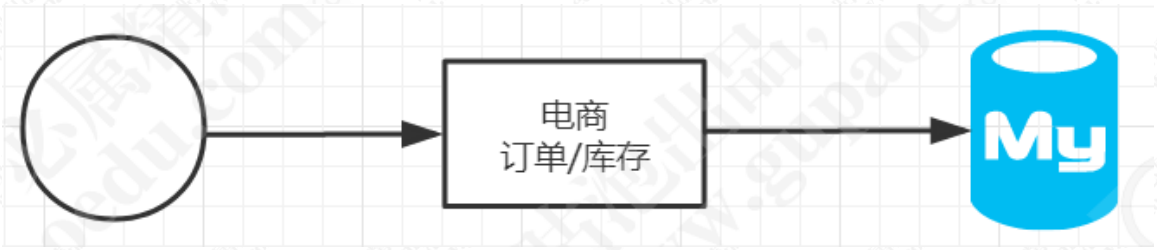
单体架构到微服务架构的带来的变化

单体架构

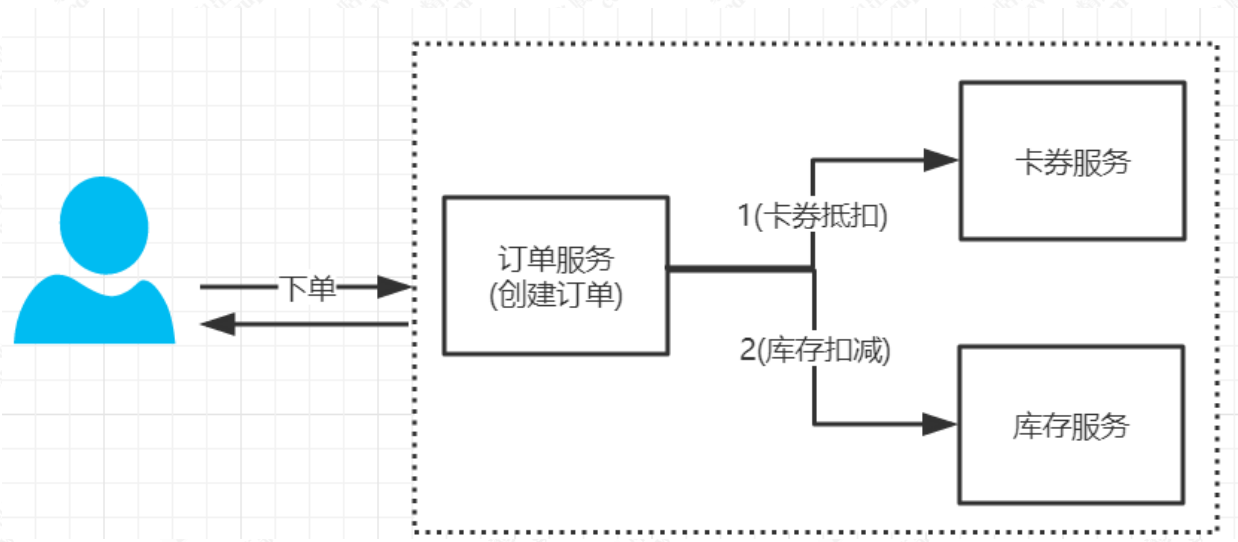


分布式架构

假设一个电商的下单场景，完成一笔订单入库，需要做几个操作

1. 创建订单
2. 卡券抵扣
3. 库存扣减

那么在分布式架构下的调用链，可能是下面这种情况。那么服务和服务之间势必会存在远程通信。



在上一次课，我们讲了服务的构建，我们通过 springboot 可以快速构建一个微服务应用，为了让大家更好的理解服务之间的通信，我们来基于 SpringBoot 模仿出上面的这种应用场景

SpringBoot+RestTemplate

创建两个服务

创建两个 spring-boot 工程

订单服务（order-service）

库存服务（repo-service）

分别创建 Controller

OrderController

```
@Autowired
RestTemplateBuilder restTemplateBuilder;

@PostMapping("/order")
public String sayHello(){
    RestTemplate rt=restTemplateBuilder.build();
    System.out.println("开始创建订单");
    rt.put("http://localhost:8081/repo/{1}",null,10001);
    return "SUCCESS";
}
```

RepoController

```
@PutMapping("/repo/{pid}")
public void serverMsg(@PathVariable("pid") String pid){
    System.out.println("扣减库存,商品 ID:"+pid);
}
```

简单了解 RestTemplate

服务于服务之间，一定不是相互隔离的，而是必须要相互联系进行数据通信才能实现完整的功能。所以在刚刚的案例中，我们拆分出来的服务使用 RestTemplate 来进行远程通信。

在了解 RestTemplate 之前，先来简单了解下 HTTP Client，我们实现对于 http 服务的远程调用，常见的手段是基于 Apache 提供的 HttpClient，或者是 Square 公司开源的 OkHttp。
还有 Netflix 公司提供的 Feign（feign 大家都比较熟悉，基于 spring cloud 开发的常见组件，可以使得我们用面向接口的编程来实现远程调用）等等.
简单来说，RestTemplate 是 Spring 提供的用来访问 REST 服务的客户端，以前我们使用 Apache HttpClient 来进行远程调用时，需要写非常多的代码，还需要考虑各种资源回收的问题。而 RestTemplate 简化了 Http 服务的通信，我们只需要提供 URL，RestTemplate 会帮我们搞定一切。
另外，需要注意的是，RestTemplate 并没有重复造轮子，而是利用了现有的技术，如 JDK 或 Apache HttpClient 、OkHttp 等来实现 http 远程调用。

虽然 RestTemplate 已经是一个很不错的 HttpClient，但是目前 Spring Cloud 中仍然采用 Feign。对于易用性和可读性这块的优势更好。后续在讲微服务组件的时候会专门讲到

源码

RestTemplate 需要使用一个实现了 ClientHttpRequestFactory 接口的类为其提供 ClientHttpRequest 实现。而 ClientHttpRequest 则实现封装了组装、发送 HTTP 消息，以及解析响应的的底层细节。

目前（5.1.8.RELEASE）的 RestTemplate 主要有四种 ClientHttpRequestFactory 的实现，它们分别是：

1. 基于 JDK HttpURLConnection 的 SimpleClientHttpRequestFactory
2. 基于 Apache HttpComponents Client 的 HttpComponentsClientHttpRequestFactory
3. 基于 OkHttp 2（OkHttp 最新版本为 3，有较大改动，包名有变动，不和老版本兼容）的 OkHttpClientHttpRequestFactory
4. 基于 Netty4 的 Netty4ClientHttpRequestFactory

消息读取的转化

RestTemplate 对于服务端返回消息的读取，提供了消息转换器，可以把目标消息转化为用户指定的格式（通过 Class<T> responseType 参数指定）。类似于写消息的处理，读消息的处理也是通过 ContentType 和 responseType 来选择的相应 HttpMessageConverter 来进行的。

Http 和 Rpc 框架的区别

虽然现在服务间的调用越来越多地使用了 RPC 和消息队列，但是 HTTP 依然有适合它的场景。

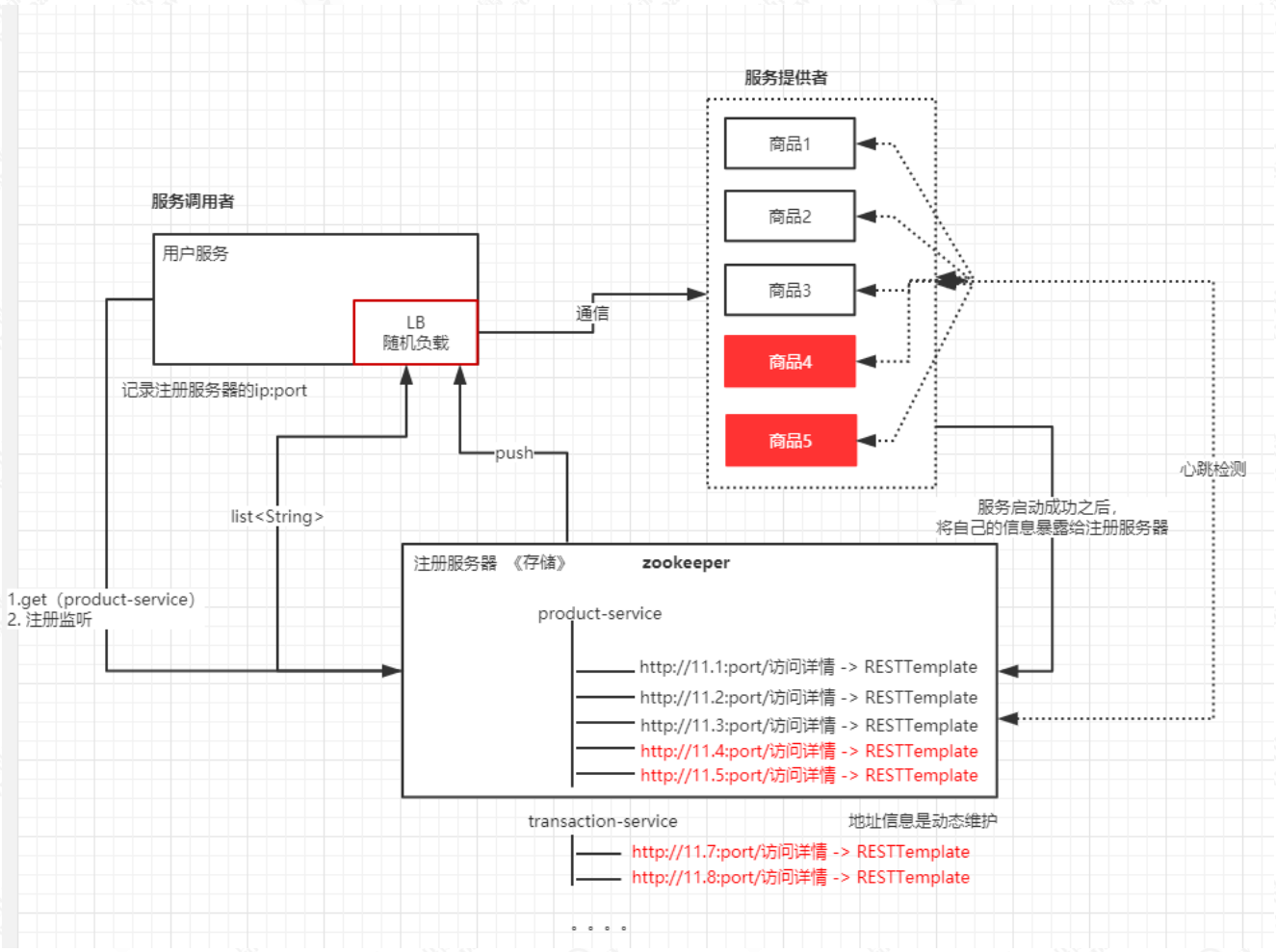
RPC 的优势在于高效的网络传输模型（常使用 NIO 来实现），以及针对服务调用场景专门设计协议和高效的序列化技术。

HTTP 的优势在于它的成熟稳定、使用实现简单、被广泛支持、兼容性良好、防火墙友好、消息的可读性高。所以 http 协议在开放 API、跨平台的服务间调用、对性能要求不苛刻的场景中有着广泛的使用。

微服务通信带来的问题

有了远程通信以后，我们势必会考虑几个问题

1. 目标服务肯定会做扩容，扩容以后，客户端会带来一些变化
2. 客户端对于目标服务如何进行负载均衡
3. 客户端如何维护目标服务的地址信息
4. 服务端的服务状态变化，如何让客户端尽心感知



引入注册中心

服务注册中心主要用于实现服务的注册和服务的发现功能，在微服务架构中，它起到了非常大的作用。

注册中心的实现

Dubbo 体系中的 Zookeeper、Spring Cloud 中的 Eureka 和 Consul

重新认识 Zookeeper

Zookeeper 的前世今生

Apache ZooKeeper 是一个高可靠的分布式协调中间件。它是 Google Chubby 的一个开源实现，那么它主要是解决什么问题的呢？那就得先了解 Google Chubby

Google Chubby 是谷歌的一个用来解决分布式一致性问题的组件，同时，也是粗粒度的分布式锁服务。

分布式一致性问题

什么是分布式一致性问题呢？简单来说，就是在一个分布式系统中，有多个节点，每个节点都会提出一个请求，但是在所有节点中只能确定一个请求被通过。而这个通过是需要所有节点达成一致的结果，所以所谓的一致性就是在提出的所有请求中能够选出最终一个确定请求。并且这个请求选出来以后，所有的节点都要知道。

这个就是典型的拜占庭将军问题

拜占庭将军问题说的是：拜占庭帝国军队的将军们必须通过投票达成一致来决定是否对某一个国家发起进攻。但是这些将军在地里位置上是分开的，并且在将军中存在叛徒。叛徒可以通过任意行动来达到自己的目标：

1. 欺骗某些将军采取进攻行动
2. 促使一个不是所有将军都统一的决定，比如将军们本意是不希望进攻，但是叛徒可以促成进攻行动
3. 迷惑将军使得他们无法做出决定

如果叛徒达到了任意一个目标，那么这次行动必然失败。只有完全达成一致那么这次进攻才可能胜利

拜占庭问题的本质是，由于网络通信存在不可靠的问题，也就是可能存在消息丢失，或者网络延迟。如何在这样的背景下对某一个请求达成一致。

为了解决这个问题，很多人提出了各种协议，比如大名鼎鼎的 Paxos； 也就是说在不可信的网络环境中，按照 paxos 这个协议就能够针对某个提议达成一致。

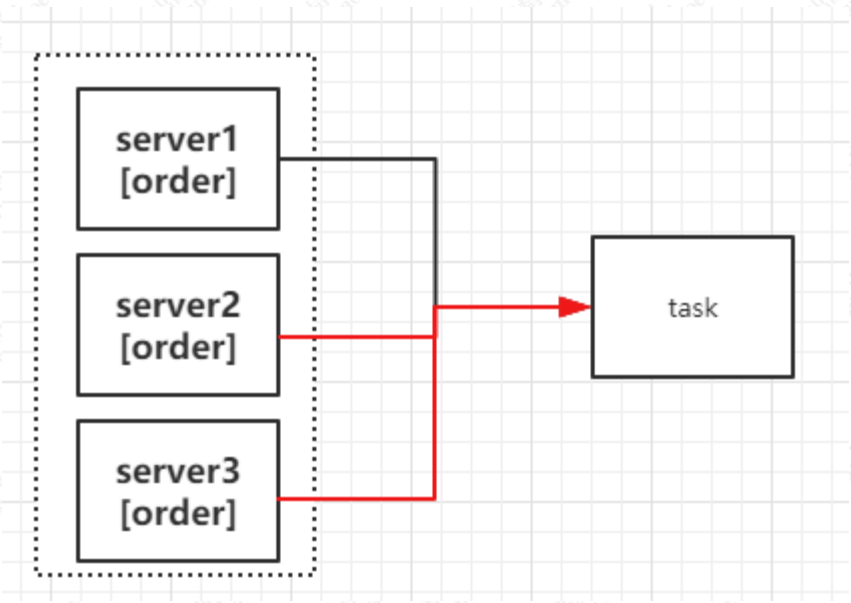
所以：分布式一致性的本质，就是在分布式系统中，多个节点就某一个提议如何达成一致

➢ 这个和 Google Chubby 有什么关系呢

在 Google 有一个 GFS(google file system)，他们有一个需求就是要从多个 gfs server 中选出一个 master server。这个就是典型的一致性问题，5 个分布在不同节点的 server，需要确定一个 master server，而他们要达成的一致性目标是：确定某一个节点为 master，并且所有节点要同意。

而 GFS 就是使用 chubby 来解决这个问题的。

实现原理是：所有的 server 通过 Chubby 提供的通信协议到 Chubby server 上创建同一个文件，当然，最终只有一个 server 能够获准创建这个文件，这个 server 就成为了 master，它会在这个文件中写入自己 的地址，这样其它的 server 通过读取这个文件就能知道被选出的 master 的地址



分布式锁服务

从另外一个层面来看，Chubby 提供了一种粗粒度的分布式锁服务，chubby 是通过创建文件的形式来提供锁的功能，server 向 chubby 中创建文件其实就表示加锁操作，创建文件成功表示抢占到了锁。

由于 Chubby 没有开源，所以雅虎公司基于 chubby 的思想，开发了一个类似的分布式协调组件 Zookeeper，后来捐赠给了 Apache。

所以，大家一定要了解，zookeeper 并不是作为注册中心而设计，他是作为分布式锁的一种设计。而注册中心只是他能够实现的一种功能而已。

zookeeper 的设计猜想

基于 Zookeeper 本身的一个设计目标，zookeeper 主要是解决分布式环境下的服务协调问题而产生的，我们来猜想一下，如果我们要去设计一个 zookeeper，需要满足那些功能呢?

防止单点故障

首先，在分布式架构中，任何的节点都不能以单点的方式存在，因此我们需要解决单点的问题。常见的解决单点问题的方式就是集群

大家来思考一下，这个集群需要满足那些功能?

1. 集群中要有主节点和从节点（也就是集群要有角色）
2. 集群要能做到数据同步，当主节点出现故障时，从节点能够顶替主节点继续工作，但是继续工作的前提是数据必须要主节点保持一直
3. 主节点挂了以后，从节点如何接替成为主节点？ 是人工干预？还是自动选举

所以基于这几个点，我们先来把 zookeeper 的集群节点画出来。

Leader 角色

Leader 服务器是整个 zookeeper 集群的核心，主要的工作任务有两项

1. 事物请求的唯一调度和处理者，保证集群事物处理的顺序性
2. 集群内部各服务器的调度者

Follower 角色

Follower 角色的主要职责是

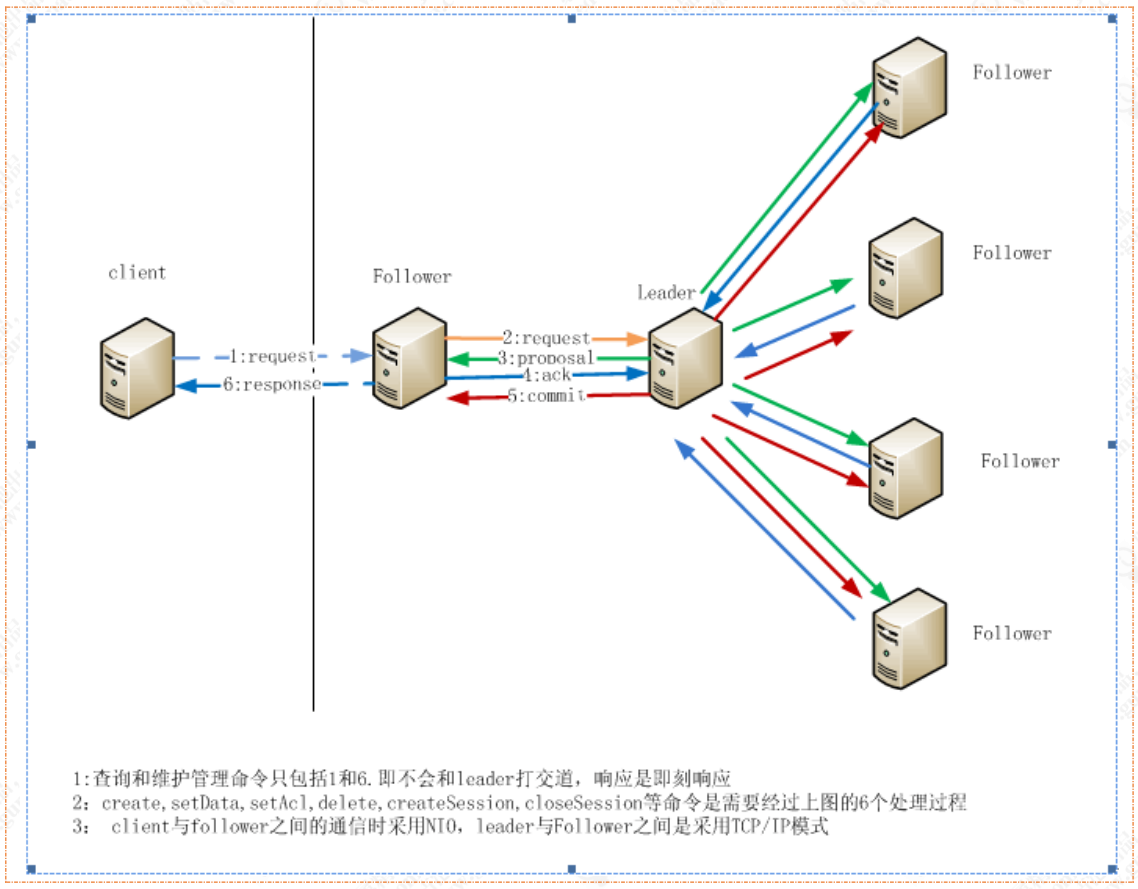
1. 处理客户端非事物请求、转发事物请求给 leader 服务器

2. 参与事物请求 Proposal 的投票（需要半数以上服务器通过才能通知 leader commit 数据; Leader 发起的提案，要求 Follower 投票）
3. 参与 Leader 选举的投票

数据同步

接着上面那个结论再来思考，如果要满足这样的一个高性能集群，我们最直观的想法应该是，每个节点都能接收到请求，并且每个节点的数据都必须要保持一致。要实现各个节点的数据一致性，就势必要一个 leader 节点负责协调和数据同步操作。这个我想大家都知道，如果在这样一个集群中没有 leader 节点，每个节点都可以接收所有请求，那么这个集群的数据同步的复杂度是非常大

所以，当客户端请求过来时，需要满足，事务型数据和非事务型数据的分开处理方式，就是 leader 节点可以处理事务和非事务型数据。而 follower 节点只能处理非事务型数据。原因是，对于数据变更的操作，应该由一个节点来维护，使得集群数据处理的简化。同时数据需要能够通过 leader 进行分发使得数据在集群中各个节点的一致性

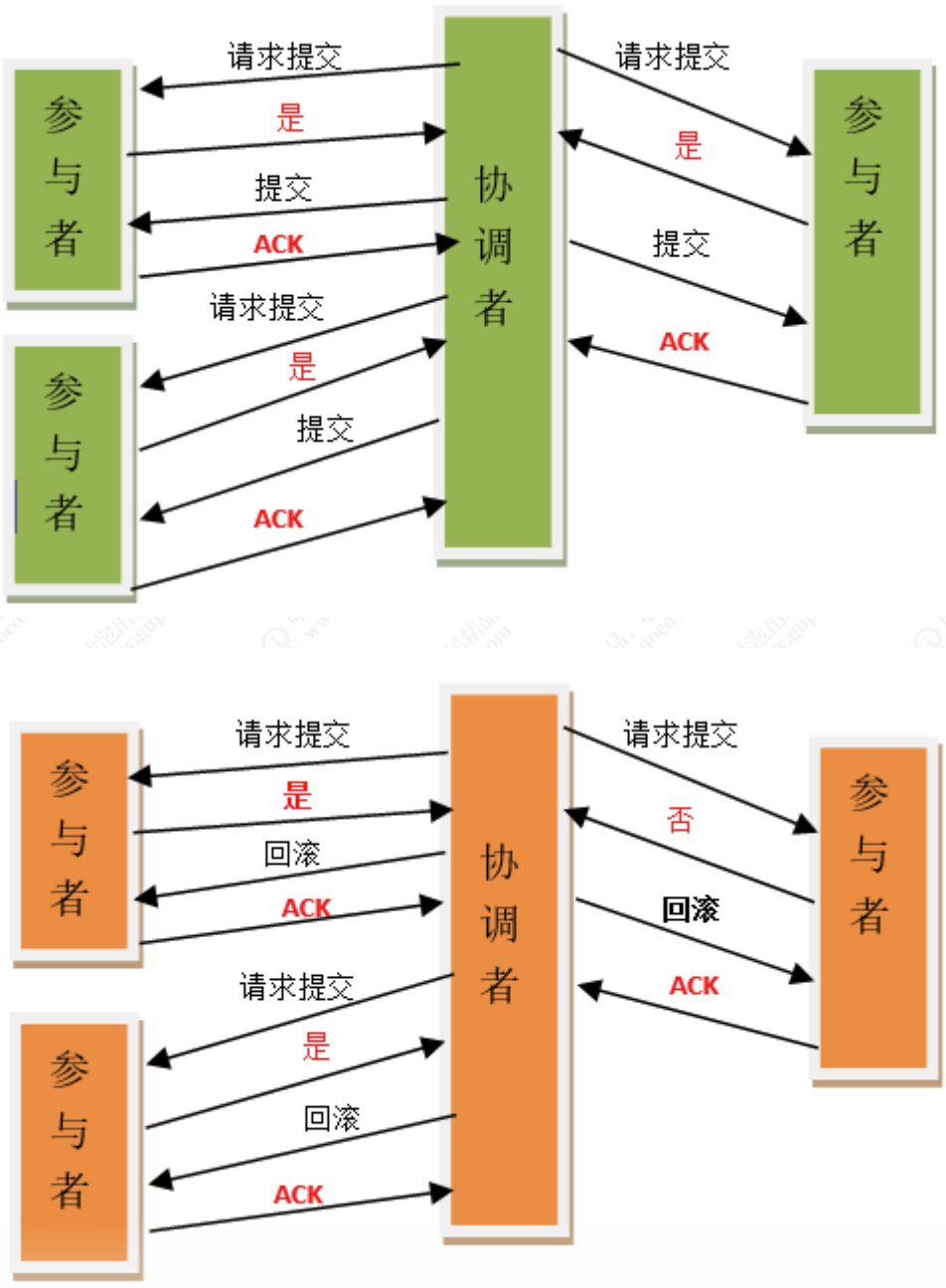


leader 节点如何和其他节点保证数据一致性，并且要求是强一致的。在分布式系统中，每一个机器节点虽然都能够明确知道自己进行的事务操作过程是成功和失败，但是却无法直接获取其他分布式节点的操作结果。所以当一个事务操作涉及到跨节点的时候，就需要用到分布式事务，分布式事务的数据一致性协议有 2PC 协议和 3PC 协议

关于 2PC 提交

（Two Phase Commitment Protocol）当一个事务操作需要跨越多个分布式节点的时候，为

了保持事务处理的 ACID 特性，就需要引入一个“协调者”（TM）来统一调度所有分布式节点的执行逻辑，这些被调度的分布式节点被称为 AP。TM 负责调度 AP 的行为，并最终决定这些 AP 是否要把事务真正进行提交；因为整个事务是分为两个阶段提交，所以叫 2pc



阶段一：提交事务请求（投票）

1. 事务询问

   协调者向所有的参与者发送事务内容，询问是否可以执行事务提交操作，并开始等待各参与者的响应

2. 执行事务

   各个参与者节点执行事务操作，并将 Undo 和 Redo 信息记录到事务日志中，尽量把提交过程中所有消耗时间的操作和准备都提前完成确保后面 100%成功提交事务

3. 各个参与者向协调者反馈事务询问的响应

   如果各个参与者成功执行了事务操作，那么就反馈给参与者 yes 的响应，表示事务可以执行；如果参与者没有成功执行事务，就反馈给协调者 no 的响应，表示事务不可以执行，上面这个阶段有点类似协调者组织各个参与者对一次事务操作的投票表态过程，因此 2pc 协议的第一个阶段称为“投票阶段”，即各参与者投票表名是否需要继续执行接下去的事务提交操作。

阶段二：执行事务提交

在这个阶段，协调者会根据各参与者的反馈情况来决定最终是否可以进行事务提交操作，正常情况下包含两种可能:执行事务、中断事务

Observer 角色

Observer 是 zookeeper3.3 开始引入的一个全新的服务器角色，从字面来理解，该角色充当了观察者的角色。

观察 zookeeper 集群中的最新状态变化并将这些状态变化同步到 observer 服务器上。Observer 的工作原理与 follower 角色基本一致，而它和 follower 角色唯一的不同在于 observer 不参与任何形式的投票，包括事物请求 Proposal 的投票和 leader 选举的投票。简单来说，observer 服务器只提供非事物请求服务，通常在于不影响集群事物处理能力的前提下提升集群非事物处理的能力

leader 选举

当 leader 挂了，需要从其他 follower 节点中选择一个新的节点进行处理，这个时候就需要涉及到 leader 选举

从这个过程中，我们推导处了 zookeeper 的一些设计思想

集群组成

通常 zookeeper 是由 2n+1 台 server 组成，每个 server 都知道彼此的存在。每个 server 都维护的内存状态镜像以及持久化存储的事务日志和快照。对于 2n+1 台 server，只要有 n+1 台（大多数）server 可用，整个系统保持可用。我们已经了解到，一个 zookeeper 集群如果要对外提供可用的服务，那么集群中必须要有过半的机器正常工作并且彼此之间能够正常通信，基于这个特性，如果向搭建一个能够允许 F 台机器 down 掉的集群，那么就要部署 2*F+1 台服务器构成的 zookeeper 集群。因此 3 台机器构成的 zookeeper 集群，能够在挂掉一台机器后依然正常工作。一个 5 台机器集群的服务，能够对 2 台机器怪调的情况下进行容灾。如果一台由 6 台服务构成的集群，同样只能挂掉 2 台机器。因此，5 台和 6 台在容灾能力上并没有明显优势，反而增加了网络通信负担。系统启动时，集群中的 server 会选举出一台 server 为 Leader，其它的就作为 follower（这里先不考虑 observer 角色）。

之所以要满足这样一个等式，是因为一个节点要成为集群中的 leader，需要有超过及群众过半数的节点支持，这个涉及到 leader 选举算法。同时也涉及到事务请求的提交投票

Zookeeper 的安装部署

安装

zookeeper 有两种运行模式：集群模式和单击模式。

下载 zookeeper 安装包：http://apache.fayea.com/zookeeper/

下载完成，通过 tar -zxvf 解压

常用命令

1. 启动 ZK 服务:

   bin/zkServer.sh start

2. 查看 ZK 服务状态:

   bin/zkServer.sh status

3. 停止 ZK 服务:

   bin/zkServer.sh stop

4. 重启 ZK 服务:

   bin/zkServer.sh restart

5. 连接服务器

   zkCli.sh -timeout 0 -r -server ip:port

单机环境安装

一般情况下，在开发测试环境，没有这么多资源的情况下，而且也不需要特别好的稳定性的前提下，我们可以使用单机部署；

初次使用 zookeeper，需要将 conf 目录下的 zoo_sample.cfg 文件 copy 一份重命名为 zoo.cfg

修改 dataDir 目录，dataDir 表示日志文件存放的路径（关于 zoo.cfg 的其他配置信息后面会讲）

集群环境安装

在 zookeeper 集群中，各个节点总共有三种角色，分别是：leader，follower，observer

集群模式我们采用模拟 3 台机器来搭建 zookeeper 集群。分别复制安装包到三台机器上并解压，同时 copy 一份 zoo.cfg。

1. 修改配置文件

修改端口

server.1=IP1:2888:3888 【2888：访问 zookeeper 的端口；3888：重新选举 leader 的端口】

server.2=IP2.2888:3888

server.3=IP3.2888:2888

server.A=B：C：D：其 中

A 是一个数字，表示这个是第几号服务器；

B 是这个服务器的 ip 地址；

C 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；

D 表示的是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，

而这个端口就是用来执行选举时服务器相互通信的端口。如果是伪集群的配置方式，由于 B 都是一样，

所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。

在集群模式下，集群中每台机器都需要感知到整个集群是由哪几台机器组成的，在配置文件中，按照格式 server.id=host:port:port，每一行代表一个机器配置

id：指的是 server ID,用来标识该机器在集群中的机器序号

2. 新建 datadir 目录，设置 myid

在每台 zookeeper 机器上，我们都需要在数据目录(dataDir)下创建一个 myid 文件，该文件只有一行内容，对应每台机器的 Server ID 数字；比如 server.1 的 myid 文件内容就是 1。【必须确保每个服务器的 myid 文件中的数字不同，并且和自己所在机器的 zoo.cfg 中 server.id 的 id 值一致，id 的范围是 1~255】

4. 启动 zookeeper