特点：

SpringData 项目支持 NoSQL 存储：

MongoDB （文档数据库）

Neo4j（图形数据库）

Redis（键/值存储）

Hbase（列族数据库）

SpringData 项目所支持的关系数据存储技术：

JDBC

JPA

Spring Data Jpa 致力于减少数据访问层 (DAO) 的开发量. 开发者唯一要做的，就是声明持久层的接口，其他都交给 Spring Data JPA 来帮你完成！Spring Data JPA 通过规范方法的名字，根据符合规范的名字来确定方法需要实现什么样的逻辑。

## 2.14、如何使用Spring Boot实现异常处理

Spring提供了一种使用**ControllerAdvice**处理异常的非常有用的方法。通过实现一个ControlerAdvice类，来处理控制类抛出的所有异常。

## 2.15、如何实现Spring Boot应用程序的安全性

为了实现Spring Boot的安全性，使用spring-boot-starter-security依赖项，并且必须添加安全配置。它只需要很少代码。配置类将必须扩展WebSecurityConfigurerAdapter并覆盖其方法。

## 2.16、Spring Security和Shiro各自的优缺点

由于Spring Boot官方提供了大量的非常方便的开箱即用的Starter，包括Spring Security的Starter，使得在SpringBoot中使用Spring Security变得更加容易，甚至只需要添加一个一来就可以保护所有接口，所以如果是SpringBoot项目，一般选择Spring Security。当然这只是一个建议的组合，单纯从技术上来说，无论怎么组合，都是没有问题的。

Shiro和Spring Security相比，主要有如下特点：

Spring Security是一个重量级的安全管理框架；Shiro则是一个轻量级的安全管理框架；

Spring Security概念复杂，配置繁琐；Shiro概念简单、配置简单；

Spring Security功能强大；Shiro功能简单

## 2.17、Spring Boot中如何解决跨域问题

@Configuration

public class CorsConfig implements **WebMvcConfigurer** {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowCredentials(true)

.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.maxAge(3600);

}

}

项目中前后端分离部署，所以需要解决跨域的问题。

我们知道一个http请求，先走filter，到达servlet后才进行拦截器的处理，如果我们把cors放在filter里，就可以优先于权限拦截器执行。

@Configuration

public class CorsConfig {

@Bean

public CorsFilter corsFilter() {

CorsConfiguration corsConfiguration = new CorsConfiguration();

corsConfiguration.addAllowedOrigin("\*");

corsConfiguration.addAllowedHeader("\*");

corsConfiguration.addAllowedMethod("\*");

corsConfiguration.setAllowCredentials(true);

UrlBasedCorsConfigurationSource urlBasedCorsConfigurationSource = new UrlBasedCorsConfigurationSource();

urlBasedCorsConfigurationSource.registerCorsConfiguration("/\*\*", corsConfiguration);

return new CorsFilter(urlBasedCorsConfigurationSource);

}

}

## 2.18、什么是CSRF攻击

**CSRF 代表跨站请求伪造**。这是一种攻击，迫使最终用户在当前通过身份验证的Web 应用程序上执行不需要的操作。CSRF 攻击专门针对状态改变请求，而不是数据窃取，因为攻击者无法查看对伪造请求的响应。

## 2.19、Spring Boot的核心注解是哪些？他主由哪几个注解组成的？

启动类上面的注解是@SpringBootApplication，他也是SpringBoot的核心注解，主要组合包含了以下3个注解：

@SpringBootConfiguration：组合了@Configuration注解，实现配置文件的功能；

@EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置的功能：@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})；

@ComponentScan：Spring组件扫描。

## 2.20、SpringBoot的核心配置文件？区别

SpringBoot的核心配置文件是**application和bootstrap**配置文件。

application配置文件这个容易理解，主要用于Spring Boot项目的自动化配置。

bootstrap配置文件有以下几个应用场景：

使用Spring Cloud Config配置中心时，这时需要在bootstrap配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；

一些固定的不能被覆盖的属性；

一些加密/解密的场景

## 2.21、SpringBoot有哪几种读取配置的方式

Spring Boot 可以通过 @PropertySource,@Value,@Environment, @ConfigurationProperties 来绑定变量。

## 2.22、Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

Spring Boot 支持 Java Util Logging, Log4j2, Lockback 作为日志框架，如果你使用Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架。

2.23、保护SpringBoot应用的方法

在生产中使用HTTPS

使用Snyk检查你的依赖关系

升级到最新版本

启用CSRF保护

使用内容安全策略防止XSS攻击

# 3、Spring MVC

## 3.1、Springmvc 执行流程



用户发送请求至前端控制器DispatcherServlet；

DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handle；

处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet；

DispatcherServlet 调用 HandlerAdapter处理器适配器；

HandlerAdapter 经过适配调用 具体处理器(Handler，也叫后端控制器)；

Handler执行完成返回ModelAndView；

HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；

DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；

ViewResolver解析后返回具体View；

DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）

DispatcherServlet响应用户

# Quartz

## 4.1、核心概念

**任务Job**

Job就是你想要实现的任务类，每一个Job必须实现org.quartz.job接口，且只需实现接口定义的execute()方法。

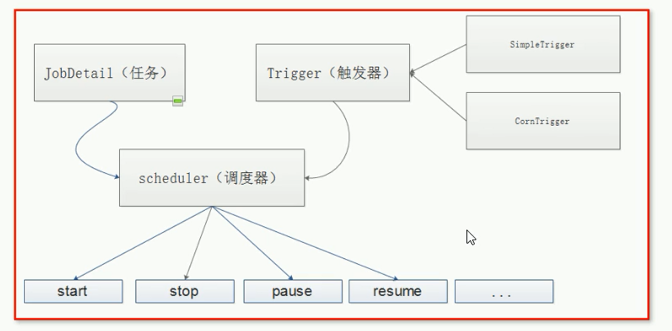
**触发器Trigger**

Trigger为你执行任务的触发器，比如你想每天定时3点发送一份统计邮件，Trigger将会设置3点执行该任务。 + Trigger主要包含两种SimplerTrigger和CronTrigger两种。

**调度器Scheduler**

Scheduler为任务的调度器，它会将任务Job及触发器Trigger整合起来，负责基于Trigger设定的时间来执行Job。

## 4.2、Quartz的体系结构



# Shiro

## 5.1、Shiro的核心概念

**Subject、SecurityManager、Realm**

Subject：主体，代表了当前“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject，如爬虫、机器人等；即一个抽象概念；所有Subject都绑定到SecurityManager，与Subject的所有交互都会委托给SecurityManager；可以把Subject认为是一个门面；**SecurityManager才是实际的执行者**。

SecurityManager：安全管理器；即所有与安全有关的操作都会与SecurityManager交互；且它管理着所有Subject；可以看出它是shiro的核心, SecurityManager相当于spring mvc中的dispatcherServlet前端控制器。

**Realm：域**，shiro从**Realm获取安全数据(如用户、角色、权限)**，就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成

# RabbitMQ

# Dubbo

## 7.1、dubbo是什么

dubbo是一个分布式框架，**远程服务调用的分布式框架**，其核心部分包含：

集群容错：提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。

远程通讯：提供对多种**基于长连接的NIO框架抽象封装**，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。

自动发现：**基于注册中心目录服务**，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

## 7.2、dubbo能做什么

透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。 软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

## 7.3、问题

### 7.3.1、默认使用的是什么通信框架，还有别的选择吗

答：默认也推荐使用 netty 框架，还有 mina。

### 7.3.2、服务调用是阻塞的

答：默认是阻塞的，可以异步调用，没有返回值的可以这么做。

### 7.3.3、一般使用什么注册中心？还有别的选择吗？

答：推荐使用 zookeeper 注册中心，还有 Multicast注册中心, Redis注册中心, Simple注册中心.

ZooKeeper的节点是通过像树一样的结构来进行维护的，并且每一个节点通过路径来标示以及访问。除此之外，每一个节点还拥有自身的一些信息，包括：数据、数据长度、创建时间、修改时间等等。

### 7.3.4、默认使用什么序列化框架，你知道的还有哪些？

答：默认使用 **Hessian** 序列化，还有 Duddo、FastJson、Java 自带序列化。 hessian是一个采用二进制格式传输的服务框架，相对传统soap web service，更轻量，更快速。

hessian的协议本身并不复杂，在此不再赘言；所谓协议(protocol)就是约束数据的格式，client按照协议将请求信息序列化成字节序列发送给server端，server端根据协议，将数据反序列化成“对象”，然后执行指定的方法，并将方法的返回值再次按照协议序列化成字节流，响应给client，client按照协议将字节流反序列话成”对象”。

### 7.3.5、服务提供者能实现失效踢出是什么原理？

答：服务失效踢出基于 zookeeper 的临时节点原理。

### 7.3.6、服务上线怎么不影响旧版本？

答：采用多版本开发，不影响旧版本。在配置中添加version来作为版本区分

### 7.3.7、如何解决服务调用链过长的问题？

答：可以结合 zipkin 实现分布式服务追踪。

### 7.3.8、说说核心的配置有哪些？

核心配置有

dubbo:service/

dubbo:reference/

dubbo:protocol/

dubbo:registry/

dubbo:application/

dubbo:provider/

dubbo:consumer/

dubbo:method/

### 7.3.9、dubbo 推荐用什么协议？

答：默认使用 dubbo 协议。

### 7.3.10、同一个服务多个注册的情况下可以直连某一个服务吗？

答：可以直连，修改配置即可，也可以通过 telnet 直接某个服务。

### 7.3.11、dubbo 在安全机制方面如何解决的？

dubbo 通过 token 令牌防止用户绕过注册中心直连，然后在注册中心管理授权，dubbo 提供了黑白名单，控制服务所允许的调用方。

### 7.3.12、集群容错怎么做？

答：读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

### 13、在使用过程中都遇到了些什么问题？ 如何解决的？

1. 同时配置了 XML 和 properties 文件，则 properties 中的配置无效

只有 XML 没有配置时，properties 才生效。

2.dubbo 缺省会在启动时检查依赖是否可用，不可用就抛出异常，阻止 spring 初始化完成，check 属性默认为 true。

测试时有些服务不关心或者出现了循环依赖，将 check 设置为 false

3. 为了方便开发测试，线下有一个所有服务可用的注册中心，这时，如果有一个正在开发中的服务提供者注册，可能会影响消费者不能正常运行。

解决：让服务提供者开发方，只订阅服务，而不注册正在开发的服务，通过直连测试正在开发的服务。设置 dubbo:registry 标签的 register 属性为 false。

4.spring 2.x 初始化死锁问题。

在 spring 解析到 dubbo:service 时，就已经向外暴露了服务，而 spring 还在接着初始化其他 bean，如果这时有请求进来，并且服务的实现类里有调用 applicationContext.getBean() 的用法。getBean 线程和 spring 初始化线程的锁的顺序不一样，导致了线程死锁，不能提供服务，启动不了。

解决：不要在服务的实现类中使用 applicationContext.getBean(); 如果不想依赖配置顺序，可以将 dubbo:provider 的 deplay 属性设置为 - 1，使 dubbo 在容器初始化完成后再暴露服务。

5. 服务注册不上

检查 dubbo 的 jar 包有没有在 classpath 中，以及有没有重复的 jar 包

检查暴露服务的 spring 配置有没有加载

在服务提供者机器上测试与注册中心的网络是否通

6. 出现 RpcException: No provider available for remote service 异常

表示没有可用的服务提供者，

1). 检查连接的注册中心是否正确

2). 到注册中心查看相应的服务提供者是否存在

3). 检查服务提供者是否正常运行

7. 出现” 消息发送失败” 异常

通常是接口方法的传入传出参数未实现 Serializable 接口。

14、dubbo 和 dubbox 之间的区别？

答：dubbox 是当当网基于 dubbo 上做了一些扩展，如加了服务可 restful 调用，更新了开源组件等。

15、你还了解别的分布式框架吗？

答：别的还有 spring 的 spring cloud

16、Dubbo 支持哪些协议，每种协议的应用场景，优缺点？

**dubbo：单一长连接和 NIO 异步通讯，适合大并发小数据量的服务调用，以及消费者远大于提供者。传输协议 TCP，异步，Hessian 序列化；**

rmi：采用 JDK 标准的 rmi 协议实现，传输参数和返回参数对象需要实现 Serializable 接口，使用 java 标准序列化机制，使用阻塞式短连接，传输数据包大小混合，消费者和提供者个数差不多，可传文件，传输协议 TCP。 多个短连接，TCP 协议传输，同步传输，适用常规的远程服务调用和 rmi 互操作。在依赖低版本的 Common-Collections 包，java 序列化存在安全漏洞；

webservice:基于 WebService 的远程调用协议，集成 CXF 实现，提供和原生 WebService 的互操作。多个短连接，基于 HTTP 传输，同步传输，适用系统集成和跨语言调用；http： 基于 Http 表单提交的远程调用协议，使用 Spring 的 HttpInvoke 实现。多个短连接，传输协议 HTTP，传入参数大小混合，提供者个数多于消费者，需要给应用程序和浏览器 JS 调用； hessian： 集成 Hessian 服务，基于 HTTP 通讯，采用 Servlet 暴露服务，Dubbo 内嵌 Jetty 作为服务器时默认实现，提供与 Hession 服务互操作。多个短连接，同步 HTTP 传输，Hessian 序列化，传入参数较大，提供者大于消费者，提供者压力较大，可传文件；

memcache： 基于 memcached 实现的 RPC 协议 redis： 基于 redis 实现的 RPC 协议

17、Dubbo 集群的负载均衡有哪些策略

Dubbo 提供了常见的集群策略实现，并预扩展点予以自行实现。

Random LoadBalance: 随机选取提供者策略，有利于动态调整提供者权重。截面碰撞率高，调用次数越多，分布越均匀；

RoundRobin LoadBalance: 轮循选取提供者策略，平均分布，但是存在请求累积的问题；

LeastActive LoadBalance: 最少活跃调用策略，解决慢提供者接收更少的请求； ConstantHash LoadBalance: 一致性 Hash 策略，使相同参数请求总是发到同一提供者，一台机器宕机，可以基于虚拟节点，分摊至其他提供者，避免引起提供者的剧烈变动；

18. 服务调用超时问题怎么解决

dubbo在调用服务不成功时，默认是会重试两次的。这样在服务端的处理时间超过了设定的超时时间时，就会有重复请求，比如在发邮件时，可能就会发出多份重复邮件，执行注册请求时，就会插入多条重复的注册数据，那么怎么解决超时问题呢？如下

对于核心的服务中心，去除dubbo超时重试机制，并重新评估设置超时时间。 业务处理代码必须放在服务端，客户端只做参数验证和服务调用，不涉及业务流程处理 全局配置实例

<dubbo:provider delay="-1" timeout="6000" retries="0"/>

当然Dubbo的重试机制其实是非常好的QOS保证，它的路由机制，是会帮你把超时的请求路由到其他机器上，而不是本机尝试，所以 dubbo的重试机器也能一定程度的保证服务的质量。但是请一定要综合线上的访问情况，给出综合的评估。