2019
版

治愈系 Java 工程师面试指导课程

[Part9・Dubbo 等专题]

[本教程梳理了 Dubbo、Nginx、FastDfs 和 Freemarker 部分的复习脉络，并且涵盖了这几部分的高频面试题；建议大家将目录结构打开，对照目录结构做一个复习的、提纲准备面试。]

1 Dubbo 重要的概念

1.1 什么是 Dubbo?

Apache Dubbo (incubating) |ˈdʌbəʊ| 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**。简单来说 Dubbo 是一个**分布式服务框架**，致力于提供高性能和透明化的 **RPC** 远程服务调用方案，以及 **SOA** 服务治理方案。

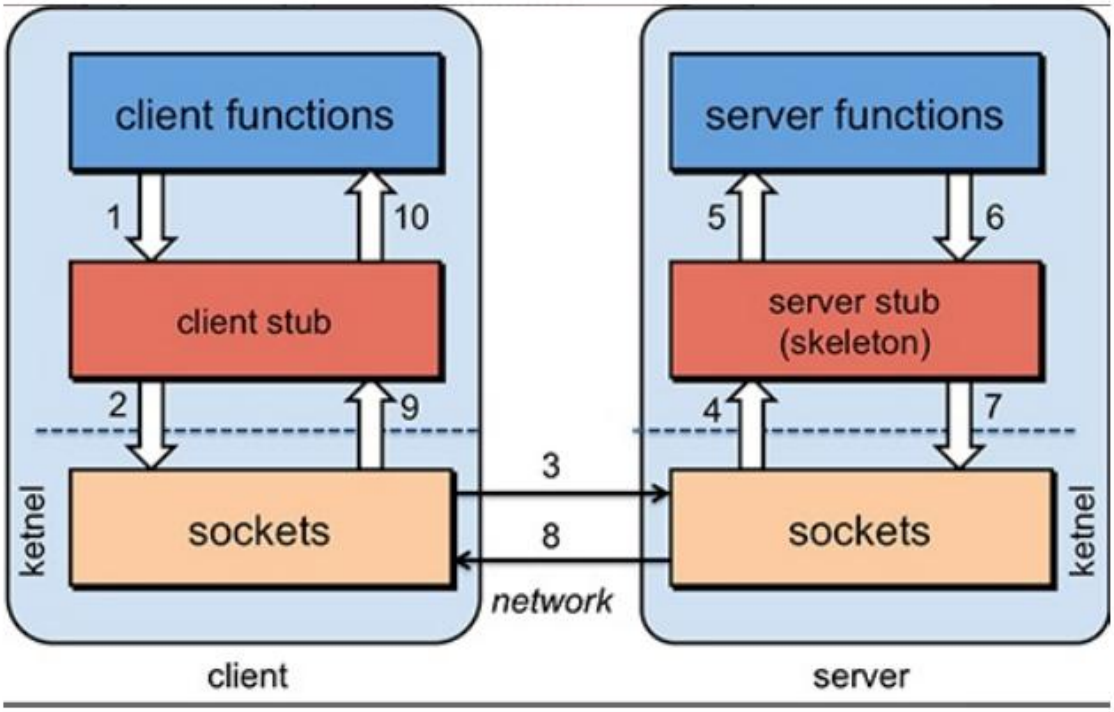
Dubbo 是由阿里开源，后来加入了 Apache 。正式由于 Dubbo 的出现，才使得越来越多的公司开始使用以及接受分布式架构。

Dubbox 当当网基于 dubbo 上做了一些扩展，如加了服务可 restful 调用、跨语言、更新了开源组件等。

1.2 什么是 RPC?RPC 原理【了解】

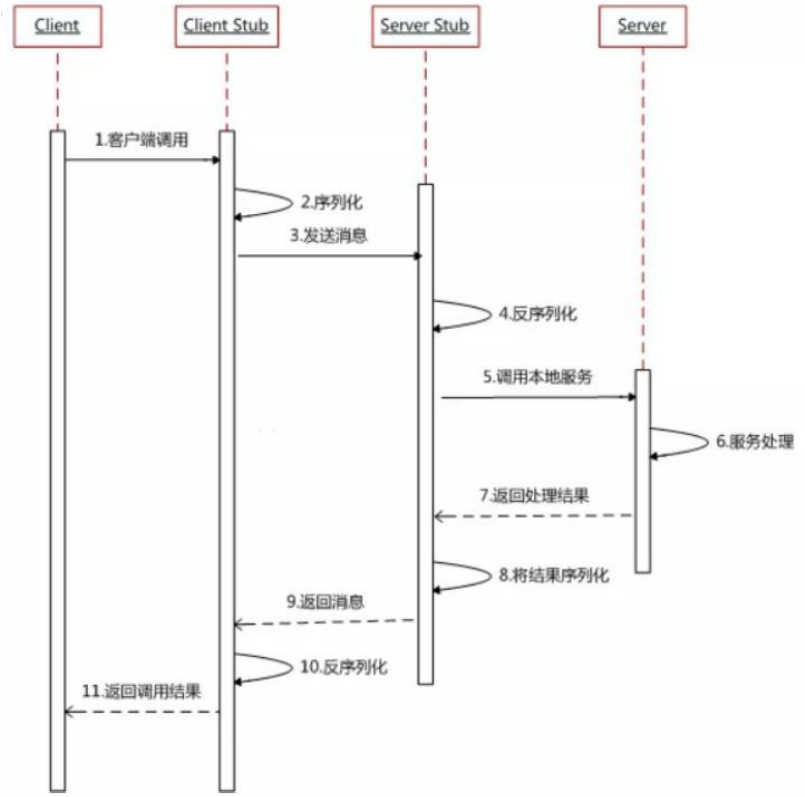
RPC（Remote Procedure Call）—远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。比如两个不同的服务 A,B 部署在两台不同的机器上，那么服务 A 如果想要调用服务 B 中的某个方法该怎么办呢？使用 HTTP 请求 当然可以，但是可能会比较慢而且一些优化做的并不好。 RPC 的出现就是为了解决这个问题。

这里简单的提一下 RPC 原理



1. 服务消费方（client）调用以本地调用方式调用服务；
2. client stub 接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；
3. client stub 找到服务地址，并将消息发送到服务端；
4. server stub 收到消息后进行解码；
5. server stub 根据解码结果调用本地的服务；

6. 本地服务执行并将结果返回给 server stub；
7. server stub 将返回结果打包成消息并发送至消费方；
8. client stub 接收到消息，并进行解码；
9. 服务消费方得到最终结果。



1.3 Dubbo 所支持的通信协议

Dubbo 支持 dubbo、rmi、hessian、http、webservice、thrift、redis 等多种协议，但是 Dubbo 官网是推荐我们使用 Dubbo 协议的。

1.3.1 支持的协议

协议名称	实现描述	连接	使用场景
dubbo	传输：mina、netty、grizzy 序列化：dubbo、hessian2、java、json	dubbo缺省采用单一长连接和NIO异步通讯	1.传入传出参数数据包较小 2.消费者 比提供者多 3.常规远程服务方法调用 4.不适合传送大数据量的服务，比如文件、传视频
rmi	传输：java rmi 序列化：java 标准序列化	连接个数：多连接 连接方式：短连接 传输协议：TCP/IP 传输方式：BIO	1.常规RPC调用 2.与原RMI客户端互操作 3.可传文件 4.不支持防火墙穿透
hessian	传输：Serverlet容器 序列化：hessian二进制序列化	连接个数：多连接 连接方式：短连接 传输协议：HTTP 传输方式：同步传输	1.提供者比消费者多 2.可传文件 3.跨语言传输
http	传输：servlet容器 序列化：表单序列化	连接个数：多连接 连接方式：短连接 传输协议：HTTP 传输方式：同步传输	1.提供者多余消费者 2.数据包混合
webservice	传输：HTTP 序列化：SOAP文件序列化	连接个数：多连接 连接方式：短连接 传输协议：HTTP 传输方式：同步传输	1.系统集成 2.跨语言调用
thrift	与thrift RPC实现集成，并在基础上修改了报文头	长连接、NIO异步传输	

1.3.2 协议的配置

<dubbo:protocal>（只需在服务提供方配置即可）

属性	类型	是否必填	缺省值	描述
name	string	必填	dubbo	协议名称
port	int	可选	dubbo协议缺省端口为20880, rmi协议缺省端口为1099， http和hessian协议缺省端口为80； 如果配置为-1或者没有配置port,则 会分配一个没有被占用的端口。	服务端口
threadpool	string	可选	fixed	线程池类型，可选：fixed/cached
threads	int	可选	100	服务h定大小)
iothreads	int	可选	CPU个数+1	io线程池大小（固定）
accepts	int	可选	0	服务提供方最大可接受连接数
serialization	string	可选	dubbo协议缺省为hessian2, rmi缺省协议为java, http协议缺省为json	可填序列化：dubbo|hessian2| java|compactedjava|fastjson
dispatcher	string	可选	dubbo协议缺省为all	协议的消息派发方式，用于指定线程模型， 比如：dubbo协议的all, direct, message, execution, connection等 参考：https://blog.csdn.net/fd2025/article/ details/79985542
queues	int	可选	0	线程池队列大小，当线程池满时， 排队等待执行的队列大小，建议不要设置， 当线程程池时应立即失败，重试其它服务 提供机器，而不是排队，除非有特殊需求。
charset	string	可选	UTF-8	序列化编码
server	server	string	dubbo协议缺省为netty， http协议缺省为servlet hessian协议缺省为jetty	协议的服务器端实现类型， 比如：dubbo协议的mina,netty等， http协议的jetty,servlet等

1.4 dubbo 中序列化的方式

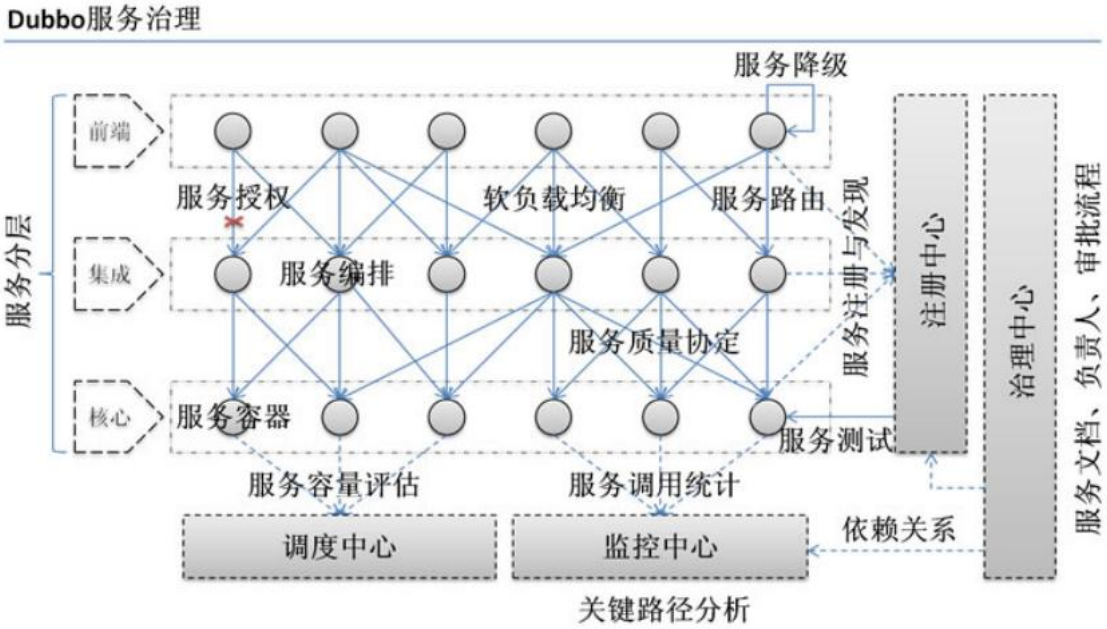
- dubbo 序列化：阿里尚未开发成熟的高效 java 序列化实现，阿里不建议在生产环境使用它
- hessian2 序列化：hessian 是一种跨语言的高效二进制序列化方式。但这里实际不是原生的 hessian2 序列化，而是阿里修改过的 hessian lite，它是 dubbo RPC 默认启用的序列化方式
- json 序列化：目前有两种实现，一种是采用的阿里的 fastjson 库，另一种是采用 dubbo 中自己实现的简单 json 库，但其实现都不是特别成熟，而且 json 这种文本序列化性能一般不如上面两种二进制序列化。
- java 序列化：主要是采用 JDK 自带的 Java 序列化实现，性能很不理想。

1.5 为什么要用 Dubbo?

Dubbo 的诞生和 SOA 分布式架构的流行有着莫大的关系。SOA 面向服务的架构（Service OrientedArchitecture），也就是把工程按照业务逻辑拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。SOA 架构中有两个主要角色：服务提供者（Provider）和服务使用者（Consumer）。

如果你要开发分布式程序，你也可以直接基于 HTTP 接口进行通信，但是为什么要用 Dubbo 呢？我觉得主要

可以从 Dubbo 提供的下面四点特性来说为什么要用 Dubbo：



1. 负载均衡——同一个服务部署在不同的机器时该调用那一台机器上的服务

2. 服务调用链路生成——随着系统的发展，服务越来越多，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。Dubbo 可以为我们解决服务之间互相是如何调用的。

3. 服务访问压力以及时长统计、资源调度和治理——基于访问压力实时管理集群容量，提高集群利用率。

4. 服务降级——某个服务挂掉之后调用备用服务

另外，**Dubbo 除了能够应用在分布式系统中，也可以应用在现在比较火的微服务系统**中。不过，由于 Spring Cloud 在微服务中应用更加广泛，所以，一般我们提 Dubbo 的话，大部分是分布式系统的情况。

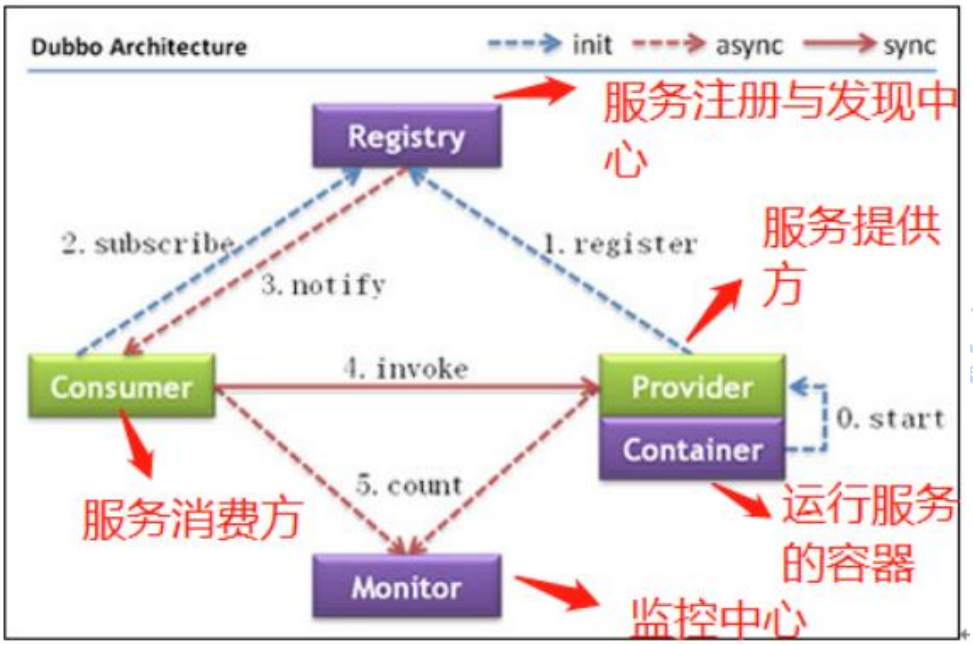
1.6 dubbo 特点

Dubbo 是一个分布式服务框架，以及 SOA 治理方案，还是一种高性能和透明化的 RPC 远程服务调用方案。其核心部分包含

- 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）
- 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。）
- 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）。
- 负载均衡

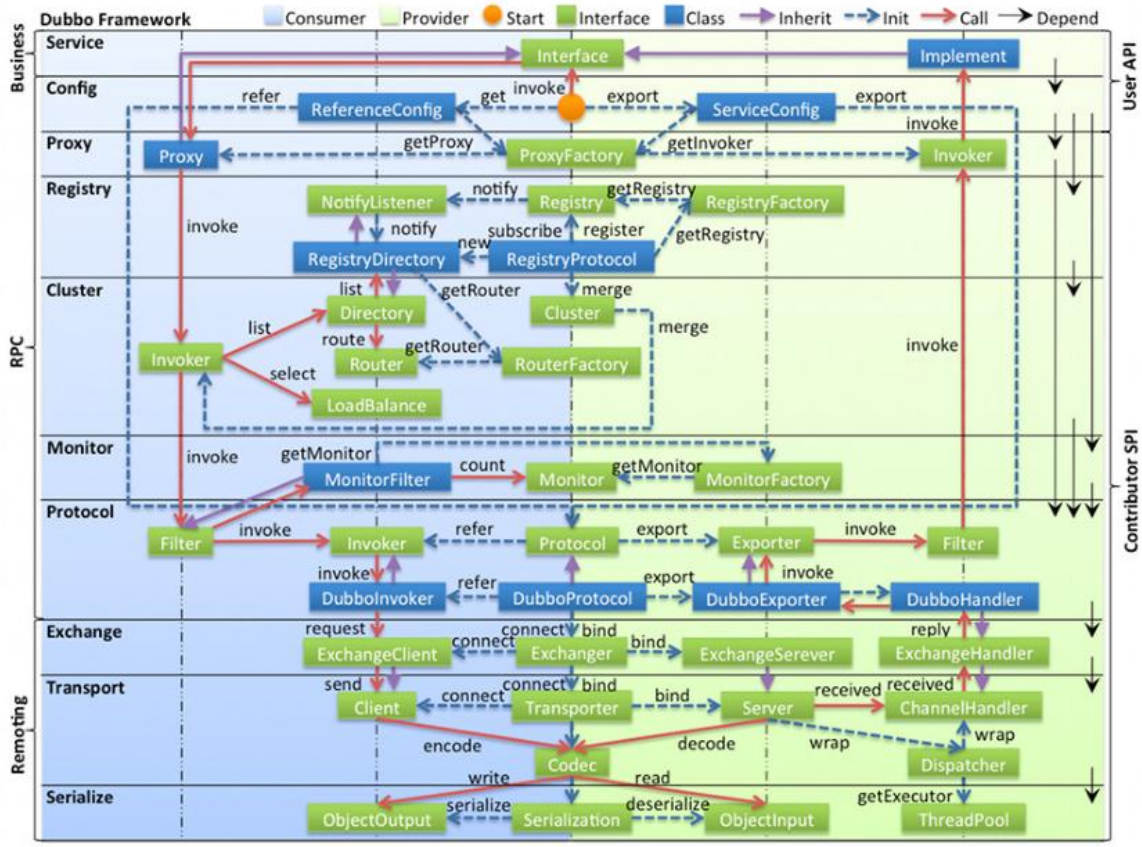
2 Dubbo 的架构

2.1 Dubbo 的架构图解



- Provider： 暴露服务的服务提供方
- Consumer： 调用远程服务的服务消费方
- Registry： 服务注册与发现的注册中心
- Monitor： 统计服务的调用次数和调用时间的监控中心
- Container： 服务运行容器
- 调用关系说明：
  - 服务容器负责启动，加载，运行服务提供者。
  - 服务提供者在启动时，向注册中心注册自己提供的服务。
  - 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，注册中心将基于长连接推送变更数据给消费者。
  - 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
  - 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

2.2 Dubbo 工作原理



图中从下至上分为十层，各层均为单向依赖，右边的黑色箭头代表层之间的依赖关系，每一层都可以剥离上层被复用，其中，Service 和 Config 层为 API，其它各层均为 SPI。

各层说明：

第一层：service 层，接口层，给服务提供者和消费者来实现的

第二层：config 层，配置层，主要是对 dubbo 进行各种配置的

第三层：proxy 层，服务接口透明代理，生成服务的客户端 Stub 和服务器端 Skeleton

第四层：registry 层，服务注册层，负责服务的注册与发现

第五层：cluster 层，集群层，封装多个服务提供者的路由以及负载均衡，将多个实例组合成一个服务

第六层：monitor 层，监控层，对 rpc 接口的调用次数和调用时间进行监控

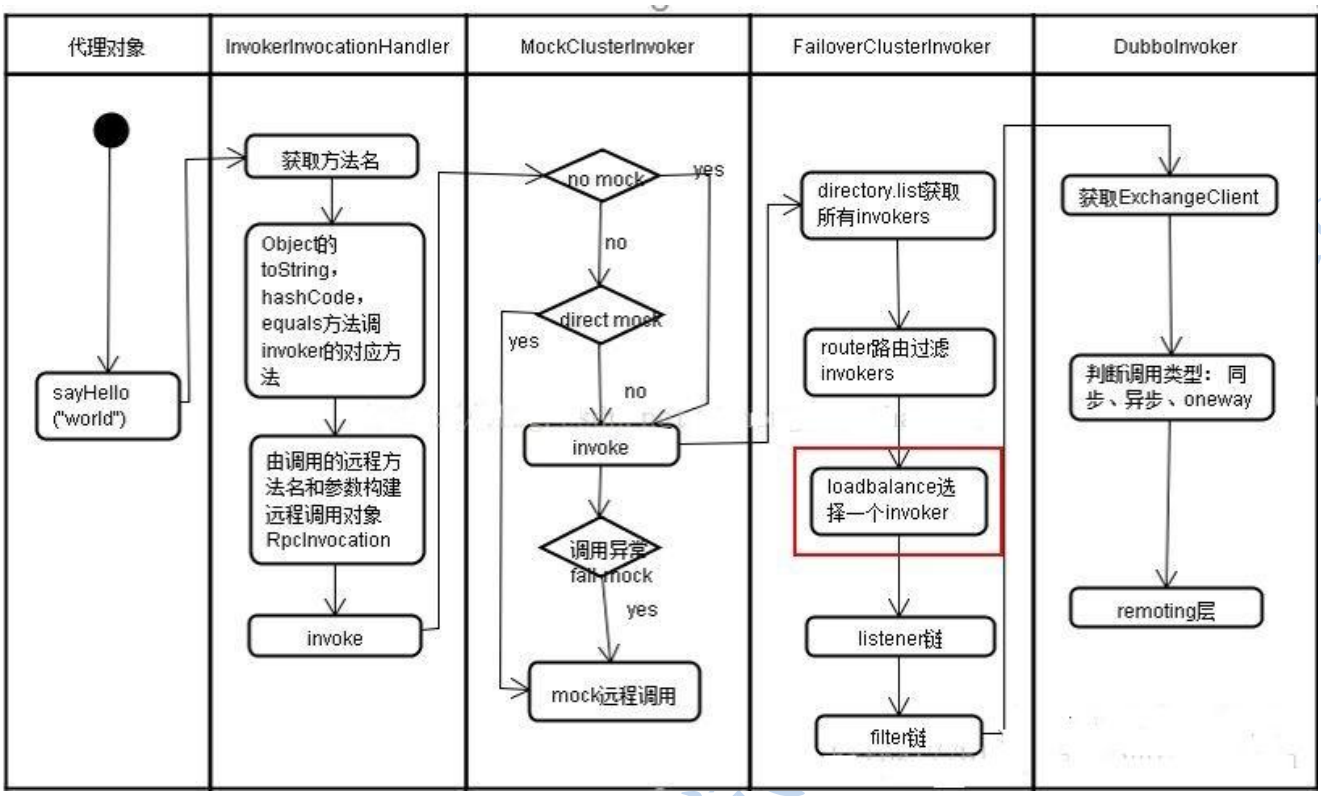
第七层：protocol 层，远程调用层，封装 rpc 调用

第八层：exchange 层，信息交换层，封装请求响应模式，同步转异步

第九层：transport 层，网络传输层，抽象 mina 和 netty 为统一接口

第十层：serialize 层，数据序列化层。网络传输需要。

2.3 Dubbo 服务调用过程



代码层自定义负载均衡策略，实现灰度发布的功能

3 Dubbo 的负载均衡策略

Dubbo 提供了多种均衡策略，默认为 random 随机调用。可以自行扩展负载均衡策略，参见：负载均衡扩展。

3.1 Random LoadBalance(默认，基于权重的随机负载均衡机制)

- 随机，按权重设置随机概率。
- 在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

3.2 RoundRobin LoadBalance(不推荐，基于权重的轮询负载均衡机制)

- 轮循，按公约后的权重设置轮循比率。
- 存在慢的提供者累积请求的问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

3.3 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。
- 使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

3.4 ConsistentHash LoadBalance

- 一致性 Hash，相同参数的请求总是发到同一提供者。(如果你需要的不是随机负载均衡，是要一类请求都到一个节点，那就走这个一致性 hash 策略。)
- 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。
- 缺省只对第一个参数 Hash，如果要修改，请配置 <dubbo:parameter key="hash.arguments" value="0,1"/>
- 缺省用 160 份虚拟节点，如果要修改，请配置 <dubbo:parameter key="hash.nodes" value="320" />

3.5 配置方式

3.5.1 xml 配置方式

- 服务端服务级别

```
<dubbo:service interface="..." loadbalance="roundrobin" />
```

- 客户端服务级别

```
<dubbo:reference interface="..." loadbalance="roundrobin" />
```

- 服务端方法级别

```
<dubbo:service interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:service>
```

- 客户端方法级别

```
<dubbo:reference interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:reference>
```

3.5.2 注解配置方式：

消费方基于基于注解的服务级别配置方式

```
@Reference(loadbalance = "roundrobin")
HelloService helloService;
```

4 Dubbo 的健壮性

4.1 dubbo 的健壮性表现

1. 监控中心宕掉不影响使用，只是丢失部分采样数据
2. 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
3. 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
4. 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
5. 服务提供者无状态，任意一台宕掉后，不影响使用
6. 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复我们前面提到过：注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小。

4.2 Zookeeper 宕机

在实际生产中，假如 zookeeper 注册中心宕掉，一段时间内服务消费方还是能够调用提供方的服务的，实际上它使用的本地缓存进行通讯，这只是 dubbo 健壮性的一种体现。

4.2.1 Zookeeper 介绍

Zookeeper 主要功能是为分布式系统提供一致性协调(Coordination)服务 ，主要有两大功能：
统一配置管理和集群负载均衡，

- 统一配置管理： 分布式系统都有好多机器，这些机器上边的配置是一致的，正常情况下需要修改好一个以后，然后 scp 到其他服务器，每次修改，就要把所有的配置都修改一遍，Zookeeper 提供了这样的一种服务：一种集中管理配置的方法，我们在这个集中的地方修改了配置，所有对这个配置感兴趣的都可以获得变更。这样就省去手动拷贝配置了，还保证了可靠和一致性。
- 集群负载均衡： 在分布式的集群中，经常会由于各种原因，比如硬件故障，软件故障，网络问题，有些节点会进进出出。有新的节点加入进来，也有老的节点退出集群。这个时候，集群中有些机器（比如 Master 节点）需要感知到这种变化，然后根据这种变化做出对应的决策。Zookeeper 会对通过选举机制，选举一个主节点作为管理者。选举一般都是奇数台，否则会选举失败。并且集群中，有半数以上宕机，则会认为整个集群挂掉。

我们项目中用到 Zookeeper 的地方主要有：solr 集群、redis 集群和 dubbo 注册中心。

4.2.2 Dubbo 直连

即在服务消费方配置服务提供方的位置信息。

- xml 配置方式：

```
<dubbo:reference id="userService" interface="com.zang.gmall.service.UserService" url="dubbo://localhost:20880" />
```

- 注解方式：

```
@Reference(url = "127.0.0.1:20880")
HelloService helloService;
```

4.2.1 Dubbo 在安全机制方面是如何解决的

如果 dubbo 支持不通过注册中心直接调用服务的提供者的服务的话，那么会有安全的问题存在。Dubbo 也考虑到了这一点，Dubbo 通过 Token 令牌防止用户绕过注册中心直连，然后在注册中心上管理授权。

服务级别 token：<dubbo:service token="" interface="">

全局级别 token：<dubbo:provide token="">

Dubbo 还提供服务黑白名单，来控制服务所允许的调用方。

4.3 服务降级 Mock

场景一、

比如说服务 A 调用服务 B，结果服务 B 挂掉了，服务 A 重试几次调用服务 B，还是不行，直接降级，走一个备用的逻辑，给用户返回响应

场景二、

在开发自测，联调过程中，经常碰到一些下游服务调用不通的场景，这个时候我们如何不依赖于下游系统，就业务系统独立完成自测？

dubbo 自身是支持 mock 服务的，在 reference 标签里，有一个参数 mock，该参数有四个值，false,default,true,或者 Mock 类的类名。分别代表如下含义:

- false，不调用 mock 服务。
- true，当服务调用失败时，使用 mock 服务。
- default，当服务调用失败时，使用 mock 服务。
- force，强制使用 Mock 服务(不管服务能否调用成功)。(使用 xml 配置不生效,使用 ReferenceConfigAPI 可以生效)
- 使用方法:将 mock 参数启用，在<dubbo:reference>中添加参数项 mock=true。实现需要调用的服务接口上游系统需要调用下游系统，则下游需要提供 jar 包给上有系统，该 jar 包只有接口，没有实现。我们需要实现该接口，且命名必须是该接口名+Mock，例如原接口是 com.alibaba.dubbo.demo.DemoService,则实现类必须是 com.alibaba.dubbo.demo.DemoServiceMock。

4.4 Dubbo 的集群容错方案

- Failover Cluster：失败自动切换，当出现失败，重试其它服务器。通常用于读操作，但重试会带来更长延迟。（默认）
- Failfast Cluster：快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作，比如新增记录。
- Failsafe Cluster：失败安全，出现异常时，直接忽略。通常用于写入审计日志等操作。
- Failback Cluster：失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作。
- Forking Cluster：并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多服务资源。可通过 forks="2" 来设置最大并行数。
- Broadcast Cluster：广播调用所有提供者，逐个调用，任意一台报错则报错 。通常用于通知所有提供者更新缓存或日志等本地资源信息。

5 Dubbo 使用过程中的优化

5.1 超时问题

Dubbo 消费端设置超时时间需要根据业务实际情况来设定，默认的超时时间是 1 秒，如果设置的时间太短，一些复杂业务需要很长时间完成，导致在设定的超时时间内无法完成正常的业务处理。这样消费端达到超时时间，那么 dubbo 会进行重试机制，不合理的重试在一些特殊的业务场景下可能会引发很多问题，需要合理设置接口超时时间。比如发送邮件，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。

我们可以通过两种方式设置超时时间

- 服务提供者端设置超时时间，在 Dubbo 的用户文档中，推荐如果能在服务端多配置就尽量多配置，因为服务提供者比消费者更清楚自己提供的服务特性。
- 服务消费者端设置超时时间，如果在消费者端设置了超时时间，以消费者端为主，即优先级更高。因为服务调用方设置超时时间控制性更灵活。如果消费方超时，服务端线程会定制，会产生警告。

dubbo 在调用服务不成功时，默认是会重试两次。

5.2 dubbo 中对服务多版本的支持（详见灰度发布）

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。
在低压力时间段，先升级一半提供者为新版本。再将所有消费者升级为新版本，然后将剩下的一半提供者升级为新版本

5.3 Dubbo 和 SpringCloud 的区别和联系

Dubbo 是 SOA 时代的产物，它的关注点主要在于服务的调用，流量分发、流量监控和熔断。而 Spring Cloud 诞生于微服务架构时代，考虑的是微服务治理的方方面面，另外由于依托了 Spirng、Spirng Boot 的优势之上，两个框架在开始目标就不一致，Dubbo 定位服务治理、Spirng Cloud 是一个生态。

最大的区别：Dubbo 底层是使用 Netty 这样的 NIO 框架，是基于 TCP 协议传输的，配合以 Hession 序列化完成 RPC 通信。而 SpringCloud 是基于 Htt 协议+Rest 接口调用远程过程的通信，相对来说，Http 请求会有更大的报文，占的带宽也会更多。但是 REST 相比 RPC 更为灵活，服务提供方和调用方的依赖只依靠一纸契约，不存在代码级别的强依赖。

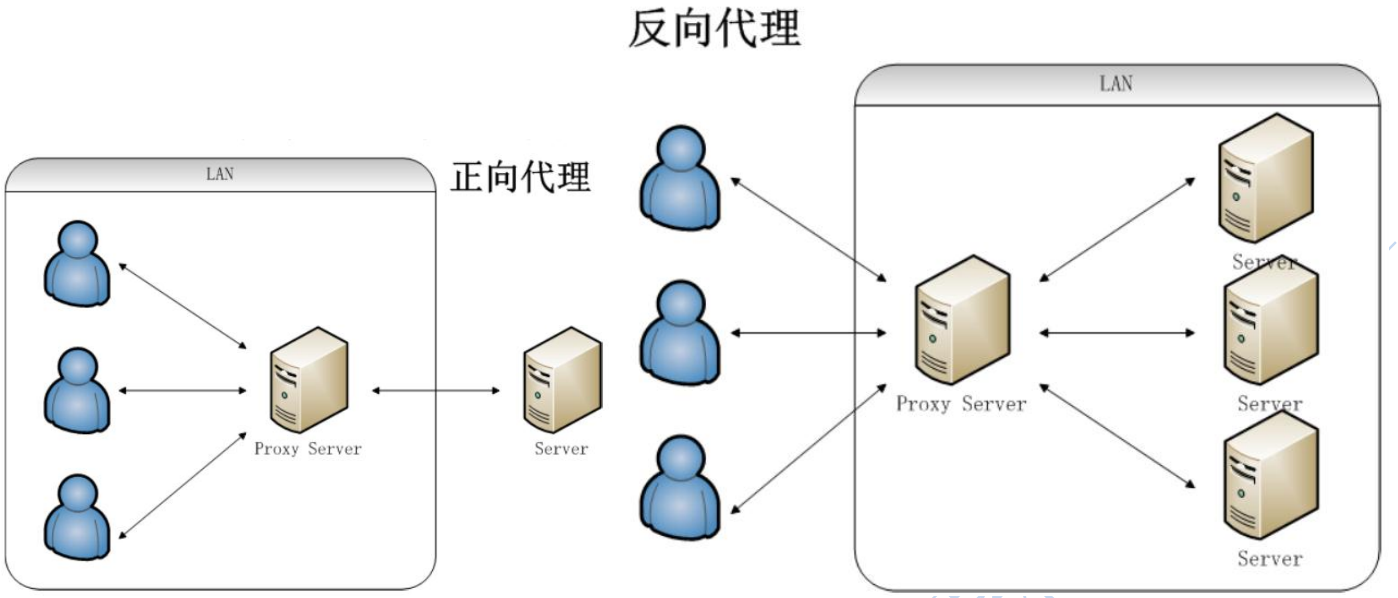
6 Nginx

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器。 Nginx 主要提供反向代理、负载均衡、动静分离(静态资源服务)等服务。

6.1 正向代理和反向代理

谈到反向代理，就不得不提一下正向代理。无论是正向代理，还是反向代理，说到底，就是代理模式的衍生版本罢

了



- 正向代理：某些情况下，代理我们用户去访问服务器，需要用户手动的设置代理服务器的 ip 和端口号。正向代理比较常见的一个例子就是 VPN 了。
- 反向代理： 是用来代理服务器的，代理我们要访问的目标服务器。代理服务器接受请求，然后将请求转发给内部网络的服务器，并将从服务器上得到的结果返回给客户端，此时代理服务器对外就表现为一个服务器。
- 所以，简单的理解，就是正向代理是为客户端做代理，代替客户端去访问服务器，而反向代理是为服务器做代理，代替服务器接受客户端请求。

6.2 负载均衡

在高并发情况下需要使用，其原理就是将并发请求分摊到多个服务器执行，减轻每台服务器的压力，多台服务器(集群)共同完成工作任务，从而提高了数据的吞吐量。

Nginx 支持的 weight 轮询（默认）、ip_hash、fair、url_hash 这四种负载均衡调度算法，感兴趣的可以自行查阅。

负载均衡相比于反向代理更侧重的时将请求分担到多台服务器上去，所以谈论负载均衡只有在提供某服务的服务器大于两台时才有意义。

6.3 动静分离

动静分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路。

6.4 为什么要用 Nginx?

这部分内容参考极客时间—Nginx 核心知识 100 讲的内容。

如果面试官问你这个问题，就一定想看你知道 Nginx 服务器的一些优点吗。

6.4.1 Nginx 有以下 5 个优点

1. 高并发、高性能（这是其他 web 服务器不具有的）
2. 可扩展性好（模块化设计，第三方插件生态圈丰富）
3. 高可靠性（可以在服务器行持续不间断的运行数年）
4. 热部署（这个功能对于 Nginx 来说特别重要，热部署指可以在不停止 Nginx 服务的情况下升级 Nginx）
5. BSD 许可证（意味着我们可以将源代码下载下来进行修改然后使用自己的版本）

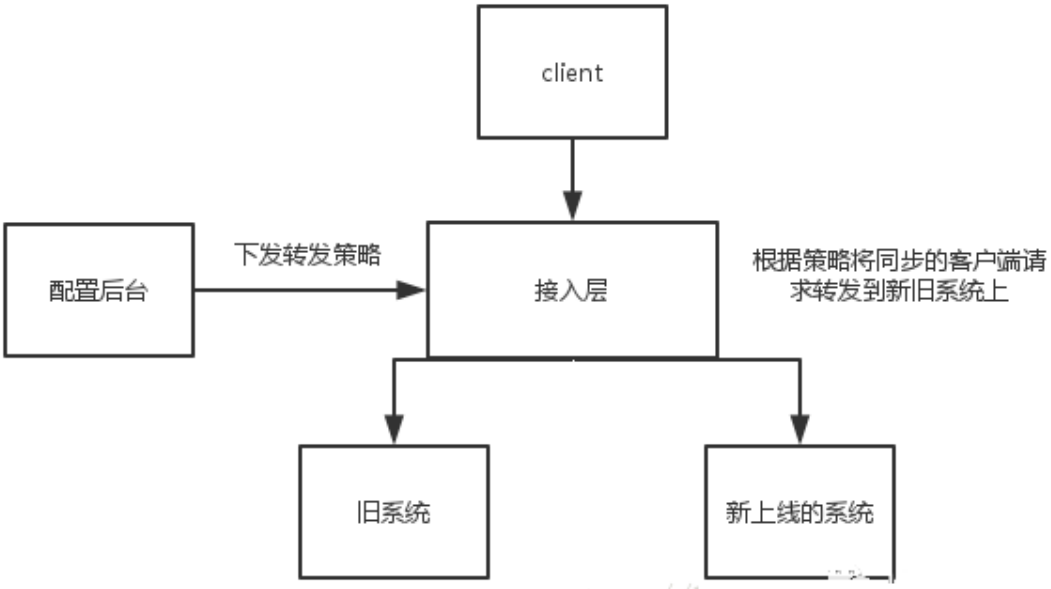
6.4.2 Nginx 的四个主要组成部分了解吗?

- Nginx 二进制可执行文件：由各模块源码编译出一个文件
- Nginx.conf 配置文件：控制 Nginx 行为
- acess.log 访问日志： 记录每一条 HTTP 请求信息
- error.log 错误日志:定位问题

7 灰度发布方案

7.1 灰度发布定义

灰度发布（又名金丝雀发布）是指在黑与白之间，能够平滑过渡的一种发布方式。在其上可以进行 A/B testing，即让一部分用户继续用产品特性 A，一部分用户开始用产品特性 B，如果用户对 B 没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到 B 上面来。灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。



7.2 实现思路方向

方案	特点	优点	缺点
在代码中做	一套线上环境，代码中做开关，对于不同的用户走不同的逻辑	灵活，粒度细；一套代码(环境)运维成本低	灰度逻辑侵入代码
在接入层做 第一是在 nginx 层实现（使用 ngx+lua）， 第二是在网关层实现（spring-cloud-zuul）。 第三是 dubbo 的灰度	多套(隔离的)线上环境，接入层针对不同用户转发到不同的环境中	无需（少）侵入代码；风险小	多套线上环境，运维成本高

可以看到我们要做灰度发布的主要诉求是保证线上的质量，尽量降低因迭代带来的服务问题。
而非要针对于不同的用户做 AB Test。考虑到如果灰度的方案涉及到修改代码，则可能引入其他不确定的风险，在此，我们采用第二种，也就是在接入层做分流的策略。
灵活的灰度方案一般需要在接入层实现，具体就是自定义负载均衡策略实现。

7.3 dubbo 灰度方案说明

7.3.1 服务分组

当一个接口有多种实现时，可以用 group 区分。

1. 提供者配置

```
<bean id="demoService" class="com.providerimpl.DemoServiceImpl" />
<bean id="demoServiceGroup" class="comproviderimpl.DemoServiceGroupImpl" />
<dubbo:service group="default" interface="com.service.DemoService"
               ref="demoService" protocol="dubbo"/>
<dubbo:service group="group" interface="com. service.DemoService"
               ref="demoServiceGroup" protocol="rmi"/>
```

2. 消费者配置

```
<dubbo:reference id="demoService" interface="com.service.DemoService"
                 loadbalance="leastactive" group="group"/>
```

7.3.2 多版本

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。可以按照以下的步骤进行版本迁移：

- 在低压力时间段，先升级一半提供者为新版本
- 再将所有消费者升级为新版本
- 然后将剩下的一半提供者升级为新版本

1. 提供者配置

```
<dubbo:service interface="com.service.DemoService"
               ref="demoService" protocol="dubbo" version="1.0.0"/>
<dubbo:service interface="com.service.DemoService"
               ref="demoServiceGroup" protocol="dubbo" version="2.0.0"/>
```

2. 消费者配置

```
<dubbo:reference id="demoService2" interface="com.service.DemoService"
                 check="false" version="2.0.0"/>
```

不区分版本：(2.2.0 以上版本支持)

```
<dubbo:reference id="barService" interface="com.service.BarService" version="*" />
```

8 freemarker

freemarker 是一个模板化引擎语言，传统项目中，后台页面基本上都是 list 列表，页面都是类似的，只是展示的数据不一样。这样我们就可以把他的样式做成一个 freemarker 模板，然后传入数据就可以展示不同的页面。他的模板一般是以.ftl 结尾的。如果使用这个呢，可以使我们开发人员不需要太关注前台。但是对于很多门户或商城类项目，每个页面都是不一样，所以用他也不是很方便。另外，由于可以把 freemarker 模板直接转换成 html、jsp、java、xml、word 等各种文档。所以我们经常使用他做代码生成、word 生成或者首页静态化等。我这里就用到了首页静态化的功能。由于任何用户访问时，首先会访问到我们的首页，所以很多东西都希望能在首页展示，但是放的东西多了，就会加载很慢。给用户的体验度很不好。所以我们在项目启动时，直接把首页需要的数据查询出来，通过 freemarker 模板生成静态的 html 页面，之后用户访问时，都去访问这个静态页面，这样就不需要频繁访问数据库，减轻了数据库压力，提高了用户体验度。但是缺点是，数据库数据数据变换了以后，数据无法实时更新。我们一般通过定时器的方式，每天凌晨重新生成。

还有就是热销商品的商品详情页面也做了静态化处理，首页我们是通过定时器每天凌晨一点，去重新生成；商品详情我们是在商品信息修改以后，给定时器表（id,业务 id、type，remark）中推送一条信息，到第二天凌晨一点时，定时任务扫描表，发现有需要重新生成的页面，就调用 freemarker 模板，重新生成 html 页面，以商品 id 作为 html 页面名称，然后删除任务表中的数据。为了预防大量静态页面给服务器造成压力，我们把 html 页面直接生成到 Nginx 的静态页面服务器上。访问时，不用经过 Tomcat，直接通过 Nginx 访问。

和他类似的技术，我还知道一种 volicity，但是这个技术现在基本上用的很少了，主要是他对 jsp 的兼容不

是很好。

freemarker 主要语法用${}输出变量，$#等要用原样输出符${r"#"}，空值会抛异常，<#list 做循环，循环下标用变量_index

${book.name?if_exists } //用于判断如果存在,就输出这个值

${book.name?default(‘xxx’)}//默认值 xxx

${book.name!"xxx"}//默认值 xxx

${book.date?string('yyyy-MM-dd')} //日期格式

${book?string.number} 20 //三种不同的数字格式

${book?string.currency}--<#-- $20.00 -->

${book?string.percent}—<#-- 20% -->

用 lt、lte、gt 和 gte 来替代<、<=、>和>=

如果存在特殊字符,可以使用${r"..."}进行过滤

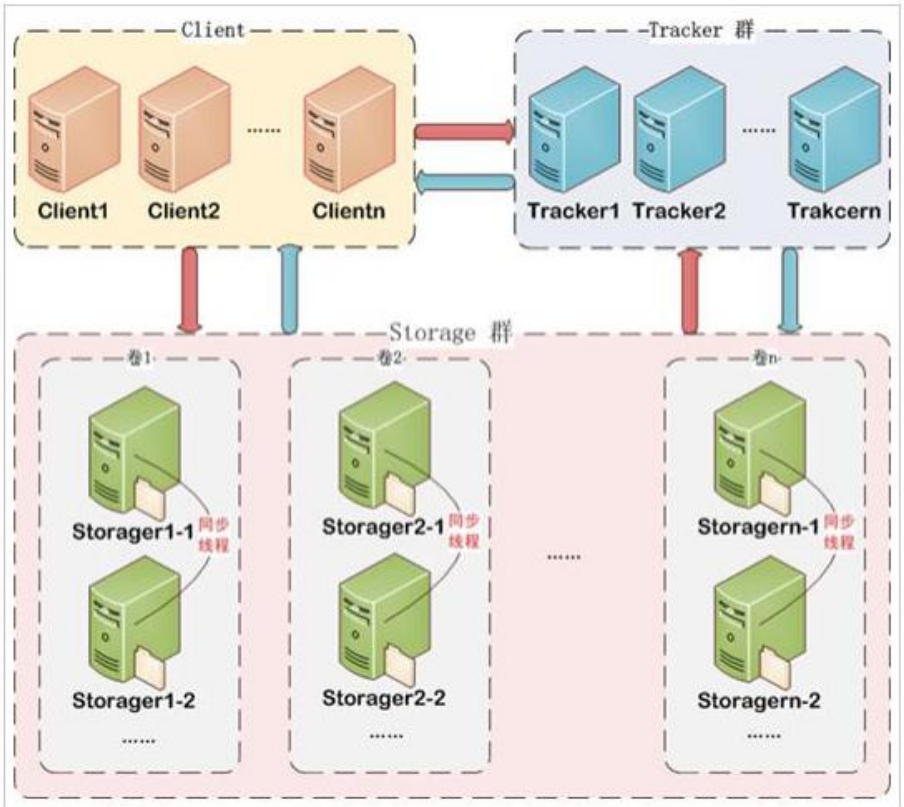
9 FastDFS

FastDFS 是用 c 语言编写的一款开源的分布式文件系统。FastDFS 为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用 FastDFS 很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 进行文件上传、下载，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。

Tracker server 作用是负载均衡和调度，通过 Tracker server 在文件上传时可以根据一些策略找到 Storage server 提供文件上传服务。可以将 tracker 称为追踪服务器或调度服务器。

Storage server 作用是文件存储，客户端上传的文件最终存储在 Storage 服务器上，Storageserver 没有实现自己的文件系统而是利用操作系统 的文件系统来管理文件。可以将 storage 称为存储服务器。

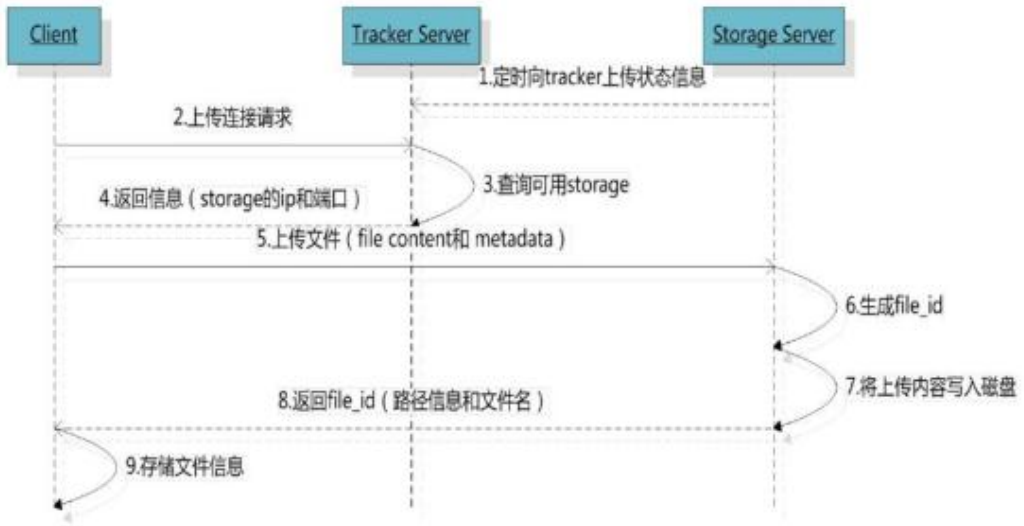


服务端两个角色：

Tracker：管理集群，tracker 也可以实现集群。每个 tracker 节点地位平等。收集 Storage 集群的状态。

Storage：实际保存文件 Storage 分为多个组，每个组之间保存的文件是不同的。每个组内部可以有多个成员，组成员内部保存的内容是一样的，组成员的地位是一致的，没有主从的概念。

9.1 文件上传流程



客户端上传文件后存储服务器将文件 ID 返回给客户端，此文件 ID 用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

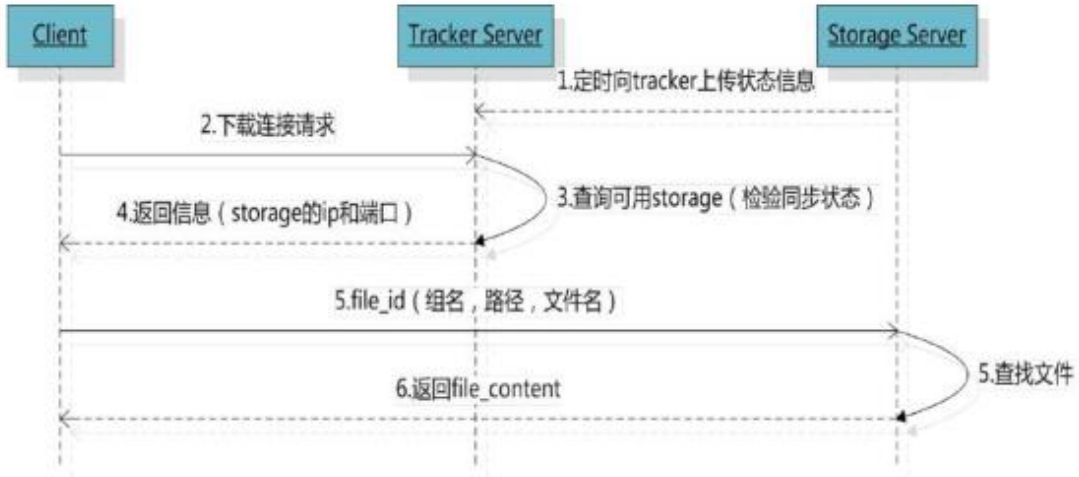
组名：文件上传后所在的 storage 组名称，在文件上传成功后有 storage 服务器返回，需要客户端自行保存。

虚拟磁盘路径：storage 配置的虚拟路径，与 fastdfs 配置文件的磁盘选项 store_path*对应。如果配置了 store_path0 则是 M00，如果配置了 store_path1 则是 M01，以此类推。

数据两级目录：storage 服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。

文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器 IP 地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

9.2 文件下载流程



9.3 fastDFS 的同步机制

同一组内的 storage server 之间是对等的，文件上传、删除等操作可以在任意一台 storage server 上进行；
文件同步只在同组内的 storage server 之间进行，采用 push 方式，即源服务器同步给目标服务器；
源头数据才需要同步，备份数据不需要再次同步，否则就构成环路了；
上述第二条规则有个例外，就是新增加一台 storage server 时，由已有的一台 storage server 将已有的所有数据（包括源头数据和备份数据）同步给该新增服务器。

9.4 fastDFS 遇到的问题

启动 storage server 时，一直处于僵死状态。
启动 storage server，storage 将连接 tracker server，如果连不上，将一直重试。直到连接成功，启动才算真正完成。出现这样情况，请检查连接不上 tracker server 的原因。
友情提示：从 V2.03 以后，多 tracker server 在启动时会做时间上的检测，判断是否需要从别的 tracker server 同步 4 个系统文件。
触发时机是第一个 storage server 连接上 tracker server 后，并发起 join 请求。
如果集群中有 2 台 tracker server，而其中一台 tracker 没有启动，可能会导致 storage server 一直处于僵死状态。
执行 fdfs_test 或 fdfs_test1 上传文件时，服务器返回错误号 2
错误号 2 表示没有 ACTIVE 状态的 storage server。可以执行 fdfs_monitor 查看服务器状态。
上传文件失败，返回错误码 28，这是怎么回事？
返回错误码 28，表示磁盘空间不足。注意 FastDFS 中有预留空间的概念，在 tracker.conf 中设置，配置项为：reserved_storage_space，缺省值为 4GB，即预留 4GB 的空间。请酌情设置 reserved_storage_space 这个参数，比如可以设置为磁盘总空间的 20%左右。
nginx 扩展模块，不能正常显示图片的问题
在配置文件/etc/fdfs/mod_fastdfs.conf 中，缺省的设置是这样的：http.need_find_content_type=false 这个参数在 nginx 中需要设置为 true，apache 中应该设置为 false