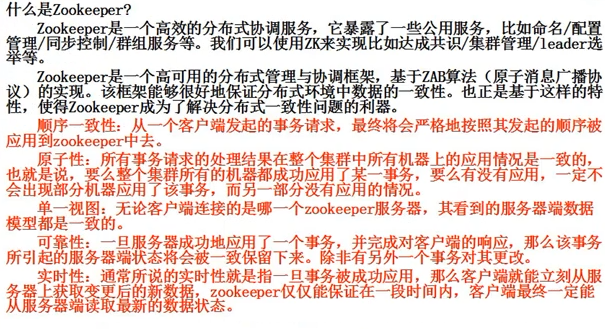
Zookeeper

# 基础知识

## 概念介绍



## Zookeeper的集群配置

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

**1)ssZookeeper的部署方案**

|  |  |  |  |
| --- | --- | --- | --- |
| 主机IP | 消息端口号 | 通信端口号 | 节点目录/usr/local/sofeware |
| 192.168.118.6 | 2181 | 2888:3888 | Zookeeper |
| 192.168.118.9 | 2181 | 2888:3888 | Zookeeper |
| 192.168.118.10 | 2181 | 2888:3888 | Zookeeper |

**2)把相关的程序进行安装**

2.1）上传文件zookeeper-3.4.11.tar.gz 文件到/usr/local/sofeware目录下

2.2）解压文件

tar -zxvf zookeeper-3.4.11.tar.gz -C /usr/local/sofeware/

2.3）修改文件夹的名称

mv zookeeper-3.4.11 zookeeper

**3）修改相关的配置文件**

Vim /etc/profile

三个服务器上都进行相关的修改即可

export ZOOKEEPER\_HOME=/usr/local/sofeware/zookeeper

export PATH=.:$JAVA\_HOME/bin:$ZOOKEEPER\_HOME/bin:$PATH

让配置文件立即生效

source /etc/profile

**4）修改Zookeeper相关的配置文件**

[root@J2001 sofeware]# ***cd zookeeper***

[root@J2001 zookeeper]# **ll**

总用量 1596

drwxr-xr-x. 2 502 games 149 11月 2 02:52 bin

-rw-r--r--. 1 502 games 87943 11月 2 02:47 build.xml

drwxr-xr-x. 2 502 games 77 11月 2 02:52 conf

drwxr-xr-x. 10 502 games 130 11月 2 02:47 contrib

drwxr-xr-x. 2 502 games 4096 11月 2 02:54 dist-maven

drwxr-xr-x. 6 502 games 4096 11月 2 02:52 docs

-rw-r--r--. 1 502 games 1709 11月 2 02:47 ivysettings.xml

-rw-r--r--. 1 502 games 8197 11月 2 02:47 ivy.xml

drwxr-xr-x. 4 502 games 4096 11月 2 02:52 lib

-rw-r--r--. 1 502 games 11938 11月 2 02:47 LICENSE.txt

-rw-r--r--. 1 502 games 3132 11月 2 02:47 NOTICE.txt

-rw-r--r--. 1 502 games 1585 11月 2 02:47 README.md

-rw-r--r--. 1 502 games 1770 11月 2 02:47 README\_packaging.txt

drwxr-xr-x. 5 502 games 47 11月 2 02:47 recipes

drwxr-xr-x. 8 502 games 211 11月 2 02:52 src

-rw-r--r--. 1 502 games 1478279 11月 2 02:49 zookeeper-3.4.11.jar

-rw-r--r--. 1 502 games 195 11月 2 02:52 zookeeper-3.4.11.jar.asc

-rw-r--r--. 1 502 games 33 11月 2 02:49 zookeeper-3.4.11.jar.md5

-rw-r--r--. 1 502 games 41 11月 2 02:49 zookeeper-3.4.11.jar.sha1

[root@J2001 zookeeper]# **cd conf**

[root@J2001 conf]# **ll**

总用量 12

-rw-r--r--. 1 502 games 535 11月 2 02:47 configuration.xsl

-rw-r--r--. 1 502 games 2161 11月 2 02:47 log4j.properties

-rw-r--r--. 1 502 games 922 11月 2 02:47 zoo\_sample.cfg

[root@J2001 conf]#  **mv zoo\_sample.cfg zoo.cfg**

[root@J2001 conf]# **ll**

总用量 12

-rw-r--r--. 1 502 games 535 11月 2 02:47 configuration.xsl

-rw-r--r--. 1 502 games 2161 11月 2 02:47 log4j.properties

-rw-r--r--. 1 502 games 922 11月 2 02:47 zoo.cfg

[root@J2001 conf]# **vim zoo.cfg**

修改

**dataDir=/usr/local/sofeware/zookeeper/data**

添加

**server.0=192.168.118.6:2888:3888**

**server.1=192.168.118.9:2888:3888**

**server.2=192.168.118.10:2888:3888**

[root@J2001 conf]# **cd ..**

[root@J2001 zookeeper]# **mkdir data**

[root@J2001 zookeeper]# **ll**

[root@J2001 data]# vim myid

在**server.0=192.168.118.6 这个主机的文件myid中添加 一个 0**

在**server.1=192.168.118.9 这个主机的文件myid中添加 一个 1**

在**server.2=192.168.118.10 这个主机的文件myid中添加 一个 2**

**五)启动服务**

[root@J2001 data]# **zkServer.sh start 启动zookeeper服务**

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Starting zookeeper ... STARTED

[root@J2001 data]# **zkServer.sh status 查看状态**

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Error contacting service. **It is probably not running.**

[root@J2001 data]# **systemctl stop firewalld 关闭防火墙**

[root@J2001 data]# **zkServer.sh restart 重启zookeeper服务器**

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Stopping zookeeper ... STOPPED

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Starting zookeeper ... STARTED

[root@J2001 data]# **zkServer.sh status 查看状态**

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Mode: **follower**

[root@J2001 data]# zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /usr/local/sofeware/zookeeper/bin/../conf/zoo.cfg

Mode: **leader**

## 配置文件的详解

**tickTime：** 基本事件单元，以毫秒为单位。这个时间是作为 Zookeeper服务器之间或客户端与服务器之间维持**心跳的时间间隔**，也就是每隔 tickTime时间就会发送一个心跳。

**initLimit：** 这个配置项是用来配置 Zookeeper接受客户端初始化连接时最长能忍受多少个心跳时间间隔数，当已经超过 10 个心跳的时间（也就是 tickTime）长度后Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是10\*2000=20 秒。

**syncLimit**： 这个配置项标识 Leader 与 Follower之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime的时间长度，总的时间长度就是 5\*2000=10 秒

**dataDir**： 存储内存中数据库快照的位置，顾名思义就是 Zookeeper保存数据的目录，默认情况下， Zookeeper将写数据的日志文件也保存在这个目录里。 /tmp/zookeeper   
修改为了dataDir=/usr/local/sofeware/zookeeper/data

**clientPort**： 这个端口就是客户端连接 Zookeeper 服务器的端口2181， Zookeeper会监听这个端口，接受客户端的访问请求。

**server.A = B:C:D :**

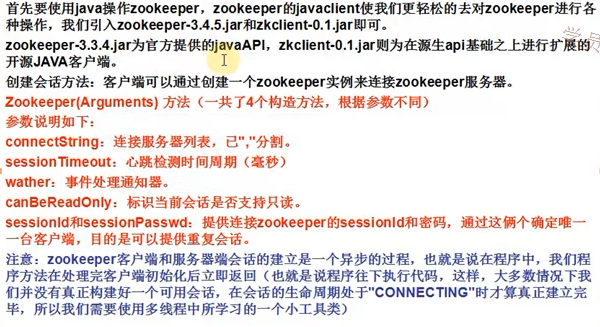
A表示这个是第几号服务器,

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；2888**

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 3888**，选出一个新的 Leader

# Zookeeper的使用



## Zookeeper原生态API的使用

### 基础用法

**package** com.guigu.zookeeper.base;

/\*\*

\* <p>project\_name:ZookeeperStudy</p>

\* <p>package\_name:com.guigu.zookeeper.base.ZookeeperBase</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月9日上午11:04:42 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**import** java.util.concurrent.CountDownLatch;

**import** org.apache.zookeeper.WatchedEvent;

**import** org.apache.zookeeper.Watcher;

**import** org.apache.zookeeper.Watcher.Event.EventType;

**import** org.apache.zookeeper.Watcher.Event.KeeperState;

**import** org.apache.zookeeper.ZooKeeper;

**public** **class** ZookeeperBase {

// 1.配置Zookeeper的地址

**public** **static** **final** String ***CONNECT\_ADDR*** = "192.168.118.6:2181,192.168.118.9:2181,192.168.118.10:2181";

// 2.配置Session超时连接时间

**public** **static** **int** *SESSION\_TIMEOUT* = 2000;// ms

// 3.阻塞程序的执行 信号量 用于等待Zookeeper的成功连接 发送成功信息

**public** **static** **final** CountDownLatch ***connectedSemaphor*** = **new** CountDownLatch(1);

**public** **static** **void** main(String[] args) **throws** Exception {

ZooKeeper zk =**new** ZooKeeper(***CONNECT\_ADDR***, *SESSION\_TIMEOUT*, **new** Watcher() { //创建连接并监听

@Override

**public** **void** process(WatchedEvent event) {

//获取事件的状态

KeeperState keeperState =event.getState();

EventType eventType =event.getType();

// try {

// Thread.sleep(5000);

// } catch (InterruptedException e) {

// e.printStackTrace();

// }

//如果是建立Zookeeper的连接

**if**(keeperState.***SyncConnected***==keeperState) { //如果是连接Zookeeper

**if**(EventType.***None***==eventType) {//第一次连接Zookeeper

***connectedSemaphor***.countDown(); //放行程序进行下一步的操作 否则禁止下一步的执行

System.***out***.println("zookeeper建立连接成功....");

}

}

}

});

//如果Zookeeper没有建立连接 阻塞程序的执行....

***connectedSemaphor***.await();

//...其他操作

System.***out***.println(".....");

///在Zookeeper上创建相关的服务节点

/\*\*

\* 1.创建父节点

\* 参数1 是节点的名字

\* 参数2 是节点的数据信息

\* 参数3 权限认证相关

\* 参数4 节点的类型

\*/

// zk.create("/testRoot", "testRootHzguigu".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

//创建子节点

// zk.create("/testRoot/children", "hellozookeeper".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

//获取节点的信息

**byte** [] data =zk.getData("/testRoot/children", **false**, **null**);

System.***out***.println(**new** String(data));

//获取子节点的名称

System.***out***.println(zk.getChildren("/testRoot", **false**));

//修改节点信息

// zk.setData("/testRoot", "data modify root test".getBytes(), -1);

//判断节点是否存在

System.***out***.println(zk.exists("/testRoot/children", **false**));

//删除节点

zk.delete("/testRoot/children", -1);

//关闭客户端连接

zk.close();

}

}

### Watcher监听、zk状态、事件类型

Zookeeper有Watcher事件， 是一次性触发的，当watch监视的数据发生改变时，通知设置watch的client 即Watcher。

同样其watcher是监听数据发送了某些变化，那么一定会有对应的事件类型和状态类型

**常见的Zookeeper的事件类型包括**： (Znode节点相关的事件)

EventType.NodeCreated

EventType.NodeDataChange

EventType.NodeChildrenChange

EventType.NodeDelete

EventType.None

**常见的Zookeeper的状态类型(是跟客户端实例相关)**

keeperState.SyncConnected 初始化连接

keeperState.AuthFailed

keeperState.Disconnected

keeperState.Expired

keeperState.ConnectedReadOnly

**package** com.guigu.zookeeper.watcher;

**import** java.io.IOException;

**import** java.util.List;

**import** java.util.concurrent.CountDownLatch;

**import** java.util.concurrent.atomic.AtomicInteger;

**import** org.apache.zookeeper.CreateMode;

**import** org.apache.zookeeper.KeeperException;

**import** org.apache.zookeeper.WatchedEvent;

**import** org.apache.zookeeper.Watcher;

**import** org.apache.zookeeper.ZooDefs.Ids;

**import** org.apache.zookeeper.ZooKeeper;

**import** org.apache.zookeeper.Watcher.Event.EventType;

**import** org.apache.zookeeper.Watcher.Event.KeeperState;

**import** org.apache.zookeeper.data.Stat;

/\*\*

\* <p>project\_name:ZookeeperStudy</p>

\* <p>package\_name:com.guigu.zookeeper.watcher.ZookeeperWatcher</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月9日下午1:40:27 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**public** **class** ZookeeperWatcher **implements** Watcher {

/\*\*定义原子变量\*\*/

AtomicInteger seq = **new** AtomicInteger();

/\*\* 配置Zookeeper的地址\*\*/

**public** **static** **final** String ***CONNECT\_ADDR*** = "192.168.118.6:2181,192.168.118.9:2181,192.168.118.10:2181";

/\*\*配置Session超时连接时间\*\*/

**public** **static** **int** *SESSION\_TIMEOUT* = 2000;// ms

/\*\*zookeeper父路径的设置\*\*/

**private** **static** **final** String ***PARETN\_PATH*** = "/testWatch";

/\*\*zookeeper子路径的设置\*\*/

**private** **static** **final** String ***CHILREN\_PATH*** = "/testWatch/children";

/\*\*进入的标识\*/

**private** **static** **final** String ***LOG\_PREFIX\_OF\_MAIN*** = "【main】";

/\*\*zookeeper变量\*/

**private** ZooKeeper zk = **null**;

/\*\*阻塞程序的执行 信号量 用于等待Zookeeper的成功连接 发送成功信息\*/

**public** **static** **final** CountDownLatch ***connectedSemaphor*** = **new** CountDownLatch(1);

/\*\*

\* 创建Zookeeper的连接

\*/

**public** **void** createConnection(String connectAddr, **int** sessionTimeout) {

realseConnection();

**try** {

zk = **new** ZooKeeper(***CONNECT\_ADDR***, *SESSION\_TIMEOUT*, **this**);

System.***out***.println(***LOG\_PREFIX\_OF\_MAIN*** + "开始连接Zookeeper服务器");

***connectedSemaphor***.await();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

/\*\*

\* 关闭Zookeeper连接

\*/

**public** **void** realseConnection() {

**if** (zk != **null**) {

**try** {

zk.close();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 创建节点的监听

\*/

**public** **boolean** createPath(String path, String data) {

// 设置监控(由于Zookeeper的监听都是一次性的 所有每次都必须手动设置监听)

**try** {

**this**.zk.exists(path, **true**);

// 把创建的节点信息在控制台输出...

System.***out***.println(***LOG\_PREFIX\_OF\_MAIN*** + "节点创建成功,Path:"

// 路径 数据 权限:所有人都可见 存储类型：永久存储

+ zk.create(path, data.getBytes(), Ids.***OPEN\_ACL\_UNSAFE***, CreateMode.***PERSISTENT***) + ",content:"

+ data);

} **catch** (KeeperException e) {

e.printStackTrace();

} **catch** (InterruptedException e) {

e.printStackTrace();

**return** **false**;

}

**return** **true**;

}

/\*\*

\* 读取指定节点的数据内容

\*/

**public** String readData(String path, **boolean** needWatcher) {

**try** {

**return** **new** String(zk.getData(path, needWatcher, **null**));

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

e.printStackTrace();

**return** "";

}

}

/\*\*

\* 更新指定节点的数据内容

\*/

**public** **boolean** writeData(String path, String data) {

**try** {

System.***out***.println(

***LOG\_PREFIX\_OF\_MAIN*** + "更新数据成功...path:" + path + ",state:" + zk.setData(path, data.getBytes(), -1));

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

e.printStackTrace();

}

**return** **false**;

}

/\*\*

\* 判断节点是否存在

\*/

**public** Stat exists(String path, **boolean** needWatch) {

**try** {

**return** zk.exists(path, needWatch);

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

e.printStackTrace();

**return** **null**;

}

}

/\*\*

\* 删除指定节点

\*/

**public** **void** deleteNode(String path) {

**try** {

zk.delete(path, -1);

System.***out***.println(***LOG\_PREFIX\_OF\_MAIN***+"删除节点成功....Path:"+path);

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

e.printStackTrace();

}

}

/\*\*

\* 获取子节点信息

\*/

**public** List<String> getChildren(String path,**boolean** needWatch){

**try** {

**return** zk.getChildren(path, needWatch);

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

e.printStackTrace();

**return** **null**;

}

}

/\*\*

\* 删除所有节点

\*/

**public** **void** deleteAllTestPath() {

//如果子节点存在 先删除子节点 然后删除父节点

**if**(**this**.exists(***CHILREN\_PATH***, **false**)!=**null**) {

**this**.deleteNode(***CHILREN\_PATH***);

}

//删除父节点

**if**(**this**.exists(***PARETN\_PATH***, **false**)!=**null**) {

**this**.deleteNode(***PARETN\_PATH***);

}

}

@Override

**public** **void** process(WatchedEvent event) {

System.***out***.println("进入Process...... Event:" + event);

**if** (event == **null**) {

**return**;

}

// 连接状态

KeeperState keeperState = event.getState();

// 事件类型

EventType eventType = event.getType();

// 受影响的Path

String path = event.getPath();

// 打印相关的数据

// 日志前缀信息

String logPrefix = "【Watcher---" + seq.incrementAndGet() + "】";

System.***out***.println(logPrefix + "手动Watcher通知");

System.***out***.println(logPrefix + "当前连接状态:\t" + keeperState.toString());

System.***out***.println(logPrefix + "事件类型:\t" + eventType.toString());

**if** (keeperState.***SyncConnected*** == keeperState) {

// 成功连接上Zookeeper服务器

**if** (eventType.***None*** == eventType) {

System.***out***.println(logPrefix + "成功连接到Zookeeper服务器上...");

***connectedSemaphor***.countDown();

}

// 创建节点

**else** **if** (eventType.***NodeCreated*** == eventType) {

System.***out***.println(logPrefix + "节点创建.....");

// 模拟监听到节点创建之后的操作...

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**this**.exists(path, **true**);

}

// 更新节点

**else** **if** (EventType.***NodeDataChanged*** == eventType) {

System.***out***.println(logPrefix + "节点数据更新被监听到....");

System.***out***.println("我就是看看是否简单，简单在学点.....");

// 模拟监听到节点更新之后的操作...

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println(logPrefix + "数据内容变更为：" + **this**.readData(***PARETN\_PATH***, **true**));

}

// 更新子节点

**else** **if**(EventType.***NodeChildrenChanged***==eventType) {

System.***out***.println(logPrefix+"子节点已经更新....");

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("子节点列表是:"+**this**.readData(***PARETN\_PATH***, **true**));

}

// 删除子节点

**else** **if**(EventType.***NodeDeleted***==eventType) {

System.***out***.println(logPrefix+"节点"+path+"被删除");

}

} **else** **if** (KeeperState.***Disconnected*** == keeperState) {

System.***out***.println(logPrefix + "与Zookeeper断开连接....");

} **else** **if** (KeeperState.***AuthFailed*** == keeperState) {

System.***out***.println(logPrefix + "权限检查失败...");

} **else** **if** (KeeperState.***Expired*** == keeperState) {

System.***out***.println(logPrefix + "会话失败.....");

} **else**

;

System.***out***.println("--------------------");

}

**public** **static** **void** main(String[] args) **throws** Exception {

// 建立Watcher监听

ZookeeperWatcher zkWatcher = **new** ZookeeperWatcher();

// 创建Zookeeper的连接

zkWatcher.createConnection(***CONNECT\_ADDR***, *SESSION\_TIMEOUT*);

System.***out***.println(zkWatcher.zk.toString());

Thread.*sleep*(1000);

//清理节点

zkWatcher.deleteAllTestPath();

// 开启不同的操作 测试监听是否起作用

**if** (zkWatcher.createPath(***PARETN\_PATH***, System.*currentTimeMillis*() + "")) {

System.***out***.println("创建成功...监听已经执行....");

// 读取数据

System.***out***.println("----------read parent data---------------");

System.***out***.println(zkWatcher.readData(***PARETN\_PATH***, **true**));

System.***out***.println("----------read parent data over---------------");

//得到子节点路径

System.***out***.println(zkWatcher.getChildren(***PARETN\_PATH***, **true**));

//更新数据

zkWatcher.writeData(***PARETN\_PATH***, System.*currentTimeMillis*()+"");

//创建子节点

zkWatcher.createPath(***CHILREN\_PATH***, System.*currentTimeMillis*()+"");

System.***out***.println(zkWatcher.getChildren(***PARETN\_PATH***, **true**));

Thread.*sleep*(3000);

}

zkWatcher.realseConnection();

}

}

## ZkClient 的使用

### ZkClient基础用法

ZkClient是在原生api基础上机型封装简化了Zk的复杂性

**package** com.guigu.zookeeper.zkclient;

**import** java.util.List;

**import** org.I0Itec.zkclient.ZkClient;

**import** org.I0Itec.zkclient.ZkConnection;

/\*\*

\* <p>project\_name:ZookeeperStudy</p>

\* <p>package\_name:com.guigu.zookeeper.zkclient.ZkClientBase</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月14日上午8:48:04 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**public** **class** ZkClientBase {

/\*\*Zookeeper地址\*/

**public** **static** **final** String ***CONNECT\_ADDR*** = "192.168.118.6:2181,192.168.118.9:2181,192.168.118.10:2181";

// 2.配置Session超时连接时间

**public** **static** **int** *SESSION\_TIMEOUT* = 2000;// ms

**public** **static** **void** main(String[] args) {

ZkClient zkc =**new** ZkClient(**new** ZkConnection(***CONNECT\_ADDR***), *SESSION\_TIMEOUT*);

//1.创建节点

// zkc.createEphemeral("/temp");

// zkc.createPersistent("/super/hzguigu",true);

//2.删除节点信息

// zkc.delete("/temp");

// zkc.deleteRecursive("/super");

//3.设置path 和data数据 并且读取子节点和每个子节点的内容

// zkc.createPersistent("/super","hello");

// zkc.createPersistent("/super/hzguigu","hzguiguabc");

// zkc.createPersistent("/super/guigu","guiguqwe");

//获取所有节点的内容

// List<String> list =zkc.getChildren("/super");

//

// for (String str : list) {

// String rp="/super/"+str;

// String data="/super/"+zkc.readData(rp);

// System.out.println("节点为:"+rp+",数据内容是:"+data);

// }

//4.更新和判断节点是否存在

zkc.writeData("/super/guigu", "newdataguigu");

Object c =zkc.readData("/super/guigu");

System.***out***.println(c);

System.***out***.println(zkc.exists("/super/hzguigu"));

//递归删除节点

// zkc.delete("");//删除一个节点

zkc.deleteRecursive("/super");//递归删除多个节点

}

}

### Watcher监听

#### 子节点的监听

**package** com.guigu.zookeeper.zkclient.watcher;

**import** java.util.List;

**import** org.I0Itec.zkclient.IZkChildListener;

**import** org.I0Itec.zkclient.IZkConnection;

**import** org.I0Itec.zkclient.ZkClient;

**import** org.I0Itec.zkclient.ZkConnection;

/\*\*

\* <p>project\_name:ZookeeperStudy</p>

\* <p>package\_name:com.guigu.zookeeper.zkclient.watcher.ZkClientWatcher1</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月14日上午9:05:56 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**public** **class** ZkClientWatcher1 {

/\*\*Zookeeper地址\*/

**public** **static** **final** String ***CONNECT\_ADDR*** = "192.168.118.6:2181,192.168.118.9:2181,192.168.118.10:2181";

// 2.配置Session超时连接时间

**public** **static** **int** *SESSION\_TIMEOUT* = 2000;// ms

**public** **static** **void** main(String[] args) **throws** Exception {

ZkClient zkc =**new** ZkClient( **new** ZkConnection(***CONNECT\_ADDR***),*SESSION\_TIMEOUT*);

//对父节点添加监听 子节点的变化

zkc.subscribeChildChanges("/super", **new** IZkChildListener() {

@Override

**public** **void** handleChildChange(String parentPath, List<String> currentChilds) **throws** Exception {

System.***out***.println("parentPath:"+parentPath);

System.***out***.println("currentChild:"+currentChilds);

}

});

zkc.createPersistent("/super");

Thread.*sleep*(1000);

zkc.createPersistent("/super"+"/"+"guigu","abcguigu");

Thread.*sleep*(1000);

zkc.createPersistent("/super"+"/"+"hzguigu","abchzgguigu");

Thread.*sleep*(1000);

zkc.deleteRecursive("/super");

}

}

#### 数据监听

**package** com.guigu.zookeeper.zkclient.watcher;

**import** java.util.List;

**import** org.I0Itec.zkclient.IZkChildListener;

**import** org.I0Itec.zkclient.IZkConnection;

**import** org.I0Itec.zkclient.IZkDataListener;

**import** org.I0Itec.zkclient.ZkClient;

**import** org.I0Itec.zkclient.ZkConnection;

/\*\*

\* <p>project\_name:ZookeeperStudy</p>

\* <p>package\_name:com.guigu.zookeeper.zkclient.watcher.ZkClientWatcher1</p>

\* <p>description：</p>

\* <p>@author：刘老师<p>

\* <p> date:2018年3月14日上午9:05:56 </p>

\* <p>comments： </p>

\* <p>@version jdk1.8</p>

\*

\* <p>Copyright (c) 2018, 980991634@qq.com All Rights Reserved. </p>

\*/

**public** **class** ZkClientWatcher2 {

/\*\*Zookeeper地址\*/

**public** **static** **final** String ***CONNECT\_ADDR*** = "192.168.118.6:2181,192.168.118.9:2181,192.168.118.10:2181";

// 2.配置Session超时连接时间

**public** **static** **int** *SESSION\_TIMEOUT* = 2000;// ms

**public** **static** **void** main(String[] args) **throws** Exception {

ZkClient zkc =**new** ZkClient( **new** ZkConnection(***CONNECT\_ADDR***),*SESSION\_TIMEOUT*);

zkc.createPersistent("/super","testSuperguigu");

//对父节点 添加子节点变化的监听

zkc.subscribeDataChanges("/super", **new** IZkDataListener() {

@Override

**public** **void** handleDataDeleted(String dataPath) **throws** Exception {

System.***out***.println("删除节点:"+dataPath);

}

@Override

**public** **void** handleDataChange(String dataPath, Object data) **throws** Exception {

System.***out***.println("变更节点为:"+dataPath+",变更的内容是:"+data);

}

});

Thread.*sleep*(1000);

zkc.writeData("/super", "helloguigu");

Thread.*sleep*(1000);

zkc.delete("/super");

Thread.*sleep*(1000);

}

}