1. Zookeeper的设计目标
2. 简单目录结构，zookeeper以简单树形结构进行相互协调的
3. 可以构成集群。3-5台，只要集群中超过半数以上机器能够正常工作，整个集群就可以对外提供服务
4. 顺序访问，对于每一个来自客户端的请求，zookeeper分配一个全局的递增id,这个编号反应了所有事务的操作顺序

应用系统可以使用zookeeper的这个特性来实现更高层次的同步。

1. 高性能：由于zookeeper将全量数据存储在内存中，并直接服务于所有的非事务请求。尤其在读的情况下，性能非常突出，可以达到12w-13w/s
2. Java操作zookeeper
3. 代码：和zookeeper建立连接

Static final CountDownLatch coonectedSemaphore = new CountDownlatch(1);

Public static void main(String args[]){

ZooKeeper zk = new ZooKeeper(Connection\_Addr,Session\_outtime,new Watcher()){

Public void process(WatchedEvent event){

keeperState keeperState = event.getState();

EventType eventType = event.getType();

If(keeperState.SyncConnected == keeperState){

If(EventType.None == eventType){

coonectedSemaphore.countDown();

System.out.println(“zk已经建立连接”);

}

}

}

});

//进行阻塞

connectedSemaphore.await();

}

1. Zookeeper构造方法，因为建立会话是一种异步的过程，所以通过countDownLatch等到SyncConnection再继续执行
2. 创建znode节点

Private void createNode(Zookeeper zk,String url){

Try{

Byte data[] = url.getBytes();

//创建一个临时节点，把url存进去

String path = zk.create(Constant. ***ZK\_PROVIDER\_PATH,data,*** ZooDefs.Ids.***OPEN\_ACL\_UNSAFE***, CreateMode.***EPHEMERAL\_SEQUENTIAL);***

} catch(KeeperException | InterruptedException){

System.out.println(e.getMessage());

}

}

//监听/registry节点下所有子节点是否有变化

Private void watchNode(final Zookeeper zk){

Try{

List<String> nodeList = zk.getChildren(Constant.***ZK\_REGISTRY\_PATH***, **new** Watcher(){

Public void process(WatchedEvent event){

If(event.getType()== Event.EventType.***NodeChildrenChanged) {***

***watchNode(zk);***

***}***

}

List <String> dataList = new ArrayList<>();

For(String node: nodeList){

Byte[] data = zk.getData(Constant.***ZK\_REGISTRY\_PATH*** + "/" + node, **false**, **null**);

System.out.println(“new String (data)=”+new String(data);

dataList.add(new String(data));

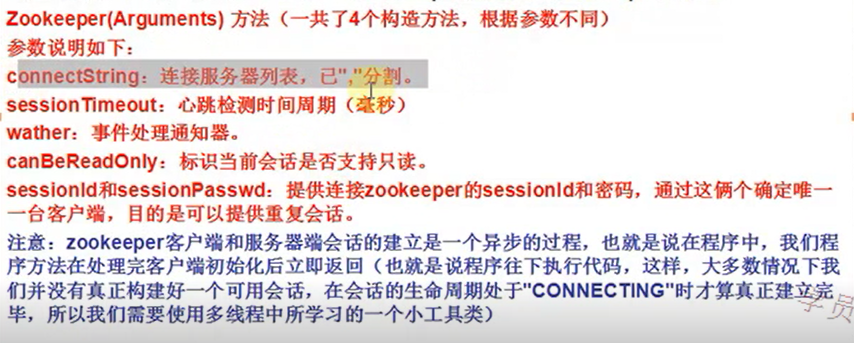
}

});

}**catch** (KeeperException | InterruptedException e) {

***LOGGER***.error("", e);

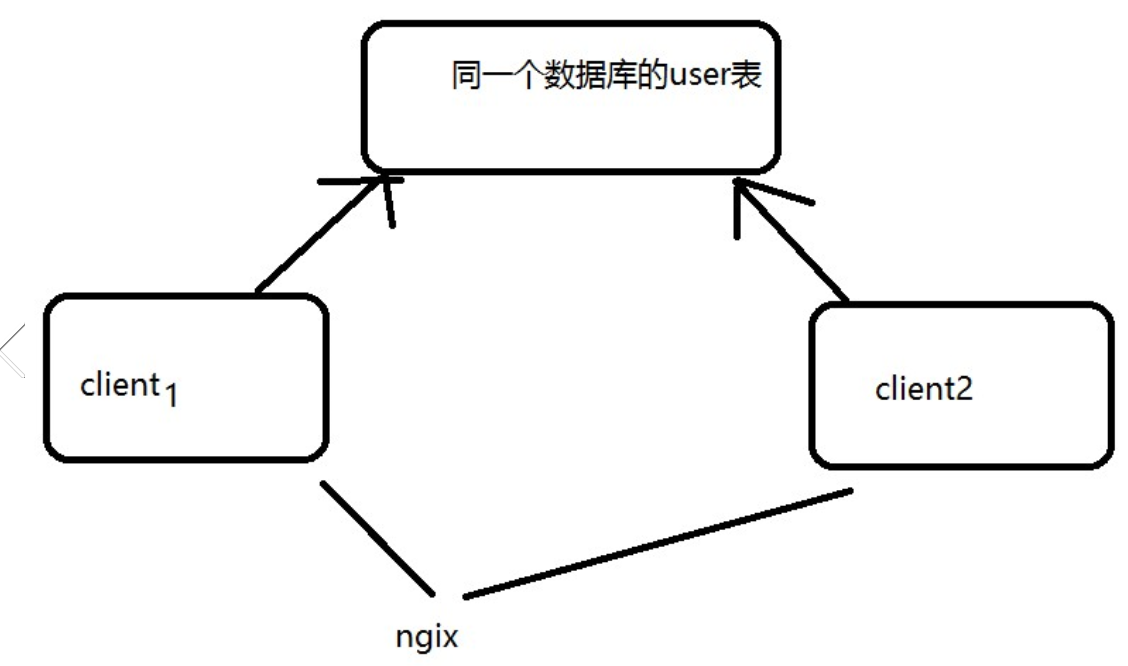
}



1. Zookeeper**解决分布式系统之间数据不一致的问题**，下面给出了一个应用场景：

现在有两个client端，都部署的相同代码，两个客户端操作同一个数据库比如说user表，然后要实现同一时间点不允许两个客户端同时给user的age减值或者是加值

并发量大情况下，上面说的这个情况就可能会出现并发的一个数据不一致的问题



利用zookeeper，一个client端一个请求过来的时候先在zookeeper上创建一个临时的节点，id是1，当我第二个client端也想对相同的用户操作

也想去create znode，会发现创建不了；它先get一遍看看有没有id = 1的这个节点，如果有的话就等待，没有的话就创建；

**Zookeeper支持同一时间点只能有一个client端去创建id，不能并发的同时去修改这个节点**，当client1操作完成之后会自动的把临时节点清空；

这个时候client2再去get发现id=1的节点没有，再去访问；这种方式就像加了一把锁一样，分布式锁

伪代码：

Client age = 81 : USER ID :1001

Client age = 82 : USER ID :1001

Get /user/1001 – 如果没有

Create node (EPHEMERAL)

Update … age 81(ip) 持久化操作 count = 5 + 1

Zk.close()

C2 操作----等待C1释放节点

1. Java 操作zookeeper(二)
2. 同步创建节点：
3. 节点路径（名称）：nodeName（不允许递归创建节点，递归删除，也就是说父节点不存在的情况下，不允许创建子节点）
4. 节点内容要求是字节数组
5. 节点权限，一般使用开发权限即可。
6. 节点类型：创建节点的类型，createMode.\*

PERSISENT(持久化节点)

PERSISENT SEQUENTIAL(持久化顺序节点)

EPHEMERAL(临时节点)

EPHEMERAL SEQUENTIAL(临时顺序节点)

1. Java 操作zookeeper（三）
2. 异步方式

在同步的基础上增加了两个参数

1. 注册一个异步回调函数

Zk.delete(“/testRoot”,-1,new AsyncCallback.VoidCallback() {

Public void processResult(int rc, String path,Object ctx){

System.out.println(rc);

System.out.println(path);

System.out.println(ctx);

}

}

);

Rc是响应码，0表示成功，-4表示端口连接，-110表示指定节点存在，-112表示节点已经存在

Path 表示接口调用时传入API的数据节点路径

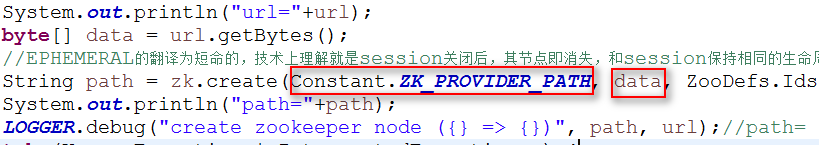
Ctx: 为接口调用传入的API的Ctx值

Name 节点名称

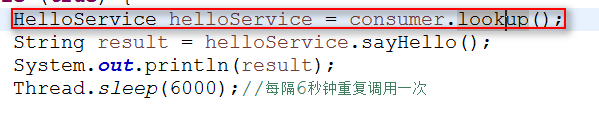
1. 程序实例：<https://my.oschina.net/huangyong/blog/345164>

有两个server，一个客户端

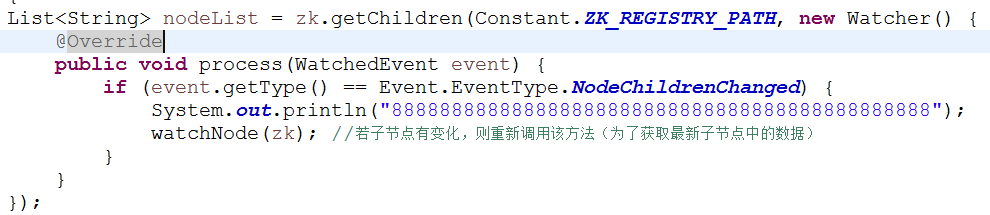
两个Server分别向zookeeper /registry/provider路径下注册临时节点



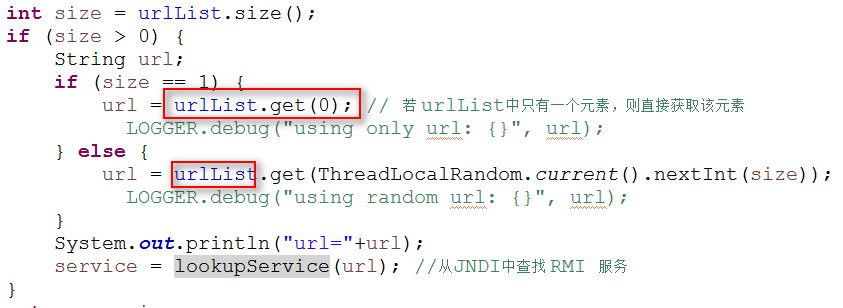
下面启动客户端：

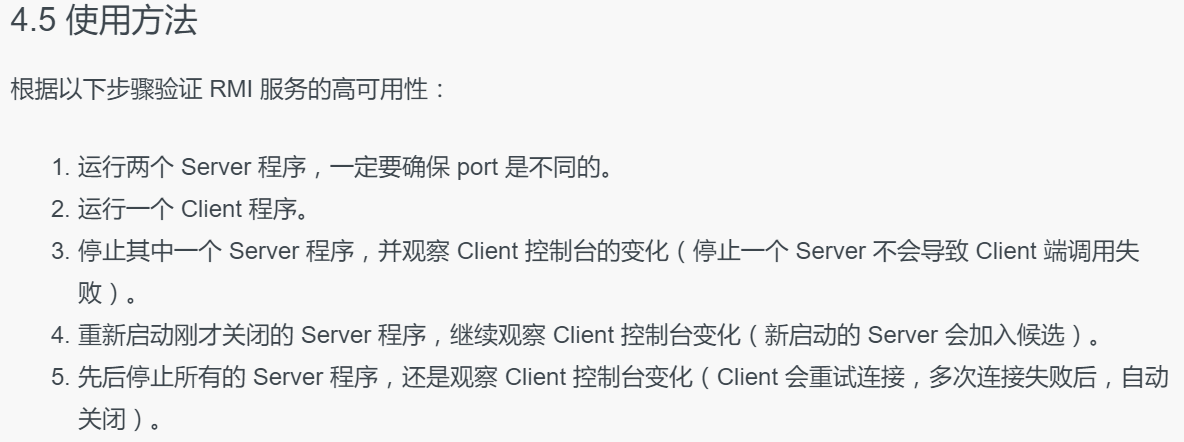


观察/registry/provider节点是否有变化，通过监听NodeChildrenChanged事件获取最新的服务providers 列表



通过调用consume的lookup方法来查找RMI服务





总结：

我们尝试使用zookeeper实现了一个RMI的高可用方案，通过zookeeper注册所有服务提供者的RMI服务，让服务消费者监听

Zookeeper的zNode节点，从而获取当前可用的RMI服务。