ZooKeeper

主讲：崔译

一、简介

官方网站

- 由 阿里 设计的 ，现在由 Apache 负责维护的
- 是一个 开源的 ， `分布式的` ， 为 `分布式系统` 提供 `协调服务` 的 项目
- 简单来说：Zookeeper 主要包括两部分
  - 文件系统
  - 通知机制
- 从设计模式的角度来理解：是一个基于 `观察者模式` 设计的 分布式框架

> 订阅中心：
>
> ZooKeeper 负责维护 和 管理 大家（项目）所关心的公共数据
>
> 并且接受 观察者的 注册
>
> 一旦 被观察者 中的数据 发生了 改变，ZooKeeper会通知 已经注册的 观察者
>
> 此时，观察者 可以做出相应的反应

- **ZooKeeper 在 分布式系统中 充当的是一个 辅助 的角色**

二、观察者模式

> 观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态上发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

```
public class MessageGroup {
    // 保存所有的订阅者
    private List<Subscriber> subscriberList = new ArrayList<>();
    /**
     * 发布情报
     * @param message
     */
    public void publishMessage(String message)
    {
        for(Subscriber s: subscriberList)
        {
            s.note(message);
        }
    }

    /**
```

```
     * 订阅者
     * @param subscriber
     */
    public void addSubscriber(Subscriber subscriber)
    {
        subscriberList.add(subscriber);
    }
}
```

```
public class Person implements Subscriber{
    private String name;
    public Person() {
    }
    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public void note(String message) {
        System.out.println(name + "收到了" + message);
    }
}
```

```
public interface Subscriber {
    /**
     * 消息处理方法
     * @param message
     */
    public void note(String message);

}
```

```
public class Test {
    public static void main(String[] args) {
        //<button>
        MessageGroup group = new MessageGroup();
        // function doCc()
        Person p1 = new Person("p1");
        Person p2 = new Person("p2");
        Person p3 = new Person("p3");

        // 让多个观察者对象同时监听，订阅某一个主题对象

        // btn.onclick = doCc
```

```
        group.addSubscriber(p1);
        group.addSubscriber(p2);
        group.addSubscriber(p3);

      // tom
        Scanner tom = new Scanner(System.in);

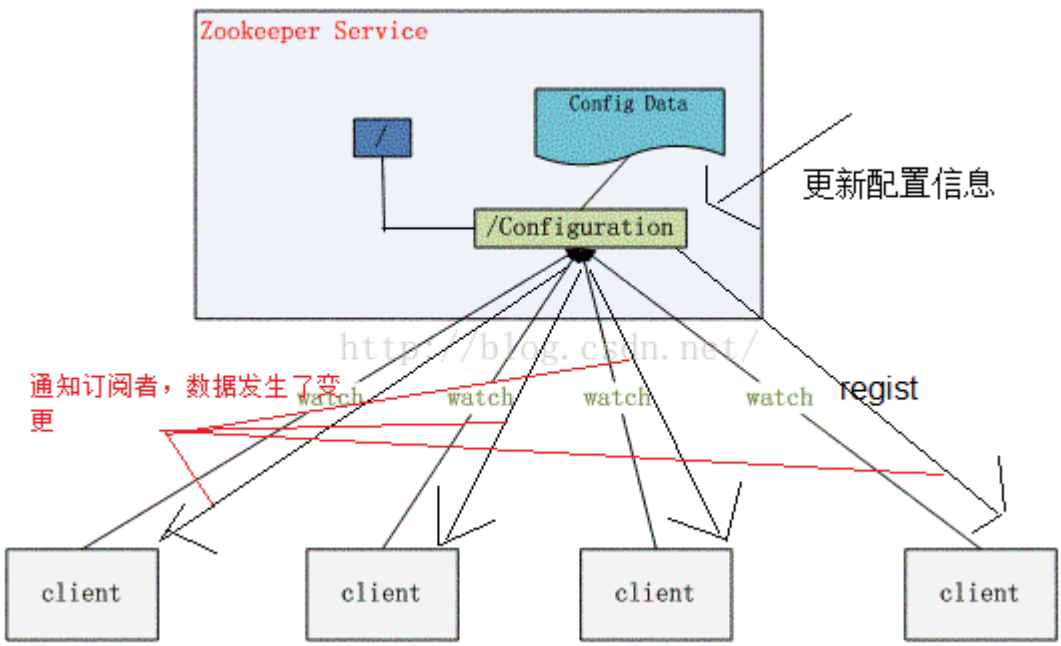
        while(true)
        {
            String msg = tom.nextLine();
            group.publishMessage(msg);
        }
    }
}
```

```
window.onload = function(){
  var btn= document.getElementById("btn");

  btn.addEventListener("click", function(){
    console.log(1111);
  })
  btn.addEventListener("click", function(){
    console.log(22222);
  })
  btn.addEventListener("click", function(){
    console.log(3333);
  })
}
```

三、应用场景

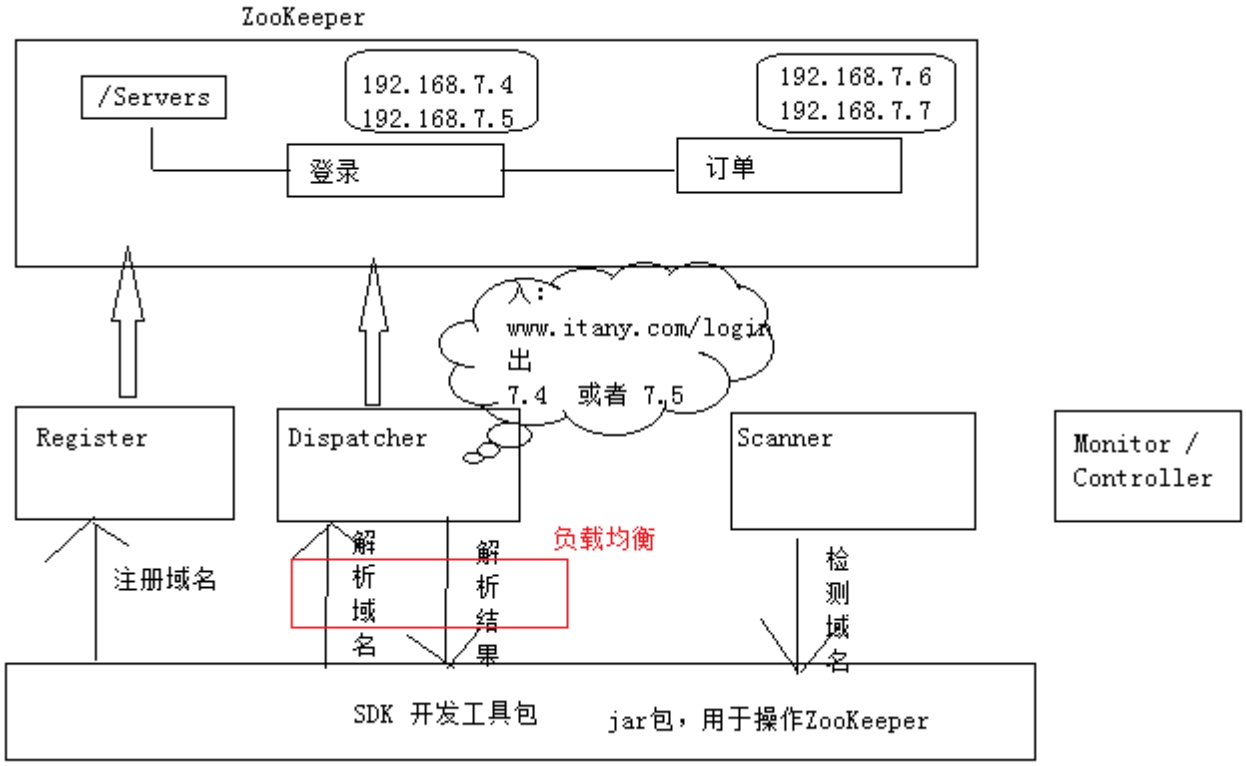
1、分布式消息同步和协调机制



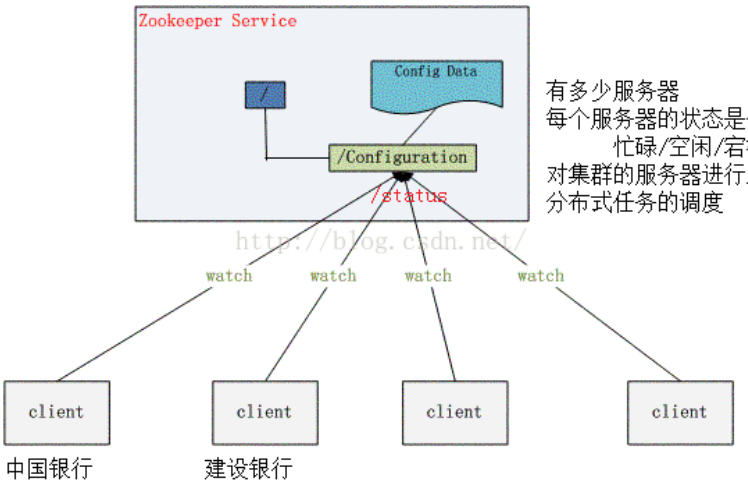
- 客户端（应用程序）启动的时候 主动的 到 ZooKeeper上获取配置信息，并且订阅配置信息（注册监听 Watcher）
- 当ZooKeeper节点（/Configuration）内容发生改变
- Zookeeper 将变更消息推送给所有的 观察者（Watcher）
- 观察者 自动调用 自己的 回调函数（CallBack，也就是上面的note方法）
- 观察者根据回调函数，主动更新 自己本地的 配置信息

2、负载均衡

- ZooKeeper-Register 负责域名的注册，即：集群下 所有的服务器域名（IP） 全部注册到ZooKeeper的域名服务下，
- ZooKeeper-Dispatcher 负责域名解析。（**此处实现了 负载均衡**）
- Scanner 通过定时的检测 服务器状态，动态的更新节点的地址信息



3、集群管理



有多少服务器
每个服务器的状态是什么
忙碌/空闲/宕机/下线
对集群的服务器进行上线/下线操作
分布式任务的调度

```
con.setAutoCommit(false)
con1.update(-100)
con2.update(100)
con.commit()
```

四、ZooKeeper安装

1、下载ZooKeeper

下载地址

2、安装ZooKeeper

解压缩下载包

```
tar -zxvf zookeeper.3.4.13.tar.gz
```

3、启动zk

3-1 创建存放数据文件的文件夹

```
zookeeper_home > mkdir data
```

3-2 创建zk配置文件

```
zookeeper_home > cd conf
# 文件名叫做zoo.cfg
# 在confg路径下，存在zoo.cfg 的sample（范例，样本），zoo_sample.cfg
zookeeper_home/conf > cp zoo_sample.cfg zoo.cfg
```

3-3 修改`zoo.cfg`配置

```
zookeeper_home/conf > vi zoo.cfg
#修改dataDir 指向3-1创建的文件夹
```

3-4 启动zk

```
zookeeper_home/bin > ./zkServer.sh start
```

3-5 测试

```
# 查看zk进程是否启动
> jps
 5335 QuorumPeerMain
```

五、常用命令

作用	命令
启动ZooKeeper	`./zkServer.sh start`
查看进程(启动后测试使用，查看ZooKeeper进程是否已经启动)	jps
查看ZooKeeper状态（启动后测试使用，查看ZooKeeper是否正常运行）	`./zkServer status`
启动客户端	./zkCli.sh
退出客户端	quit
停止ZooKeeper	./zkServer.sh stop

六、配置文件

```
# 通信心跳数，ZooKeeper服务器的心跳时间，单位是ms
# ZooKeeper使用的基本时间
# 服务器与客户端之间 、 服务器与服务器之间 维持心跳的时间间隔
# 也就是说 每隔 2s， 会发送一个心跳请求
# 用于ZooKeeper的心跳机制，并且session（会话）的超时时间 为 两倍心跳时间
#（即： maxSession = 2 * tickTime）
tickTime=2000
# LF初始通信时限
# 用于ZooKeeper集群（多个ZooKeeper集中在一起）
# 集群中的Follower 和 Leader 之间的初始连接时能容忍的最大心跳数（ticktime的数量）
# Follower 在启动过程中，会从Leader同步更新所有的数据，然后开始对外服务
# Leader 允许 Follower 在 initLimit * tickTime 时间内完成该项工作
# 一般情况下，该值 initLimit <= (tickTime * 2 * FollowerCount) / tickTime
# 即： 2 * FollowerCount
initLimit=10
#LF的同步通信时限
#集群中Leader 和 Follower 的最大响应时间
# 当Follower的响应时间 超过 syncLimit * tickTime 时，Leader认为该Follower宕机
# 此时。Leader 会从服务器列表中删除该Follower
syncLimit=5
# 数据目录，需要自己创建。此处指定目录地址
dataDir=../data
```

```
# 客户端连接端口号
clientPort=2181
#就是设置多少小时清理一次日志文件
autopurge.purgeInterval=3
#设置保留多少个snapshot 日志文件，之前的则删除。
autopurge.snapRetainCount=3
```

七、ZooKeeper数据结构

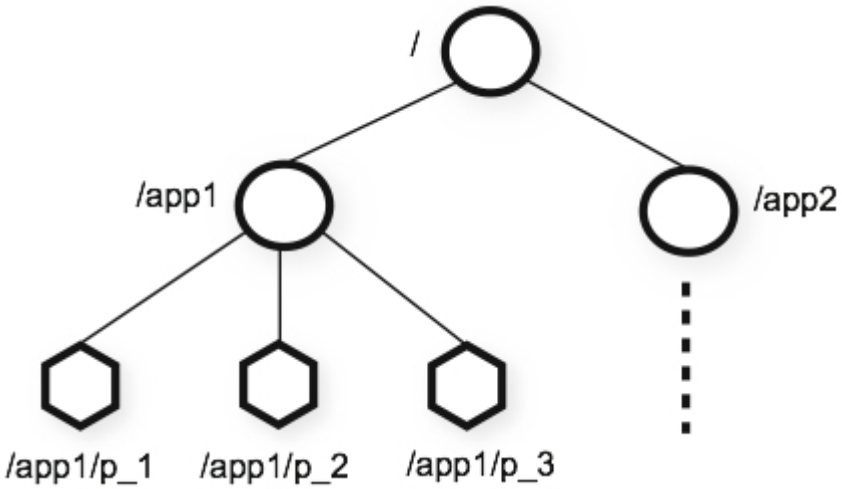
1、结构

ZooKeeper 数据模型结构 和 Linux 的文件系统相似，是一种 树形结构

每个 节点 叫做 ZNode （ ZooKeeper Node ）

每个ZNode 默认只能存储大约 1M 的数据

每个ZNode 可以通过其路径唯一的标识



2、关于ZNode

- 每个节点 唯一的 对应 一个 路径
- 每个节点 存储大约 1M的数据
- 节点类型
  - 短暂（ ephemeral ）：

    客户端(zkCli) 和服务器（ zkServer ）断开连接后，该类型节点会被自动删除
  - 持久（ persistent ）

    客户端(zkCli) 和服务器（ zkServer ）断开连接后，该类型节点不会删除
- ZNode 的四种形式的 目录节点
  - 持久化目录节点（ persistent ）

    客户端(zkCli) 和服务器（ zkServer ）断开连接后，该类型节点不会删除
  - 持久化顺序编号目录节点（ persistent_sequential ）

客户端(zkCli) 和服务器（zkServer）断开连接后，该类型节点不会删除

节点会被顺序编号（001 002 003.....）

编号会紧跟ZNode名称

- 短暂（临时）目录节点（ephemeral）

  客户端(zkCli) 和服务器（zkServer）断开连接后，该类型节点会被自动删除

- 短暂（临时）顺序编号目录节点（ephemeral_sequential）

  客户端(zkCli) 和服务器（zkServer）断开连接后，该类型节点会被自动删除

  节点会被顺序编号（001 002 003.....）

- 顺序编号 是一个递增的计数器
- 顺序编号由 父节点 维护
- 在分布式系统中，顺序号 可以被用于 为所有的事件进行全局排序，这样客户端就可以根据序号推断 事件的顺序

八、客户端命令行操作

1、查看所有的命令（help）

1. 随便乱输入命令
2. help

```
ZooKeeper -server host:port cmd args
  stat path [watch]
  set path data [version]
  ls path [watch]
  delquota [-n|-b] path
  ls2 path [watch]
  setAcl path acl
  setquota -n|-b val path
  history
  redo cmdno
  printwatches on|off
  delete path [version]
  sync path
  listquota path
  rmr path
  get path [watch]
  create [-s] [-e] path data acl
  addauth scheme auth
  quit
  getAcl path
  close
  connect host:port
```

2、查看某个节点下的子节点

```
ls 路径
```

3、查看当前znode中的数据信息

```
ls2 路径
```

```
# 子节点名称数组
[zookeeper]

#================== 以下所有信息叫做 stat  结构体 ============================
# 创建该znode的zxid，创建该znode 的事务的zxid （ZooKeeper Transaction ID）
# 事务ID ： 是ZooKeeper中所有修改（update/delete/insert）总的次序,每次修改都有一个唯一的id，值越小，表示越先执行
cZxid = 0x0
# 创建时间
ctime = Thu Jan 01 08:00:00 CST 1970
# 最后一次更新的zxid
mZxid = 0x0
# 最后一次更新的时间
mtime = Thu Jan 01 08:00:00 CST 1970
# 最后更新的子节点的zxid
pZxid = 0x0
# znode 子节点的变化号， znode 子节点的修改次数
cversion = -1
# znode 数据的变化号
dataVersion = 0
# access Control language 访问控制列表的变化号
aclVersion = 0
# 如果是临时节点，表示 znode 拥有者的 sessionId
# 如果不是临时节点，值是0
ephemeralOwner = 0x0
# 数据长度
dataLength = 0
# 子节点个数
numChildren = 1
```

4、创建普通节点

```
create 节点路径（节点名） 内容（如果没有空格，可以直接写，如果有空格，必须加双引号）
create /dahuzi "there are some dahuzis"
```

5、获取节点中的值

```
get 节点路径
# 节点内容
there are some dahuzis

# stat 结构体
```

```
cZxid = 0xb
ctime = Fri Sep 14 15:50:51 CST 2018
mZxid = 0xb
mtime = Fri Sep 14 15:50:51 CST 2018
pZxid = 0xb
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 22
numChildren = 0
```

6、创建短暂节点

```
create -e  节点路径  内容
create -e /dahuzi/zhangfei laozhang
quit
./zkCli.sh
ls /dahuzi
==> []
```

7、创建顺序编号节点

```
create -s 节点路径  内容
create -s /dahuzi/zhangfei laozhang
create -s /dahuzi/guanyu erge
```

编号从已有的子节点个数开始（包括临时节点和删除的节点）

8、修改节点内容

```
set 节点路径  新值
set /dahuzi "there are many dahuzis"
get /dahuzi
```

9、删除节点

```
delete 节点路径
delete /youdu/laoxie0000000000
```

只能删除空节点（没有子节点）

10、递归删除节点

```
rmr 节点路径

rmr /book
```

11、查看节点状态

查看stat 结构体

```
stat  节点路径
stat /dahuzi
```

九、ZooKeeper集群

1、安装、启动ZooKeeper

> 在7.24 25 26 分别安装，配置了 ZooKeeper

2、配置ZooKeeper

在`[zookeeper_home]/conf/zoo.cfg`中添加如下格式内容

```
server.24=192.168.7.24:2888:3888
server.25=192.168.7.25:2888:3888
server.26=192.168.7.26:2888:3888
```

格式:

`server.A=B:C:D`

- A 是一个数字，表示这是第几号服务器
- B 是服务器的IP地址或者 域名（www.serverA.com/192.168.x.x）
- C 是这个服务器与集群中的Leader交换信息的端口
- D 是 执行选举Leader服务器相互通信的端口

3、编写myid配置文件

在集群环境下，需要在`dataDir`目录下，配置`myid`文件，文件内容是 当前服务器对应的上马的A的值（即：该服务器的服务器编号）

```
#dataDir 目录下执行
echo A的值 >> myid
```

4、测试集群环境

```
./zkServer.sh start
./zkServer.sh start
./zkServer.sh start
./zkServer.sh status
./zkServer.sh status
./zkServer.sh status
```

十、集群特性

1. 一个ZooKeeper集群，有一个领导者（Leader），和多个 跟随者（Follower）
2. Leader 负责进行投票的发起和决议，更新系统状态
3. Follower 用于接收客户端请求，并向客户端返回结果，在选举Leader过程中参与投票
4. **半数机制：**集群中只要有半数以上的节点存活，ZooKeeper集群就能正常服务

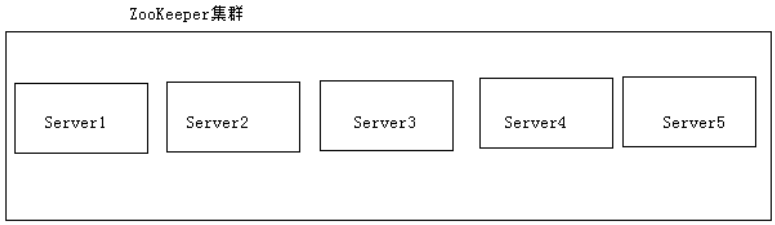
   一般情况下，集群数为奇数
5. 全局数据一致：每个ZooKeeper服务器保存一份相同的副本，即：不论客户端连接到集群中的哪个服务器，数据都是一致的
6. 更新请求顺序执行，来自同一个客户端的更新请求按顺序依次执行
7. 数据更新的原子性：一次数据更新，要么成功，要么失败
8. 实时性：在一定时间范围内，客户端能读到最新数据

十一、选举机制

1. **半数机制：**集群中只要有半数以上的节点存活，ZooKeeper集群就能正常服务

   一般情况下，集群数为奇数
2. ZooKeeper 在配置文件中（`zoo.cfg`），并没有指定 Leader（Master），和 Follower（Slaver）。但是，ZooKeeper在提供服务过程中，自动`选举`了一个节点作为Leader，其他服务器都是 Follower。
3. 选举机制:

```
选举机制
        假设在集群中存在 5 台 服务器
        id值 1 - 5 ，
                server.1=
                server.2=
                ....
                server.5=
        假设是最新启动，并且从1 到 5 依次启动
```



   1. Server1启动，此时，只有一台服务器启动，它发出的报文（消息）没有任何响应，所以此时选举状态为Locking
   2. Server2 启动，它与Server1通信，互换自己的选举结果。由于Server2 的id 值较大，所以Server2 胜出，**但是，由于没有达到半数以上的服务器同意选举Server2**，此时Sever1 和 Server2 都处于 Locking 状态
   3. Server3 启动，它与Server1 / Server2通信，互换自己的选举结果，由于Server3 的id值较大，所以，Server3 胜出，**此时，票数已经过半。**所以 Server3 成为Leader
   4. Server4 启动，理论上，Server4 id 值大于1,2,3 ， Server4 应该成为Leader，但是，**Server1,2,3 已经选举了 Server3，**所以Server4 只能是 Follower
   5. Server5 同 Server4

概括为一句话：启动顺序中，前（集群数/2+1）个服务器中，id值最大的

十二、集群操作

1、节点值变化监听

```
# 在集群中的A服务器观察某个节点值的变化
# 监听事件只会触发一次
get /stus watch

# 在集群的B服务器修改对应节点值
set /stus abc

#此时A服务器会收到NodeDataChanged事件
WATCHER::
WatchedEvent state:SyncConnected type:NodeDataChanged path:/stus
```

2、节点的子节点变化监听

```
# 在集群中的A服务器观察某个节点的子节点的变化
# 监听事件只会触发一次
ls /stus watch
# 在集群的B服务器修改（创建和删除）对应节点的子节点
create /stus/laoxie youdu
#此时A服务器会收到NodeChildrenChanged事件
WATCHER::
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/stus
```

十三、使用Java操作ZooKeeper

1、添加jar包

```
<!--ZooKeeper 客户端核心包-->
<dependency>
  <groupId>org.apache.zookeeper</groupId>
  <artifactId>zookeeper</artifactId>
  <version>3.4.13</version>
</dependency>
```

2、编写代码

```
public class HelloWorld {

    private  static ZooKeeper zkClient;

    static {
       try {
           // 连接的服务器地址，多个地址之间逗号分隔，地址包括：ip：端口
           String connectString = "localhost:2181";
           // 会话的超时时间，单位毫秒
           int sessionTimeout = 8000;
           // 创建ZooKeeper客户端
           zkClient = new ZooKeeper(connectString,
                                    sessionTimeout,
```

```
                                    new ZooKeeperEventWatcher());
        }catch (Exception t)
        {
            t.printStackTrace();
        }
    }

    public static void ls() throws KeeperException, InterruptedException {
        // 查看某个节点下的子节点 ls /  ls / watch
        List<String> children = zkClient.getChildren("/", false);
        System.out.println(children);
    }

    public static void create(String path,String value) throws KeeperException,
InterruptedException {
        zkClient.create(
          path,
          value.getBytes(),
          ZooDefs.Ids.OPEN_ACL_UNSAFE,
          CreateMode.PERSISTENT
        );
    }

    public static void get(String path)
      throws KeeperException, InterruptedException {
        Stat stat = new Stat();
        byte[] buffer = zkClient.getData(path,false,stat);
        String value = new String(buffer);
        System.out.println(value);
        System.out.println(stat);
    }
    public static void set(String path,String value)
      throws KeeperException, InterruptedException {
        // 当前数据版本
//       zkClient.setData(path,value.getBytes(),1);
        Stat s = new Stat();
        zkClient.getData(path,false,s);
        zkClient.setData(path,value.getBytes(),s.getVersion());

    }

    public static void delete(String path)
      throws KeeperException, InterruptedException {
//        zkClient.delete(path,0);
        Stat s = new Stat();
        zkClient.getData(path,false,s);
        zkClient.delete(path,s.getVersion());
    }

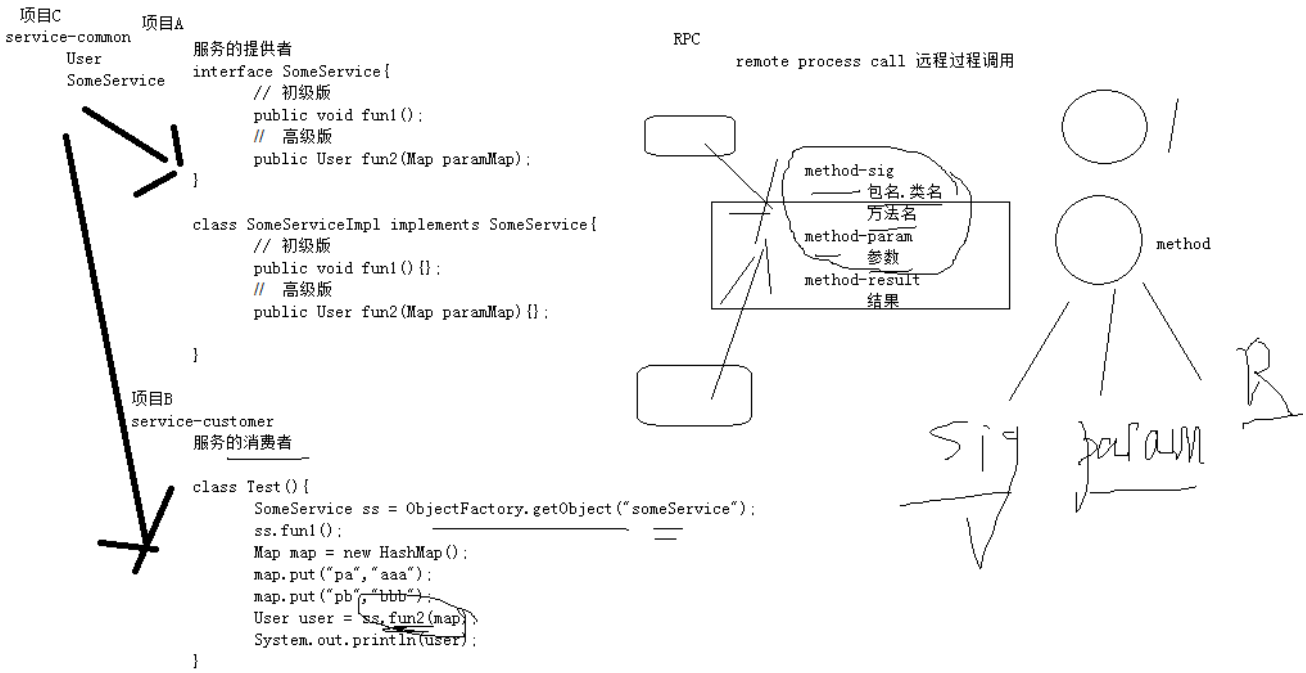

    public static void main(String[] args)
      throws IOException, KeeperException, InterruptedException {

        ls();
```

```
        System.out.println("-----------");
//        create("/laoxie","youdu");
//        ls();
//        get("/laoxie");
//        set("/laoxie","222");
//        delete("/lx");
    }
}
```

十四、使用ZooKeeper实现RPC

1、实现原理



2、步骤

1. 创建三个Maven项目

```
service-common
service-provider
        依赖common
        添加ZooKeeper 依赖  fastjson 依赖
service-customer
        依赖common
        添加ZooKeeper 依赖  fastjson 依赖
```

2. 编写service-common

```
interface SomeService{
  public User fun(Map map);
}
class User{
  private String name;
  private String address;
}
```

3. 编写service-provider

```
//SomeServiceImpl
public class SomeServiceImpl implements SomeService {
    @Override
    public User fun(User user) {
        System.out.println("方法被调用");
        User u = new User();
        u.setName("new" + user.getName());
        u.setAddress("new" + user.getAddress());
        return u;
    }
}
//Server
public class Server {
    private  static final String METHOD_CALL_PATH = "/method/call";
    public  static final String METHOD_RESULT_PATH = "/method/result";

  public  void start() throws KeeperException, InterruptedException {
        new Thread(){
            @Override
            public void run() {
                try {
                    initWatch();
                } catch (KeeperException e) {
                    e.printStackTrace();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }.start();

        synchronized (this){
            this.wait();
        }
    }

    protected   static void initWatch()
      throws KeeperException, InterruptedException {
        ZkUtil.watch(METHOD_CALL_PATH,new MethodWater());
    }

}
```

```
class MethodWater implements Watcher{

    @Override
    public void process(WatchedEvent event) {
        try {
            String path = event.getPath();
            String value = ZkUtil.getValue(path);
            MethoVO vo = JSON.parseObject(value,MethoVO.class);
            System.out.println("客户端请求调用:" + vo.getClsName() + "." +
vo.getMethodName() );
            System.out.println("参数是:" + Arrays.toString(vo.getParams()));

            //此处应该使用IOC容器
            SomeService service = new SomeServiceImpl();

            User u = JSON.parseObject(((JSONObject)vo.getParams()
[0]).toJSONString(),User.class);

            User result = service.fun(u);

            String resultJSON = JSON.toJSONString(result);
            ZkUtil.setValue(Server.METHOD_RESULT_PATH,resultJSON);

            Server.initWatch();
        }catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
//ZkUtil
public class ZkUtil {

    private static String connectString = "localhost:2181";

    private static ZooKeeper zkClient ;

    static{
        try {
            zkClient = new ZooKeeper(connectString,4000,null);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void watch(String path, Watcher watcher) throws KeeperException,
InterruptedException {
        zkClient.getData(path,watcher,new Stat());
    }
```

```
   public static String getValue(String path) throws KeeperException,
InterruptedException {
       return new String(zkClient.getData(path,true,new Stat()));
   }

   public static void setValue(String path,String value) throws KeeperException,
InterruptedException {
       Stat st = new Stat();
       zkClient.getData(path,true,st);
       zkClient.setData(path,value.getBytes(),st.getVersion());
   }
```

} //MethoVO public class MethoVO { private String clsName; private String methodName; private Object[] params; }

```
4. 编写service-customer

```java
public class Application {
    public static SomeService get()
    {
        return (SomeService) Proxy
          .newProxyInstance(
                      Application.class.getClassLoader(),
                new Class[]{SomeService.class},
                new InvocationHandler() {
                    private String value;

                    @Override
                    public Object invoke(Object proxy,
                                         Method method, Object[] args)
                      throws Throwable {
                        MethoVO vo = new MethoVO();
                        vo.setClsName("com.itany.SomeService");
                        vo.setMethodName(method.getName());
                        vo.setParams(args);

                        String str = JSON.toJSONString(vo);

                        ZkUtil.setValue("/method/call",str);
;
                        ZkUtil.watch("/method/result", new Watcher() {
                            @Override
                            public void process(WatchedEvent event) {
                                String path = event.getPath();
                                try {
                                    value = ZkUtil.getValue(path);

                                } catch (KeeperException e) {
                                    e.printStackTrace();
                                } catch (InterruptedException e) {
                                    e.printStackTrace();
```
```

```
                                }
                            }
                        });
                        Thread.sleep(2000);
                        return JSON.parseObject(value,User.class);
                    }
                }
        );
    }
}
public class Test {
    public static void main(String[] args) {
        SomeService service = Application.get();
        User user = new User();
        user.setName("laoxie");
        user.setAddress("youdu");
        User temp = service.fun(user);
        System.out.println(temp);
    }
}
```

ZooKeeper 实现RPC 原理