Zookeeper分布式系统开发实战 第3课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

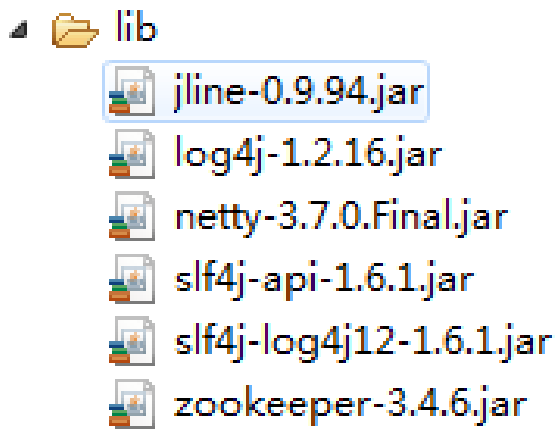
http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

第三讲 java客户端

- 准备
  - 通过eclipse创建一个普通java工程
  - 工程的src目录中创建lib目录
  - Api文档： http://zookeeper.apache.org/doc/r3.4.6/api/index.html
  - 导入zk的jar包：



第三讲 java客户端

- 创建会话
  - ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)
  - ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly)
  - ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd)
  - ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly)
  - 客户端与服务端会话的建立是一个异步的过程，即
    - 完成客户端的初始化后就返回，此时连接并没有真正的建立起来
    - 当连接真正建立起来后，客户端会收到一个事件通知

第三讲 java客户端

- Zk构造方法参数说明

参数名	说明
connectString	1.指zk的服务器列表，以英文输入法下逗号分割的host：port，比如：192.168.1.1：2181，192.168.1.2：2181 2.也可以通过在后面跟着根目录，表示此客户端的操作都是在此目录下，比如：192.168.1.1：2181，192.168.1.2：2181/zk-book；这个表示此客户端操作的节点都是在/zk-book根目录下，比如创建/foo/bar，实际完整路径为/zk-book/foo/bar
sessionTimeout	会话超时时间，单位是毫秒；当在这个时间内没有收到心跳检测，会话就会失效
watcher	注册的watcher，null表示不设置
canBeReadOnly	用于标识当前会话是否支持”read-only”模式； ”read-only”模式：当zk集群中的某个机器与集群中过半以上的机器网络端口后，此机器将不会接受客户端的任何读写请求，但是，有时候，我们希望继续提供读请求，因此设置此参数为true，即此客户端还可以从与集群中半数以上节点网络不通的机器节点中读取数据
sessionId和sessionPasswd	分别代表会话ID和会话密钥；这两个参数一起可以唯一确定一个会话，客户端通过这两个参数可以实现客户端会话复用

第三讲 java客户端

```java
public class ZKCreateSample implements Watcher{
    private static CountDownLatch connectedSemaphore = new CountDownLatch(1);
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        ZooKeeper zookeeper = new ZooKeeper("localhost:2181",6000, new ZKCreateSample());
        System.out.println("begin state="+zookeeper.getState());
        try {
            connectedSemaphore.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
            System.out.println("Zookeeper session established.");
        }
        System.out.println("end state="+zookeeper.getState());
    }

    @Override
    public void process(WatchedEvent event) {
        // TODO Auto-generated method stub
        System.out.println("receive watched event:"+event);
        if(KeeperState.SyncConnected==event.getState()){
            connectedSemaphore.countDown();
        }
    }
```

```
<terminated> ZKCreateSample [Java Application] D:\Program Files\Java\jdk1.7.0_67\bin\javaw.exe (201
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
begin state=CONNECTING
receive watched event:WatchedEvent state:SyncConnected type:None path:null
end state=CONNECTED
```

第三讲 java客户端

- 创建节点
  - String create(String path, byte[] data, List<ACL> acl, CreateMode createMode)
    - 以同步的方式创建节点
  - void create(String path, byte[] data, List<ACL> acl, CreateMode createMode, AsyncCallback.StringCallback cb, Object ctx)
    - 以异步的方式创建节点
  - 无论是上面的同步或者异步都不支持递归创建节点
  - 当节点存在时，抛出异常NodeExistsException

第三讲 java客户端

- Zk create api参数说明

参数名	说明
path	被创建的节点路径，比如：/zk-book/foo
data[]	节点中的数据
acl	Acl策略
createMode	节点类型，枚举类型，有四种选择： ●持久（PERSISTENT） ●持久顺序（PERSISTENT_SEQUENTIAL） ●临时（EPHEMERAL） ●临时顺序（EPHEMERAL_SEQUENTIAL）
cb	异步回调函数，需要实现接口StringCallback接口；当服务器端创建完成后，客户端会自动调用这个对象的方法processResult
ctx	用于传递一个对象，可以在回调方法执行的时候用，通常用于传递业务的上下文信息

第三讲 java客户端

- 创建节点时的ACL
  - 通过接口Ids可以有预先定义的几种scheme模式
    - OPEN_ACL_UNSAFE：相当于world:anyone:cdrwa
    - CREATOR_ALL_ACL:相当于auth:用户名：密码,但是需要通过ZooKeeper的addAuthInfo添加对应的用户和密码对
    - READ_ACL_UNSAFE：相当于world:anyone:r，即所有人拥有读权
  - 可以自己定义，比如：

```
public List<ACL> getDigestAcl(){
    List<ACL> acls = new ArrayList<ACL>();
    Id digestId = new Id("digest", "javaclient2:CGf2ylBdcKMdCYuzd08lQfOPvN0=");
    acls.add(new ACL(Perms.ALL,digestId));
    return acls;
}
```

添加digest模式的权限

第三讲 java客户端

- 删除节点
  - void delete(String path, int version)
    - 以同步的方式删除节点
  - void delete(String path, int version, AsyncCallback.VoidCallback cb, Object ctx)
    - 以异步的方式删除节点,如果写测试代码，客户端主线程不能退出，否则可能请求没有发到服务器或者异步回调不成功

参数名	说明
Path	被删除的节点的路径
Version	知道节点的数据版本，如果指定的版本不是最新版本，将会报错，它的作用类似于hibernate中的乐观锁
cb	异步回调函数
ctx	传递上下文信息，即操作之前的信息传递到删除之后的异步回调函数里面

第三讲 java客户端

- 获取子节点
  - List<String> getChildren(String path, boolean watch)
    - 返回path节点的子节点列表
  - Void getChildren(String path, boolean watch, AsyncCallback.Children2Callback cb, Object ctx)
    - 以异步的方式返回子节点，返回path指定节点的状态信息（stat）
  - void getChildren(String path, boolean watch, AsyncCallback.ChildrenCallback cb, Object ctx)
    - 以异步的方式返回子节点，不返回path节点的状态信息（stat）
  - List<String> getChildren(String path, boolean watch, Stat stat)
    - 返回stat和子节点

第三讲 java客户端

- 获取子节点
  - List<String>　getChildren(String path, Watcher watcher)
  - void　getChildren(String path, Watcher watcher, AsyncCallback.Children2Callback cb, Object ctx)
  - void　getChildren(String path, Watcher watcher, AsyncCallback.ChildrenCallback cb, Object ctx)

第三讲 java客户端

- Zk getChildren参数说明

参数名	说明
path	数据节点的路径，比如：/zk-book/foo，获取该路径下的子节点列表
watcher	设置watcher后，如果path对应节点的子节点数量发生变化，将会得到通知，允许为null
watch	是否使用默认的watcher
stat	指定数据节点的状态信息
cb	异步回调函数
ctx	用于传递一个对象，可以在回调方法执行的时候用，通常用于传递业务的上下文信息

第三讲 java客户端

- 获取节点数据
  - void getData(String path, boolean watch, AsyncCallback.DataCallback cb, Object ctx)
  - byte[] getData(String path, boolean watch, Stat stat)
  - void getData(String path, Watcher watcher, AsyncCallback.DataCallback cb, Object ctx)
  - byte[] getData(String path, Watcher watcher, Stat stat)

```
public class Stat implements Record {
  private long czxid;
  private long mzxid;
  private long ctime;
  private long mtime;
  private int version;
  private int cversion;
  private int aversion;
  private long ephemeralOwner;
  private int dataLength;
  private int numChildren;
  private long pzxid;
  public Stat() {
  }
```

第三讲 java客户端

- Zk getData参数说明

参数名	说明
path	数据节点的路径，比如：/zk-book/foo，获取该路径节点的数据
watcher	设置watcher后，如果path对应节点的数据发生变化，将会得到通知，允许为null
watch	是否使用默认的watcher
stat	指定数据节点的状态信息
cb	异步回调函数
ctx	用于传递一个对象，可以在回调方法执行的时候用，通常用于传递业务的上下文信息

- 修改数据
  - Stat setData(String path, byte[] data, int version)
  - void setData(String path, byte[] data, int version, AsyncCallback.StatCallback cb, Object ctx)

参数名	说明
Path	被修改的节点的路径
data	新的数据
Version	知道节点的数据版本，如果指定的版本不是最新版本，将会报错，它的作用类似于hibernate中的乐观锁
cb	异步回调函数
ctx	传递上下文信息，即操作之前的信息传递到删除之后的异步回调函数里面

第三讲 java客户端

- 检查节点是否存在
  - Stat exists(String path, boolean watch)
    - 如果节点不存在则返回null
  - void exists(String path, boolean watch, AsyncCallback.StatCallback cb, Object ctx)
  - Stat exists(String path, Watcher watcher)
  - void exists(String path, Watcher watcher, AsyncCallback.StatCallback cb, Object ctx)

```
interface StatCallback extends AsyncCallback {
    public void processResult(int rc, String path, Object ctx, Stat stat);
}
```

第三讲 java客户端

- Zk exists API参数说明

参数名	说明
path	数据节点的路径，比如：/zk-book/foo，即API调用的目的是检测该节点是否存在
watcher	注册的watcher，用于监听一下三个事件： •节点被创建 •节点被删除 •节点被更新
watch	是否使用默认的watcher
cb	异步回调函数
ctx	用于传递一个对象，可以在回调方法执行的时候用，通常用于传递业务的上下文信息

第三讲 zk集群—分布式系统理解

- A distributed system consists of multiple computers that communicate through a computer network and interact with each other to achieve a common goal
  - 多台计算机构成
  - 计算机之间通过网络进行通信
  - 彼此进行交互
  - 共同目标

来自wikipedia

- 分布式系统
  - 一个硬件或者软件组织分布在网络计算机上，仅仅通过消息传递进行通信和动作协调的系统

----分布式系统概念和设计

第三讲 zk集群—分布式系统理解

- 业务对IT系统的要求
  - 性能
    - 快速响应
    - 支持更多的用户
    - 支持更多的业务功能
  - 可用性
    - 7×24小时在线
    - 快速恢复
  - 安全
    - 数据不能丢失（损坏，误操作追回）
    - 数据一致性

第三讲 zk集群—分布式系统理解

- 传统的IT系统如何满足业务需求
  - 性能
    - 优化代码，减少对资源的消耗
    - 买更贵，配置更好的服务器
  - 可用性
    - 优化代码，避免程序导致系统宕机
    - 买更贵更可靠的服务器，降低服务器宕机率
  - 安全
    - 数据不能丢失（损坏，误操作追回）
    - 买更贵，有多种磁盘防护的机器

缺点

✓无论多贵的硬件，计算能力及存储能力都有上限

✓无论多贵的硬件，都有宕机的时候，可能是掉电，也可能是硬件损坏

✓即使有性能足够，也不宕机的硬件主机，也不是每个公司都有购买的实力

第三讲 zk集群—分布式系统理解

- 分布式系统如何满足业务需求
  - 性能
    - 优化代码，减少对资源的消耗
    - 增加集群节点（PC Server即可）
    - Sharding（拆分功能点和数据量）
  - 可用性
    - 优化代码，避免程序导致系统宕机
    - 集群，同一个业务采用多台服务器，宕机一个不受影响
  - 安全
    - 副本：数据实时冗余存储

分布式系统优点

✓资源充分利用，不够再加

✓理论上容量无上限

✓不用担心单节点宕机导致不可用或者数据丢失

第三讲 zk集群—分布式系统理解

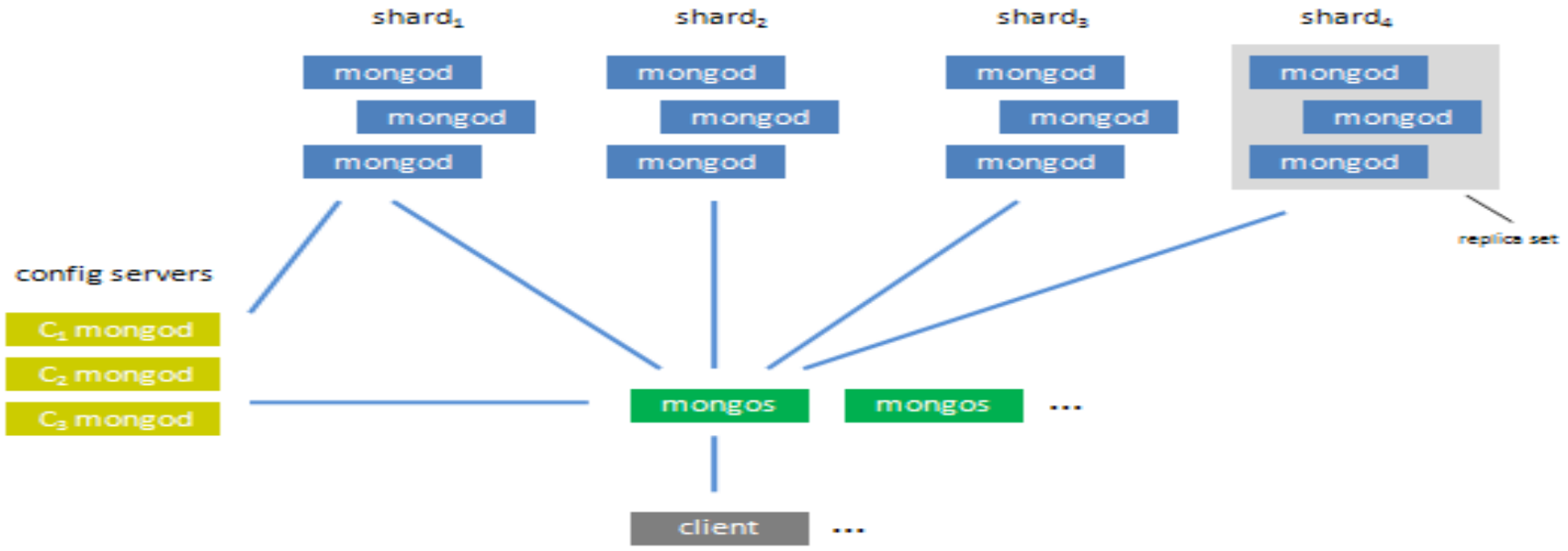
- 分布式系统缺点或者挑战
  - 增加系统复杂性
    - 软件结构复杂性
    - 运维复杂性
  - 增大获取数据一致性的难度
  - 影响系统稳定性的因素会更多，比如网络不可靠导致问题
  - 增大系统开发及调试难度
  - 增大系统获取安全性的难度

第三讲 zk集群—分布式系统理解

- 分布式系统之集群
  - 所有节点维护的数据一致
  - 所有节点都可以提供相同的业务功能（不一定是在同一时刻提供）
  - 可以保障系统的高可用，某个节点宕机不会影响服务
  - 集群举例：zookeeper tomcat
- 集群环境下如何保障数据一致性
  - 数据复制
    - 单节点写入再复制到其它节点，zookeeper就是这样实现的
    - 多节点同时写入：只适合多节点写入的数据不是相同数据的应用场景，比如tomcat集群
  - 集中存储
    - 借助可靠性较高的集中存储，比如SAN NAS存储，分布式缓存（redis）等

第三讲 zk集群—分布式系统理解

- 分布式系统之sharding
  - 对于业务系统来说就是业务拆分，不同的子模块单独是一个集群，整体业务系统是个大分布式系统
    - 举例：京东 淘宝等系统都是由很多子系统构成支撑的
  - 对于数据系统（比如mongodb ）来说就是数据的拆分，一般基于每个数据片上的功能都相同

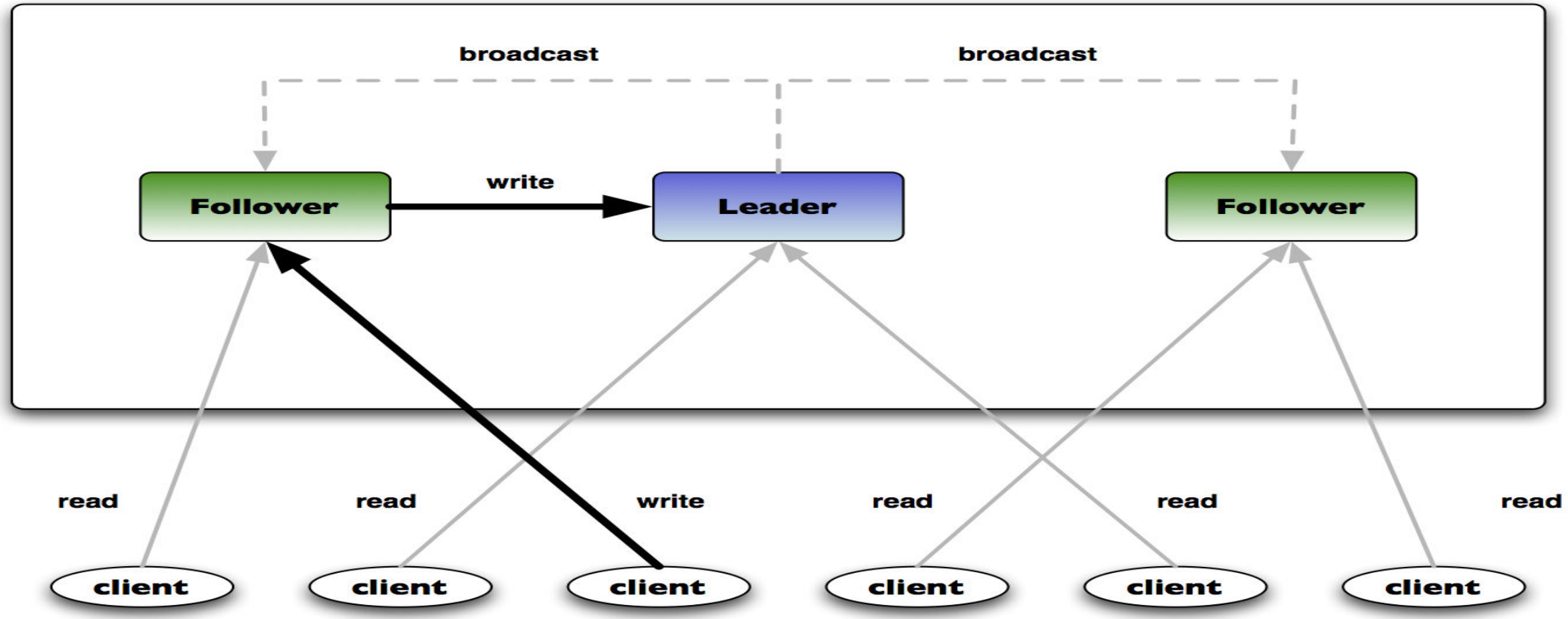


第三讲 zk集群—分布式系统理解

- Zookeeper集群
  - 是一种对等集群，所有节点（机器）数据都一样
  - 集群节点之间靠心跳感知彼此的存在
  - 所有写操作都在主节点，其它节点只能读，虽然可以接收写请求，但是内部会把写操作转给主节点
  - 通过选举机制选出主节点，从而保障了主节点的高可用
  - 至少3个节点，必须是基数个节点（机器），后面讲选举算法的时候再讲为什么
  - 当一半以上的节点数据写入成功后，则返回写入成功，是最终一致性的策略

第三讲 zk集群—分布式系统理解

- Zookeeper集群



第三讲 zk集群—分布式系统理解

- 分布式系统理论之CAP定义
  - 一个分布式系统不可能同时满足一致性（Consistency）、可用性（Availability）和分区容忍性（Partition tolerance）这丧基本需求，只能满足其中的两项
- CAP解读
  - 一致性：分布式环境下，一致性主要是指数据在多个副本间是否保持一致
  - 可用性：是指系统提供的服务必须一直处于可用的状态，对于用户的请求总是能够在有限的时间内返回结果；有限时间强调的是用户能接受的时间
  - 可用性与常说的高可用性的可用性不是一个概念，即如果服务访问不到，不属于没有可用性
  - P：集群出现网络割裂（即脑裂）时，集群还能继续提供一定的可用性和一致性，除非整个网络不可用
  - 只能满足其中的两项不是另外一项就完全没有，而是要求没有那么严格
  - 分区容忍是分布式系统必须有的特性，因为网络不可靠，所以只能在C和A中进行权衡

第三讲 zk集群—分布式系统理解

- CAP解读

放弃CAP定理	说明
放弃P	要么集群节点无状态，要么把所有数据都放在一个节点上，同时也将会失去扩展性，也不能叫做分布式系统了；所以，单机应用因为放弃了P，得到很好的CA
放弃A	放弃A，并不是完全没有A，是指允许响应超时的时间可以更长，比如报表可以运行10分钟左右，甚至在某些情况下允许超时错误，比如在双11的时候，支付宝支付页就出现了超时错误
放弃C	放弃一致性之的是放弃数据的强一致性，而保留数据的最终一致性，即数据最终是完全一致的，但是有一个时间窗口的问题，这需要根据不同的业务来定义，比如：新浪微博的内容，时间窗口就可以超过10分钟；而交易订单一般就允许几秒

第三讲 zk集群—分布式系统理解

- 分布式系统理论之BASE理论
  - BASE是Basically Available（基本可用）、 Soft state（软状态）和Eventually Consistent（最终一致性）三个短语的简写，它是对CAP中的一致性和可用性权衡的结果。
  - BASE的核心思想是即使无法做到强一致性，但是每个应用可以根据自身的业务特点，采用适当的方式达到最终一致性，同时获取到系统可用性

- BASE之基本可用
  - 响应时间上的损失：有些要求1s内返回，有些要求5s内返回
  - 功能上的损失：对于电商来说，某些区域可能不能购买某些商品，又或者大促时，部分消费者被引流到降级的页面

第三讲 zk集群—分布式系统理解

- BASE之弱状态
  - 也称为软状态，是指允许系统中的数据存在中间状态，并认为该状态不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间存在一定的延时
  - 举例：双11的时候，我在京东购买东西，明明已经付款成功，但是系统提示的还是待付款

- BASE之最终一致性
  - 系统中的数据副本在经过一段时间同步后，最终能够达到一个一致的状态

第三讲 zk集群—集群部署

- 安装java运行环境，1.6版本以后（参加第一节课）
- 下载Zookeeper（参加第一节课）
- 配置文件zoo.cfg
- 创建myid文件
  - 在dataDir配置目录下添加此文件，只有一个数字
- 按照相同的步骤配置其它几个机器
- 启动服务器
- 验证服务
  - 可以用telnet ip 端口，然后执行stat查看结果
  - 也可以直接用zkcli.sh或者zkCli.bat -server ip:port 连接到zk集群中的任何节点

第三讲 zk集群—集群部署

- 配置文件zoo.cfg
  - 机器节点：server.1=IP1:2888:3888,
    - 其中的1是机器序号，与myid文件的数字一致，范围为1-255
    - 2888是follower服务器与leader之间进行运行时通信和数据同步的端口
    - 3888是选举过程中的投票通信端口
  - initLimit
    - 用于leader等待follower启动和数据同步完成后的时间
    - 它不是具体的时间，ticktime×initLimit才是真正的时间
    - 默认值是10，如果ticktime为2000ms，则默认等待follower的时间是20s
    - 当zk的节点数据较多时，可以适当调大，具体多大，需要根据当时情况来定，一般默认配置即可

第三讲 zk集群—集群部署

- 配置文件zoo.cfg
  - syncLimit
    - 用于follower和leader之间的心跳检测的最大延迟时间，超过这个时间表示Follower已经脱离了leader的网
    - 它不是具体的时间，ticktime×syncLimit才是真正的时间
    - 默认值是5
    - 一般不用修改，如果网络环境不稳定，可以适当调高

第三讲 zk集群—集群部署

```
G:\开发软件\zookeeper-3.4.6\zookeeper-3.4.6\bin>zkCli.cmd -server localhost:2181
Connecting to localhost:2181
2015-11-28 09:50:44,582 [myid:] - INFO  [main:Environment@100] - Client environme
```

```
G:\开发软件\zookeeper-3.4.6\zookeeper-3.4.6\bin>zkCli.cmd -server localhost 2182
Connecting to localhost
2015-11-28 09:48:41,377 [myid:] - INFO  [main:Environment@100] - Client environme
```

```
[zk: localhost:2181(CONNECTED) 0] ls /
[acl, zk-book, zk-acl, zk-createtest, zk-book2, nodeauth, watchtest, newznode, zookeeper, nodetest,
[zk: localhost:2181(CONNECTED) 1] create /zkCluster clusterdata
Created /zkCluster
```

在2181命令行创建节点

```
[zk: localhost:2182(CONNECTED) 2] ls /
[acl, zk-book, zk-acl, zk-createtest, zk-book2, nodeauth, zkCluster, watchtest, newznode, zookeeper,
[zk: localhost:2182(CONNECTED) 3] get /zkCluster
clusterdata
```

在2182命令行读取节点

Thanks

FAQ时间