**实验题目：**

**实验 4 ZooKeeper 搭建与操作**

**实验环境：**

本次环境是： Centos7.6 + jdk1.

IP 地址是：47.113.102.106

工具在/soft 目录下

**实验技术原理：**

**【4-1】 ZK 安装与操作**

[1] Linux 平台下安装 zookeeper

通常服务类程序我们都将它安装在 linux 平台上，zookeeper 在 linux 平台上也能发挥最佳性能。从官网下载 zookeeper，利用 ssh 工具或其他方式发送至 linux 服务器上，解压 zookeeper 安装包，进入 conf 目录下创建 zoo.cfg 配置文件并修改。进入 bin 目录下利用 zookeeper 自带工具 zkServer.sh 和 zkCli.sh 进行启动和连接 zookeeper 服务。

[2] Zookeeper 伪集群模式配置

为完全发挥并提高 zookeeper 的性能，在安装部署过程中需要采用伪集群或集群部署。实验中主要以 linux 平台为主介绍 zookeeper 的集群部署。

[3] zookeeper 服务端和客户端常用命令

Zookeeper 上面有个类似文件系统的目录结构，可以在上面添加各种目录，但是这种目录在 zookeeper 中叫做“节点”，同样这些节点都可以进行增删改操作；

（1）进入 zookeeper 的命令操作模式：

首先进入到 bin 路径下面：cd /usr/local/zk/bin，执行命令：zkCil.sh 进入 zookeeper 命令操作模式：进入之后可以查看到 zk 的相关命令，这些命令包括创建、删除、查看等命令，类似于 hdfs 系统中的命令已经 Linux 的 shell 命令，但是定义有区别，截图看看 zk 的命令：

（2）用 shell 简单操作：查看节点下面的节点：ls /，例如：ls /zookeeper，就能查看 zookeeper 节点下面的内容：创建节点，设置内容：create /chaorenhadoop，就能在创建的文件中设置上“hadoop”的文本内容：查看文件内容：get /chaoren，文本内容中会输出一大串其他像版本、日期等信息

**【4-2】ZK 的 JAVA 开发**

通过 zookeeper JavaAPI 创建会话连接，并利用 zookeeper 对象创建节点。判断当前连接是否连接并且事件类型是否等于 None 后通过 zookeeper 对象调用判断节点是否存在的方法，传入满足条件的参数后并返回节点状态信息。利用 zookeeper 对象调用异步判断节点是否存在方法，根据参数类型传入参数返回为空，其中需要传入实现了 StatCallBack 接口的类。在实现接口的类中实现必须的方法，在方法体中收集异步判断节点返回的节点状态信息

实验过程

**【4-1】 ZK 安装与操作**

**[1] Linux 平台下安装 zookeeper**

1.1 zookeeper 软件存放在 soft 文件下,在 soft 文件下查看。

```
(base) [root@ferry ~]# cd ~/simple/soft
(base) [root@ferry soft]# ls
eclipse-java-luna-SR2-linux-gtk-x86_64.tar.gz  hadoop-2.4.1.tar.gz  zookeeper-3.4.14.tar.gz
(base) [root@ferry soft]#
```

1.2 利用 tar 解压 zookeeper 到 simple 目录下。

```
(base) [root@ferry soft]# tar -zxvf ~/simple/soft/zookeeper-3.4.14.tar.gz -C ~/simple
```

1.3 在 simple 目录下进行查看解压的文件，

```
(base) [root@ferry soft]# cd ~/simple
(base) [root@ferry simple]# ls
core-site.xml  Hadoop-2.4.1      HelloWorld.txt  Hw    jdk       out         soft        test.txt  zookeeper-3.4.14  数据.txt
hadoop-2.4.1   HelloWorld.class  hw              Hw.c  mapper.sh reducer.sh  source.txt  word.txt  数据 2.txt
(base) [root@ferry simple]#
```

1.4 进入到 zookeeper-3.4.5/conf 目录下，复制 zoo_sample.cfg 文件并改名 zoo.cfg。

```
(base) [root@ferry simple]# cd zookeeper-3.4.14/conf/
(base) [root@ferry conf]# ls
configuration.xsl  log4j.properties  zoo_sample.cfg
(base) [root@ferry conf]# cp zoo_sample.cfg zoo.cfg
(base) [root@ferry conf]# ls
configuration.xsl  log4j.properties  zoo.cfg  zoo_sample.cfg
(base) [root@ferry conf]#
```

1.5 vi zoo.cfg 进行 zoo.cfg 配置文件修改，

```
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/root/simple/zookeeper-3.4.14/data_single
dataLog=/root/simple/zookeeper-3.4.14/data_single_log
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
```

1.6 在 simple 目录下通过新建 data_single 和 data_single_log 文件。

```
(base) [root@ferry conf]# cd ~/simple
(base) [root@ferry simple]# mkdir data_single
(base) [root@ferry simple]# mkdir data_single_log
(base) [root@ferry simple]# ls
core-site.xml     hadoop-2.4.1      HelloWorld.txt  Hw.c       out          source.txt  zookeeper-3.4.14
data_single       Hadoop-2.4.1      hw              jdk        reducer.sh   test.txt    数据 2.txt
data_single_log   HelloWorld.class  Hw              mapper.sh  soft         word.txt    数据.txt
(base) [root@ferry simple]#
```

1.7 通过 cd zookeeper-3.4.5/bin 进入到 zookeeper bin 文件下，通过密令./zkServer.sh start 来启动服务

```
(base) [root@ferry simple]# cd zookeeper-3.4.14/bin
(base) [root@ferry bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
(base) [root@ferry bin]#
```

1.8 在 zookeeper bin 目录下，打开客户端连接 zookeeper 服务，

```
(base) [root@ferry bin]# ./zkCli.sh -timeout 5000 -server 47.113.102.106
Connecting to 47.113.102.106
2020-06-06 12:42:05,447 [myid:] - INFO  [main:Environment@100] - Client environment:zookeeper.version=3.
e8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
2020-06-06 12:42:05,450 [myid:] - INFO  [main:Environment@100] - Client environment:host.name=ferry.szu
2020-06-06 12:42:05,451 [myid:] - INFO  [main:Environment@100] - Client environment:java.version=1.8.0_2
2020-06-06 12:42:05,452 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=Oracle C
2020-06-06 12:42:05,452 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/root/simp
2020-06-06 12:42:05,453 [myid:] - INFO  [main:Environment@100] - Client environment:java.class.path=/roo
4/bin/../zookeeper-server/target/classes:/root/simple/zookeeper-3.4.14/bin/../build/classes:/root/simple
/zookeeper-server/target/lib/*.jar:/root/simple/zookeeper-3.4.14/bin/../build/lib/*.jar:/root/simple/zoo
```

1.9 查看根节点下有哪些节点，/zookeeper 节点是 zookeeper 自带节点，

```
JLine support is enabled
2020-06-06 12:42:05,583 [myid:] - INFO  [main-SendThread(47.113.102.1
ction to server 47.113.102.106/47.113.102.106:2181. Will not attempt
2020-06-06 12:42:05,608 [myid:] - INFO  [main-SendThread(47.113.102.1
ablished to 47.113.102.106/47.113.102.106:2181, initiating session
2020-06-06 12:42:05,717 [myid:] - INFO  [main-SendThread(47.113.102.1
t complete on server 47.113.102.106/47.113.102.106:2181, sessionid =

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: 47.113.102.106(CONNECTED) 0] ls /
[zookeeper]
[zk: 47.113.102.106(CONNECTED) 1]
```

## [2] Zookeeper 伪集群模式配置

一、伪集群模式

1.1 zookeeper 软件存放在 soft 文件下,在 soft 文件下查看。

1.2 利用 tar 解压 zookeeper 到 simple 目录下。

1.3 在 simple 下进行查看解压的文件，

此步骤已于前面完成解压。

1.4 通过命令 cd zookeeper-3.4.5 进入到 zookeeper 目录下，命令 mkdir data_single 新建 data_single 文件夹， 在文件夹下 touch myid 在新建 myid 文件,对文件进行编译 echo 1 >> myid。

```
(base) [root@ferry data_single]# cd ~/simple/zookeeper-3.4.14
(base) [root@ferry zookeeper-3.4.14]# ls
bin          ivysettings.xml  pom.xml              zookeeper-3.4.14.jar.asc   zookeeper-docs
build.xml    ivy.xml          README.md            zookeeper-3.4.14.jar.md5   zookeeper-it
conf         lib              README_packaging.txt zookeeper-3.4.14.jar.sha1  zookeeper-jute
data_single  LICENSE.txt      src                  zookeeper-client           zookeeper-recipes
dist-maven   NOTICE.txt       zookeeper-3.4.14.jar zookeeper-contrib          zookeeper-server
(base) [root@ferry zookeeper-3.4.14]# cd data_single
(base) [root@ferry data_single]# touch myid
(base) [root@ferry data_single]# ls
myid  version-2  zookeeper_server.pid
(base) [root@ferry data_single]# echo 1 >> myid
(base) [root@ferry data_single]# ls
myid  version-2  zookeeper_server.pid
(base) [root@ferry data_single]# cat myid
1
(base) [root@ferry data_single]#
```

1.5 相同的步骤在 zookeeper 目录下新建 data_single_2 文件夹，并在文件夹下新建 myid 文件，为 myid 文件赋值 2。

```
(base) [root@ferry zookeeper-3.4.14]# cd ~/simple/zookeeper-3.4.14
(base) [root@ferry zookeeper-3.4.14]# mkdir data_single_2
mkdir: 无法创建目录"data_single_2": 文件已存在
(base) [root@ferry zookeeper-3.4.14]# ls
2            data_single    ivy.xml      pom.xml              zookeeper-3.4.14.jar
bin          data_single_2  lib          README.md            zookeeper-3.4.14.jar.asc
build.xml    dist-maven     LICENSE.txt  README_packaging.txt zookeeper-3.4.14.jar.md5
conf         ivysettings.xml NOTICE.txt  src                  zookeeper-3.4.14.jar.sha1
(base) [root@ferry zookeeper-3.4.14]# cd data_single_2
(base) [root@ferry data_single_2]# touch myid
(base) [root@ferry data_single_2]# ls
myid
(base) [root@ferry data_single_2]# echo 2 >> myid
(base) [root@ferry data_single_2]# cat myid
2
(base) [root@ferry data_single_2]#
```

1.6 相同的步骤在 zookeeper 目录下新建 data_single_3 文件夹，并在文件夹下新建 myid 文件，为 myid 文件赋值 3。

```
(base) [root@ferry data_single_2]# cd ~/simple/zookeeper-3.4.14
(base) [root@ferry zookeeper-3.4.14]# mkdir data_single_3
(base) [root@ferry zookeeper-3.4.14]# ls
2            data_single_2    lib          README_packaging.txt      zookeepe
bin          data_single_3    LICENSE.txt  src                       zookeepe
build.xml    dist-maven       NOTICE.txt   zookeeper-3.4.14.jar      zookeepe
conf         ivysettings.xml  pom.xml      zookeeper-3.4.14.jar.asc  zookeepe
data_single  ivy.xml          README.md    zookeeper-3.4.14.jar.md5  zookeepe
(base) [root@ferry zookeeper-3.4.14]# cd data_single_3
(base) [root@ferry data_single_3]# touch myid
(base) [root@ferry data_single_3]# ls
myid
(base) [root@ferry data_single_3]# echo 3 >> myid
(base) [root@ferry data_single_3]# cat myid
3
(base) [root@ferry data_single_3]#
```

1.7 进入到 zookeeper-3.4.5/conf 目录下，复制 zoo_sample.cfg 文件并改名 zoo.cfg。

```
(base) [root@ferry data_single_3]# cd ~/simple
(base) [root@ferry simple]# cd zookeeper-3.4.14/conf/
(base) [root@ferry conf]# ls
configuration.xsl  log4j.properties  zoo.cfg  zoo_sample.cfg
(base) [root@ferry conf]#
```

1.8 vi　zoo.cfg 进行 zoo.cfg 配置文件修改，设置伪集群配置参数，添加伪集群配置，

```
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1

server.1=127.0.0.1:2222:2223
server.2=127.0.0.1:3333:3334
server.3=127.0.0.1:4444:4445
-- INSERT --
```

1.9 命令 cp　zoo.cfg zk2.cfg 复制配置文件 zoo.cfg 为 zk2.cfg 并修改端口为 2182，

```
(base) [root@ferry data_single_3]# cd ~/simple
(base) [root@ferry simple]# cd zookeeper-3.4.14/conf/
(base) [root@ferry conf]# ls
configuration.xsl  log4j.properties  zoo.cfg  zoo_sample.cfg
(base) [root@ferry conf]# vi zoo.cfg
(base) [root@ferry conf]# cp zoo.cfg zk2.cfg
(base) [root@ferry conf]# vi zk2.cfg
```

```
# example sakes.
dataDir=/root/simple/zookeeper-3.4.14/data_single_2
# the port at which the clients will connect
clientPort=2182
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
```

1.10 命令 cp　zoo.cfg zk3.cfg 复制配置文件 zoo.cfg 为 zk3.cfg 并修改端口为 2183，

```
(base) [root@ferry data_single_3]# cd ~/simple
(base) [root@ferry simple]# cd zookeeper-3.4.14/conf/
(base) [root@ferry conf]# ls
configuration.xsl  log4j.properties  zoo.cfg  zoo_sample.cfg
(base) [root@ferry conf]# vi zoo.cfg
(base) [root@ferry conf]# cp zoo.cfg zk2.cfg
(base) [root@ferry conf]# vi zk2.cfg
(base) [root@ferry conf]# cp zoo.cfg zk3.cfg
(base) [root@ferry conf]# vi zk3.cfg
```

```
# example sakes.
dataDir=/root/simple/zookeeper-3.4.14/data_single_3
# the port at which the clients will connect
clientPort=2183
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintena
#
```

1.11 启动集群:通过 cd zookeeper-3.4.5/bin 进入到 zookeeper bin 文件下，通过命

令./zkServer.sh start 来启动服务，

```
(base) [root@ferry ~]# cd ~/simple/zookeeper-3.4.14/bin
(base) [root@ferry bin]# ls
README.txt    zkCli.cmd  zkEnv.cmd  zkServer.cmd  zkTxnLogToolkit.cmd  zookeeper.out
zkCleanup.sh  zkCli.sh   zkEnv.sh   zkServer.sh   zkTxnLogToolkit.sh
(base) [root@ferry bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... already running as process 14737.
(base) [root@ferry bin]#
```

1.12 同理启动 zk2.cfg zk3.cfg 配置的服务器,利用 jps 查看。

```
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... already running as process 14737.
(base) [root@ferry bin]# ./zkServer.sh start zk2.cfg
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zk2.cfg
Starting zookeeper ... STARTED
(base) [root@ferry bin]# jps
14737 QuorumPeerMain
32694 QuorumPeerMain
470 Jps
(base) [root@ferry bin]# ./zkServer.sh start zk3.cfg
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zk3.cfg
Starting zookeeper ... STARTED
(base) [root@ferry bin]# jps
14737 QuorumPeerMain
32694 QuorumPeerMain
873 QuorumPeerMain
1018 Jps
(base) [root@ferry bin]#
```

**[3] zookeeper 服务端和客户端常用命令**

一、服务端命令

1.1 启动 zookeeper,在 simple 下的 zookeeper-3.4.5/bin 下执行命令./zkServer.sh star

```
(base) [root@ferry ~]# cd /simple/zookeeper-3.4.5/bin/
-bash: cd: /simple/zookeeper-3.4.5/bin/: 没有那个文件或目录
(base) [root@ferry ~]# cd ~/simple/zookeeper-3.4.14/bin/
(base) [root@ferry bin]# pwd
/root/simple/zookeeper-3.4.14/bin
(base) [root@ferry bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... already running as process 14737.
(base) [root@ferry bin]#
```

1.2 查看服务状态及集群,重启 zookeeper,停止 zookeeper,

```
(base) [root@ferry bin]# ./zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Mode: standalone
(base) [root@ferry bin]# ./zkServer.sh restart
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... ^[[ASTARTED
(base) [root@ferry bin]# ./zkServer.sh stop
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
(base) [root@ferry bin]#
```

二、客户端命令

2.1 在 zookeeper 启动状态下

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

```
(base) [root@ferry bin]# cd ..
(base) [root@ferry zookeeper-3.4.14]# bin/zkCli.sh -timeout 5000 -r -server 47.113.102.106
Connecting to 47.113.102.106
2020-06-06 14:15:27,416 [myid:] - INFO  [main:Environment@100] - Client environment:zookeeper.version=3.4.14-4c
e8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
2020-06-06 14:15:27,419 [myid:] - INFO  [main:Environment@100] - Client environment:host.name=ferry.szu
2020-06-06 14:15:27,419 [myid:] - INFO  [main:Environment@100] - Client environment:java.version=1.8.0_241
2020-06-06 14:15:27,422 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=Oracle Corporat
2020-06-06 14:15:27,422 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/root/simple/jdk/
2020-06-06 14:15:27,422 [myid:] - INFO  [main:Environment@100] - Client environment:java.class.path=/root/simpl
4/bin/../zookeeper-server/target/classes:/root/simple/zookeeper-3.4.14/bin/../build/classes:/root/simple/zookee
/zookeeper-server/target/lib/*.jar:/root/simple/zookeeper-3.4.14/bin/../build/lib/*.jar:/root/simple/zookeeper-
```

2.2 创建节点,

格式：create [-s] [-e] path data acl

```
[zk: 47.113.102.106(CONNECTED) 1] create /nodel test
Created /nodel
[zk: 47.113.102.106(CONNECTED) 2] ls /
[nodel, zookeeper]
[zk: 47.113.102.106(CONNECTED) 3]
```

2.3 修改节点数据,

格式：set path data [version]

```
[zk: 47.113.102.106(CONNECTED) 3] get /nodel
test
cZxid = 0x4
ctime = Sat Jun 06 14:16:17 CST 2020
mZxid = 0x4
mtime = Sat Jun 06 14:16:17 CST 2020
pZxid = 0x4
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 4
numChildren = 0
[zk: 47.113.102.106(CONNECTED) 4] set /nodel tt
cZxid = 0x4
ctime = Sat Jun 06 14:16:17 CST 2020
mZxid = 0x5
mtime = Sat Jun 06 14:16:59 CST 2020
pZxid = 0x4
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 2
numChildren = 0
[zk: 47.113.102.106(CONNECTED) 5]
```

2.4 获取节点数据以及节点其他信息

格式：get path [watch]

```
[zk: 47.113.102.106(CONNECTED) 5] get /nodel
tt
cZxid = 0x4
ctime = Sat Jun 06 14:16:17 CST 2020
mZxid = 0x5
mtime = Sat Jun 06 14:16:59 CST 2020
pZxid = 0x4
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 2
numChildren = 0
```

2.5 查询子节点个数,

格式：ls path [watch]

```
[zk: 47.113.102.106(CONNECTED) 6] ls /
[nodel, zookeeper]
```

2.6 查看当前节点概况（不显示子节点）

格式：stat path [version]

```
[zk: 47.113.102.106(CONNECTED) 7] stat /
cZxid = 0x0
ctime = Thu Jan 01 08:00:00 CST 1970
mZxid = 0x0
mtime = Thu Jan 01 08:00:00 CST 1970
pZxid = 0x4
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 0
numChildren = 2
[zk: 47.113.102.106(CONNECTED) 8]
```

2.7 查看当前节点概况（包括子节点列表）

格式：ls2 path [watch]

```
[zk: 47.113.102.106(CONNECTED) 8] ls2 /
[nodel, zookeeper]
cZxid = 0x0
ctime = Thu Jan 01 08:00:00 CST 1970
mZxid = 0x0
mtime = Thu Jan 01 08:00:00 CST 1970
pZxid = 0x4
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 0
numChildren = 2
[zk: 47.113.102.106(CONNECTED) 9]
```

2.8 设置配额,

格式：setquota -n|-b val path

```
[zk: 47.113.102.106(CONNECTED) 9] setquota -n 2 /nodel
Comment: the parts are option -n val 2 path /nodel
[zk: 47.113.102.106(CONNECTED) 10]
```

2.9 查看配额

格式：listquota /path

```
[zk: 47.113.102.106(CONNECTED) 10] listquota /nodel
absolute path is /zookeeper/quota/nodel/zookeeper_limits
Output quota for /nodel count=2,bytes=-1
Output stat for /nodel count=1,bytes=2
[zk: 47.113.102.106(CONNECTED) 11]
```

**【4-2】ZK 的 JAVA 开发**

**1. 实验 4-2 zookeeper API 操作**

**3-1 Zookeeper Java API-判断节点是否存在（同步和异步）**

1.切换到 simple/zookeeper-3.4.5 目录下。

```
(base) [root@ferry ~]# cd ~/simple/zookeeper-3.4.14
(base) [root@ferry zookeeper-3.4.14]# ls
2                lib                          zookeeper-3.4.14.jar.sha
bin              LICENSE.txt                  zookeeper-client
build.xml        NOTICE.txt                   zookeeper-contrib
conf             pom.xml                      zookeeper-docs
data_single      README.md                    zookeeper-it
data_single_2    README_packaging.txt         zookeeper-jute
data_single_3    src                          zookeeper.out
dist-maven       zookeeper-3.4.14.jar         zookeeper-recipes
ivysettings.xml  zookeeper-3.4.14.jar.asc     zookeeper-server
ivy.xml          zookeeper-3.4.14.jar.md5
```

2.利用 bin/zkServer.sh start 启动 zookeeper 服务。

```
(base) [root@ferry zookeeper-3.4.14]# bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

3.使用 bin/zkServer.sh status 查看服务是否启动。此处使用单机模式，需要把之前实验 1-2 中 zoo.cfg 文件中的伪集群配置参数去掉，否则查看服务启动状态时会出现问题。

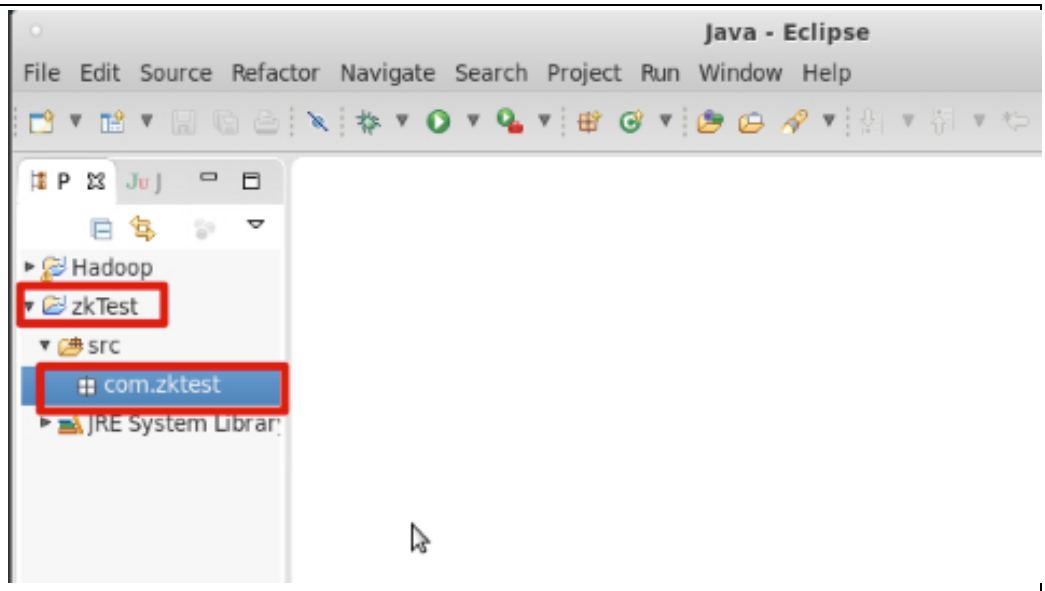
```
(base) [root@ferry zookeeper-3.4.14]# bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Mode: standalone
(base) [root@ferry zookeeper-3.4.14]#
```

4.利用./bin/zkCli.sh -timeout 5000 -server 192.168.0.202:2181 连接测试 zookeeper 服务。此处的 ip 地址要使用自己虚拟机的 ip 地址。

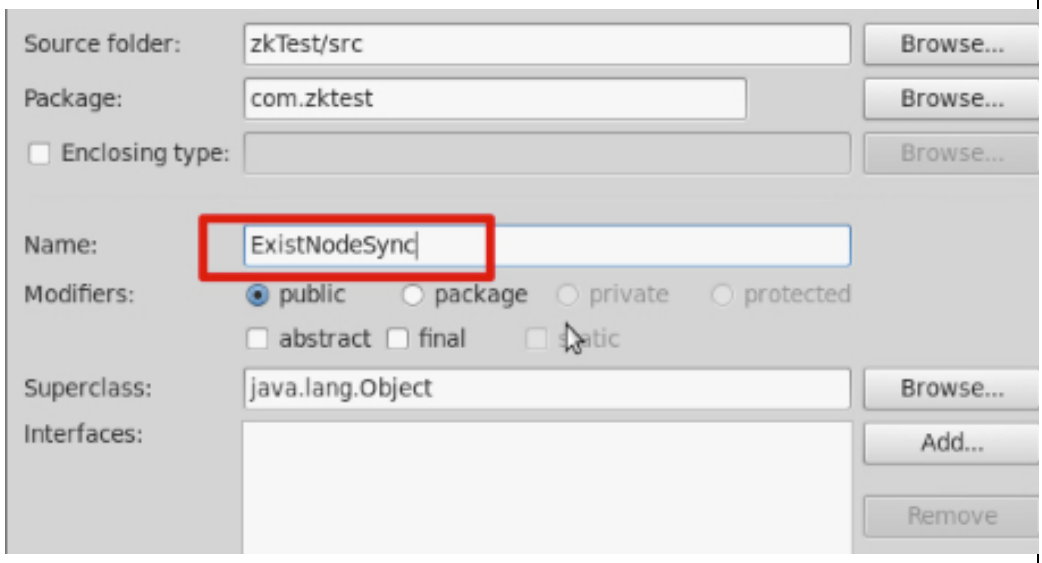
```
(base) [root@ferry zookeeper-3.4.14]# ./bin/zkCli.sh -timeout 5000 -server 47.113.102.106:2181
Connecting to 47.113.102.106:2181
2020-07-10 13:16:05,550 [myid:] - INFO  [main:Environment@100] - Client environment:zookeeper.version=3.4.14-4c25d480e66a
1de8bd2fd8da255ac140bcf, built on 03/06/2019 16:18 GMT
2020-07-10 13:16:05,553 [myid:] - INFO  [main:Environment@100] - Client environment:host.name=ferry.szu
2020-07-10 13:16:05,553 [myid:] - INFO  [main:Environment@100] - Client environment:java.version=1.8.0_241
2020-07-10 13:16:05,555 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=Oracle Corporation
2020-07-10 13:16:05,555 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/root/simple/jdk/jdk1.8.0_24
```

5.在 eclipse 的项目列表中右键点击，新建一个项目并命名为 zkTest。

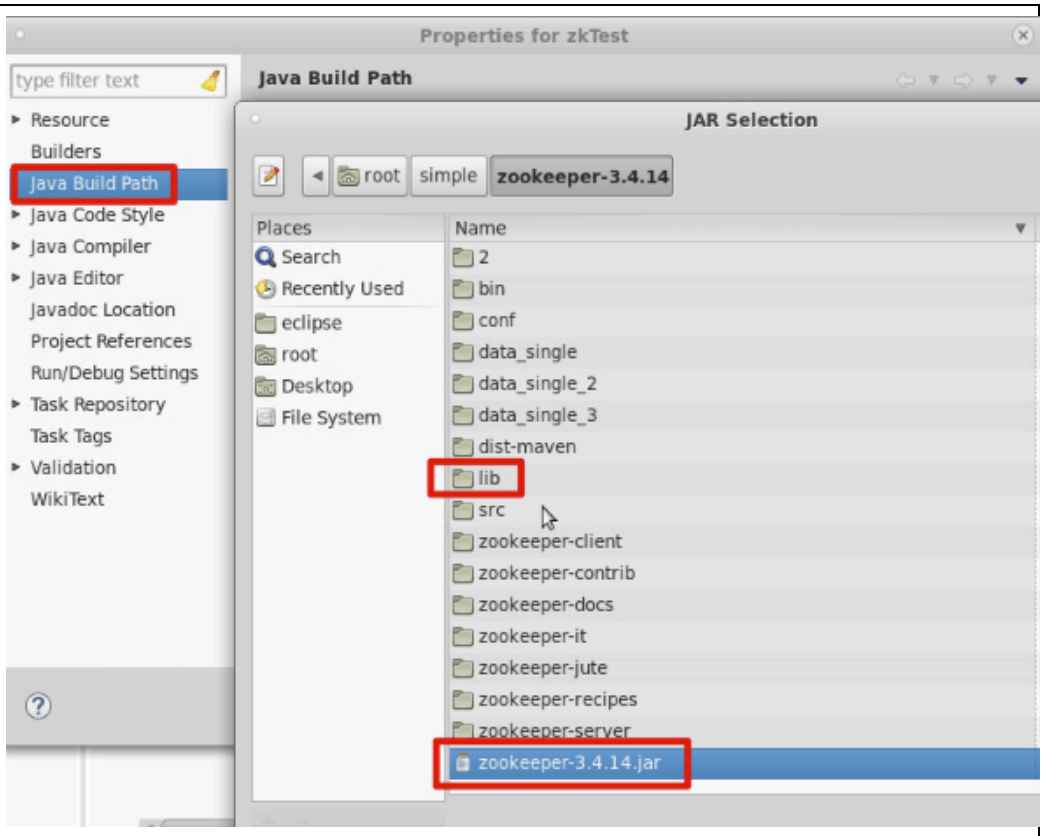
6.在项目的 src 目录下新建一个名为 com.zktest 的包。

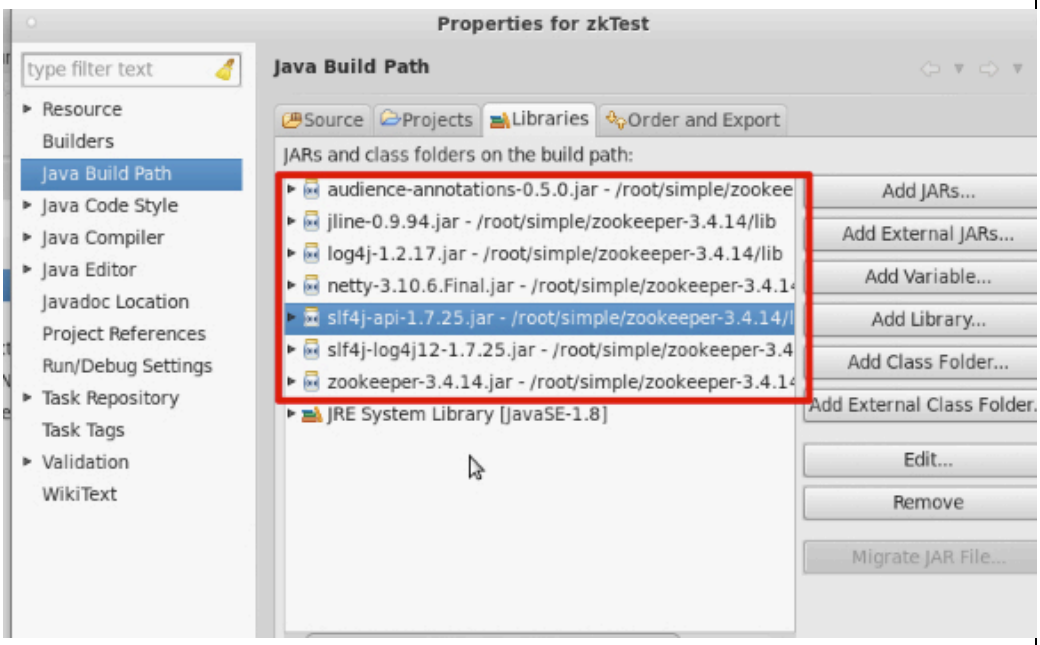



7.右键点击 com.zktest，新建一个名为 ExistNodeSync 的类。

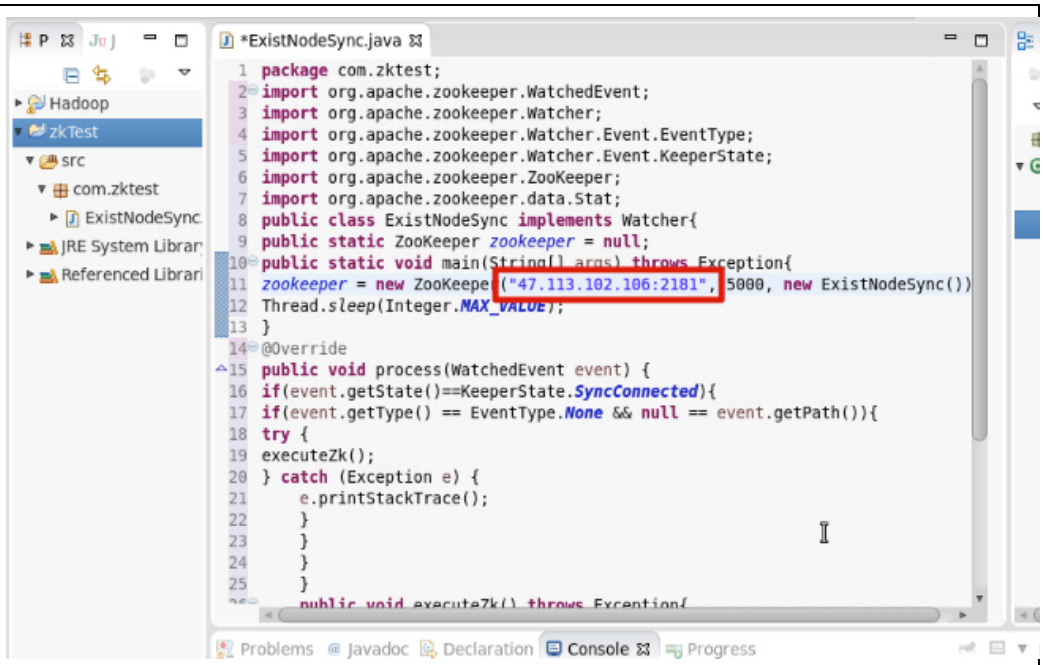



8.在项目上右键点击，通过 build path->configure build path->libraries->add external jars->文件系统->simple->zookeeper-3.4.5，添加 lib 文件夹下的 jar 包以及 zookeeper-3.4.5.jar，共 6 个 jar 包。




9.利用 zookeeper 对象创建会话，创建对象时的 ip 地址需要设置成自己虚拟机的 ip 地址，之后设置连接 zookeeper 后休眠；实现类的 watcher 接口并实现接口的方法 process(WatchedEvent event)，在 process(WatchedEvent event)加入判断当前状态及事件类型。




```
package com.zktest;
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;
import org.apache.zookeeper.data.Stat;
public class ExistNodeSync implements Watcher{
public static ZooKeeper zookeeper = null;
public static void main(String[] args) throws Exception{
zookeeper = new ZooKeeper("47.113.102.106:2181", 5000, new ExistNodeSync())
Thread.sleep(Integer.MAX_VALUE);
}
@Override
public void process(WatchedEvent event) {
if(event.getState()==KeeperState.SyncConnected){
if(event.getType() == EventType.None && null == event.getPath()){
try {
executeZk();
} catch (Exception e) {
    e.printStackTrace();
    }
    }
    }
    }
    public void executeZk() throws Exception{
```

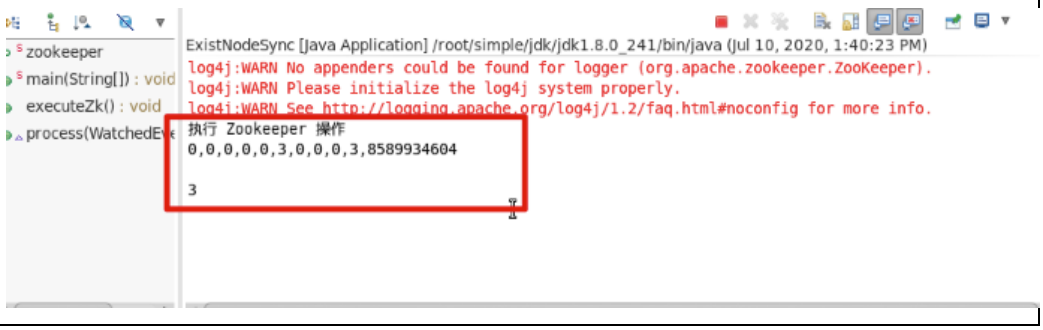
10. 创建 public void executeZk()方法，并在其中利用 zookeeper 对象判断节点是否存在。

```
   public void executeZk()throws Exception{
      System.out.println("执行 Zookeeper 操作");
      Stat stat = zookeeper.exists("/", false);
      if(stat == null){
      System.out.println("无此节点:/node6");
      }else{
      System.out.println(stat);
      System.out.println(stat.getNumChildren());
      }
      }
}
```

11.在 process(WatchedEvent event)方法中，判断如果服务连接成功并且事件类型返回 none，则调用 executeZk()方法，若发生异常则在命令行打印程序中出错的位置及原因。

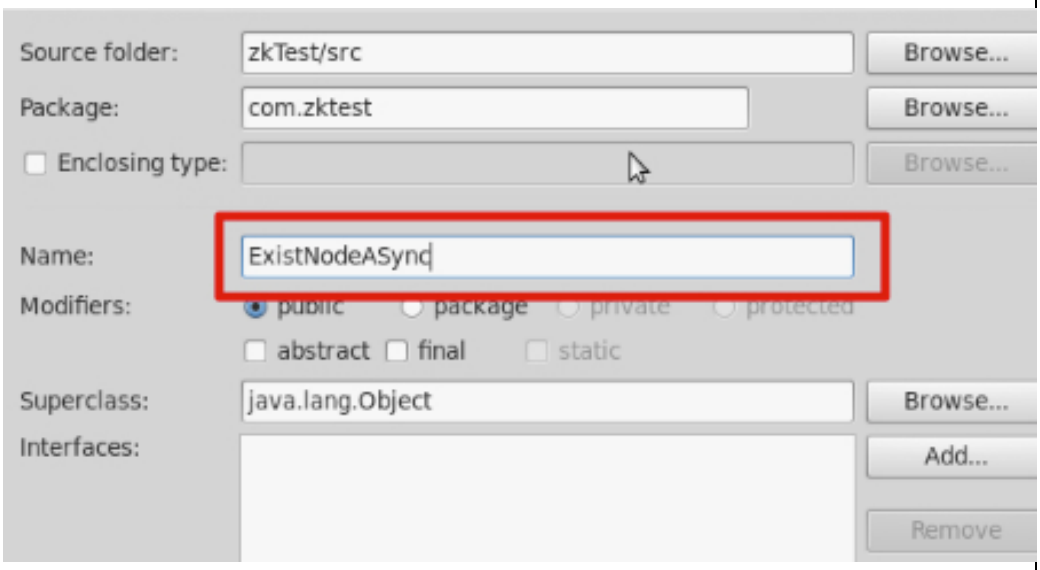
```
      if(event.getState() == KeeperState.SyncConnected){
      if(event.getType() == EventType.None && null ==
      event.getPath()){
         try{
      executeZk();
         }catch(Exception e){
            e.printStackTrace();
         }
      }
   }
}
```

12.右键，选择 run as->1 java application 运行程序，运行结果如图，可以看见显示的节点个数。

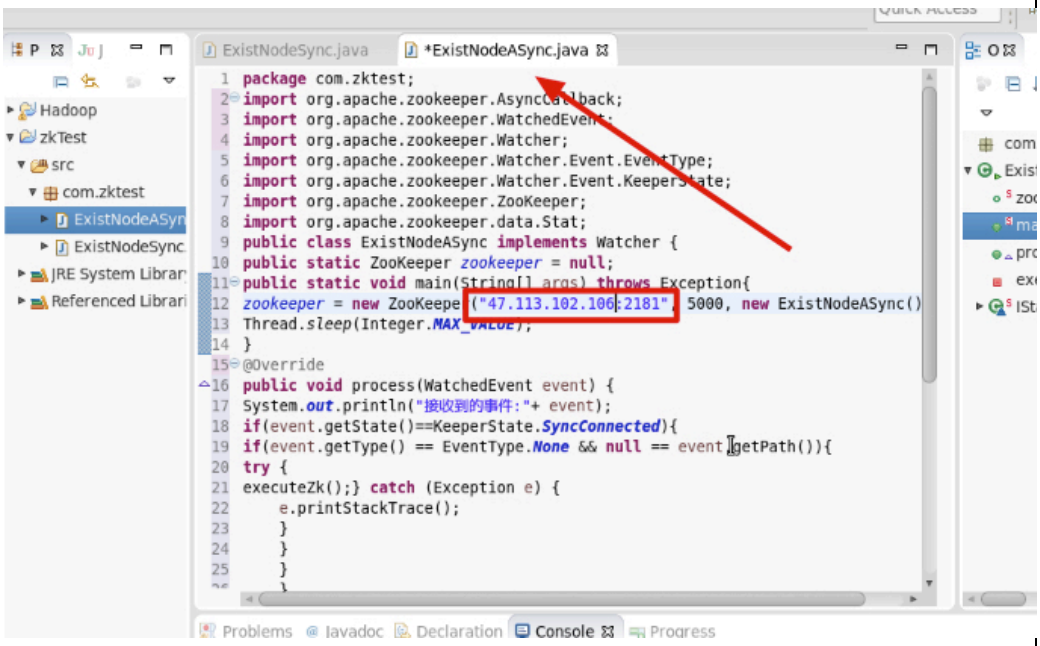


13.在终端中查看根节点下是否存在两个节点。


14.在 com.zktest 包中创建一个名为 ExistNodeASync 的类。



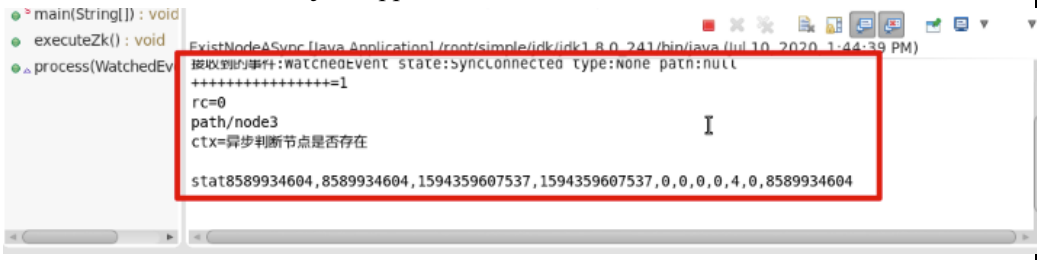

15. 利用 zookeeper 对象创建会话，创建对象时的 ip 地址需要设置成自己虚拟机的 ip 地址，之后设置连接 zookeeper 后休眠；实现类的 watcher 接口并实现接口的方法 process(WatchedEvent event)，在 process(WatchedEvent event)加入判断当前状态及事件类型。



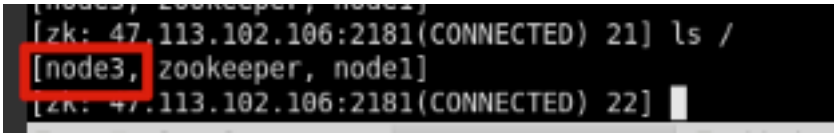

16. 创建 public void executeZk()方法，在方法中利用 zookeeper 对象判断/nodel 节点是否存在。创建一个内部类 IStatCallBack 并实现 AsyncCallback.StatCallback 接口，并且实现其 public void processResult(int rc, String path, Object ctx, Stat stat)方法，在此方法中输出异步接收判节点的状态。

```
    private void executeZk() throws Exception{
    System.out.println("+++++++++++++++=1");
    zookeeper.exists("/node3", true, new IStatCallBack(), "异步判断节点是否存
    }
    static class IStatCallBack implements AsyncCallback.StatCallback{
    @Override
    public void processResult(int rc, String path, Object ctx, Stat stat)
    StringBuilder sb = new StringBuilder();
    sb.append("rc="+rc).append("\n");
    sb.append("path"+path).append("\n");
    sb.append("ctx="+ctx).append("\n");
    System.out.println(sb.toString());
    System.out.println("stat"+stat);
    }
    }
    }
```

17. 右键，选择 run as->1 java application 运行程序，运行结果如图。

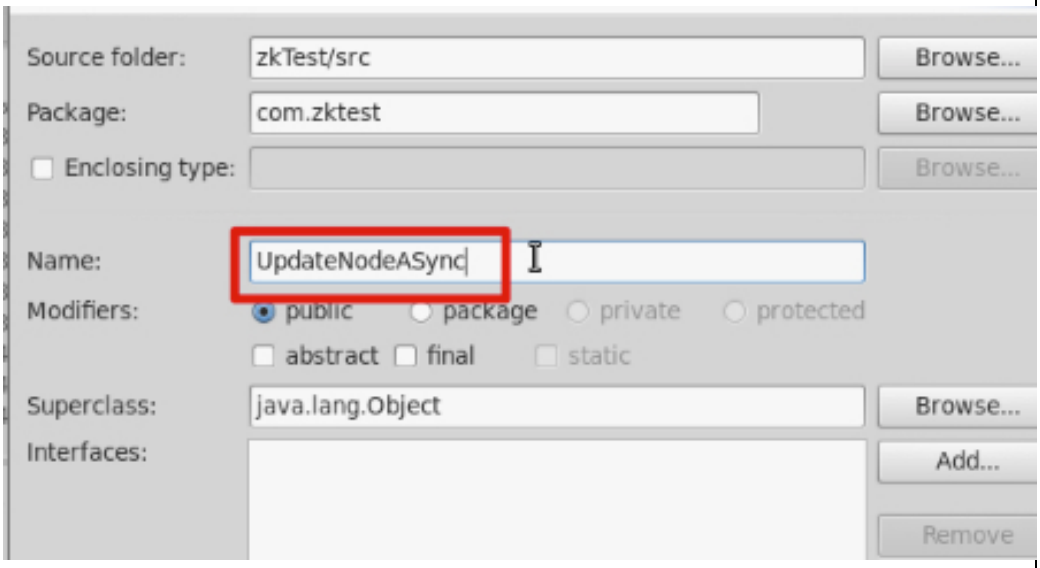



18.在终端中查看根节点下是否存在 nodel 节点。




**3-2 zookeeper javaAPI-修改节点（异步）**

1.创建新类




2.利用 zookeeper 对象创建会话，创建对象时的 ip 地址需要设置成自己虚拟机的 ip 地址，之后设置连接 zookeeper 后休眠；实现类的 watcher 接口并实现接口的方法 process(WatchedEvent event)，在 process(WatchedEvent event)加入判断当前状态及事

件类型。

UpdateNodeASync.java

```
1  package com.zktest;
2  import org.apache.zookeeper.AsyncCallback;
3  import org.apache.zookeeper.WatchedEvent;
4  import org.apache.zookeeper.Watcher;
5  import org.apache.zookeeper.Watcher.Event.EventType;
6  import org.apache.zookeeper.Watcher.Event.KeeperState;
7  import org.apache.zookeeper.ZooKeeper;
8  import org.apache.zookeeper.data.Stat;
9  public class UpdateNodeASync implements Watcher{
10 public static ZooKeeper zookeeper = null;
11 public static void main(String[] args) throws Exception {
12 zookeeper = new ZooKeeper("47.113.102.106:2181", 5000, new UpdateNodeASy
13 Thread.sleep(Integer.MAX_VALUE);
14 }
15 @Override
16 public void process(WatchedEvent event) {
17 System.out.println("接收到的事件:"+ event);
18 if(event.getState() == KeeperState.SyncConnected){
19 if(event.getType() == EventType.None && null == event.getPath()){
20 executeZk();
21 }
22 }
23 }
```

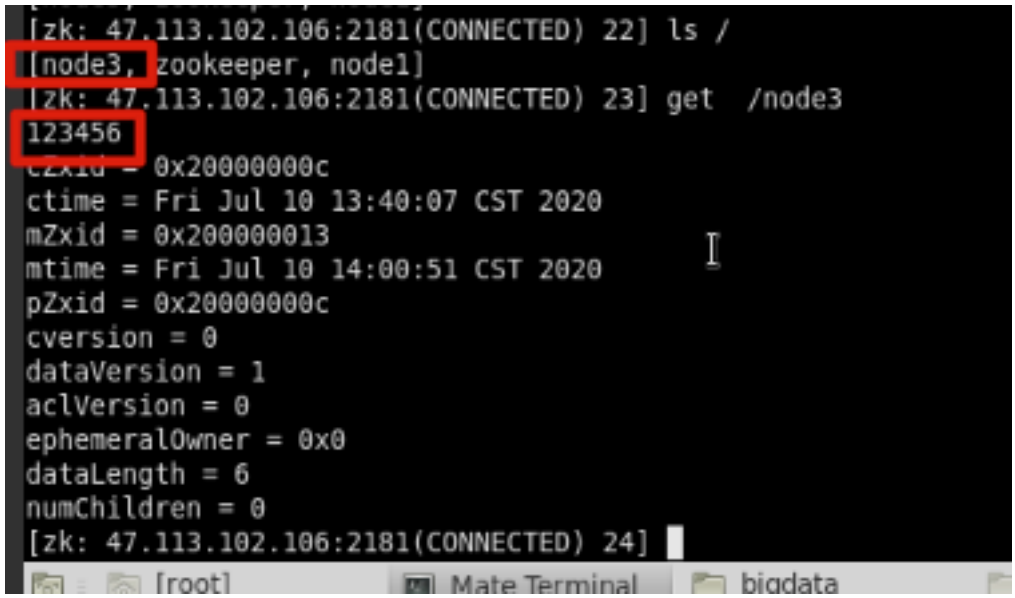
3.创建一个内部类 IStatCallBack 并实现 AsyncCallback.StatCallback 接口，并且实现其 public void processResult(int rc, String path, Object ctx, Stat stat)方法，在此方法中输出异步接收修改节点返回的值。创建 public void executeZk()方法，在方法中利用 zookeeper 对象修改/node3 节点数据为“123456”。

```
24 static class IStatCallback implements AsyncCallback.StatCallback{
25 @Override
26 public void processResult(int rc, String path, Object ctx, Stat stat) {
27 // TODO Auto-generated method stub
28 StringBuilder sb = new StringBuilder();
29 sb.append("rc="+rc).append("\n");
30 sb.append("path"+path).append("\n");
31 sb.append("ctx="+ctx).append("\n");
32 sb.append("Stat="+stat).append("\n");
33 System.out.println(sb.toString());
34 }
35 }
36 public void executeZk(){
37 zookeeper.setData("/node3", "123456".getBytes(), -1,new IStatCallback(),
38 }
39 }
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

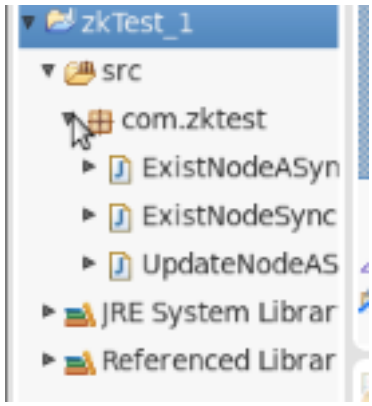
```
UpdateNodeASync (1) [Java Application] /root/simple/jdk/jdk1.8.0_241/bin/java (Jul 10, 2020, 2:00:51 PM)
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
接收到的事件:WatchedEvent state:SyncConnected type:None path:null
rc=0
path/node3
ctx=修改节点
Stat=8589934604,8589934611,1594359607537,1594360851694,1,0,0,0,6,0,8589934604
```

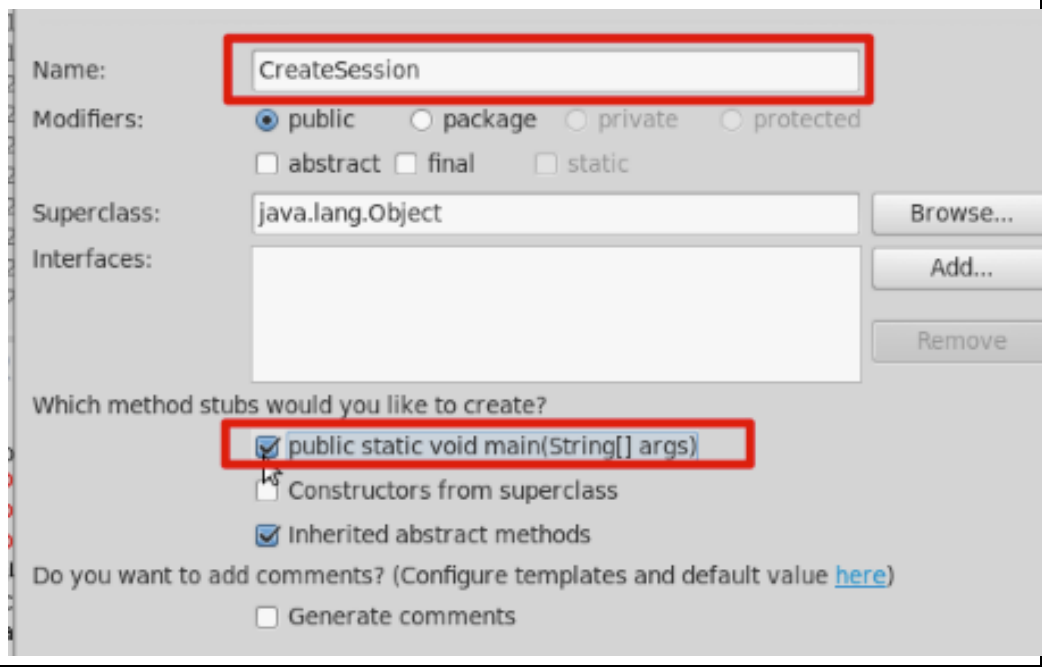
5.在终端中查看节点 nodel 的节点值是否为修改后的值。



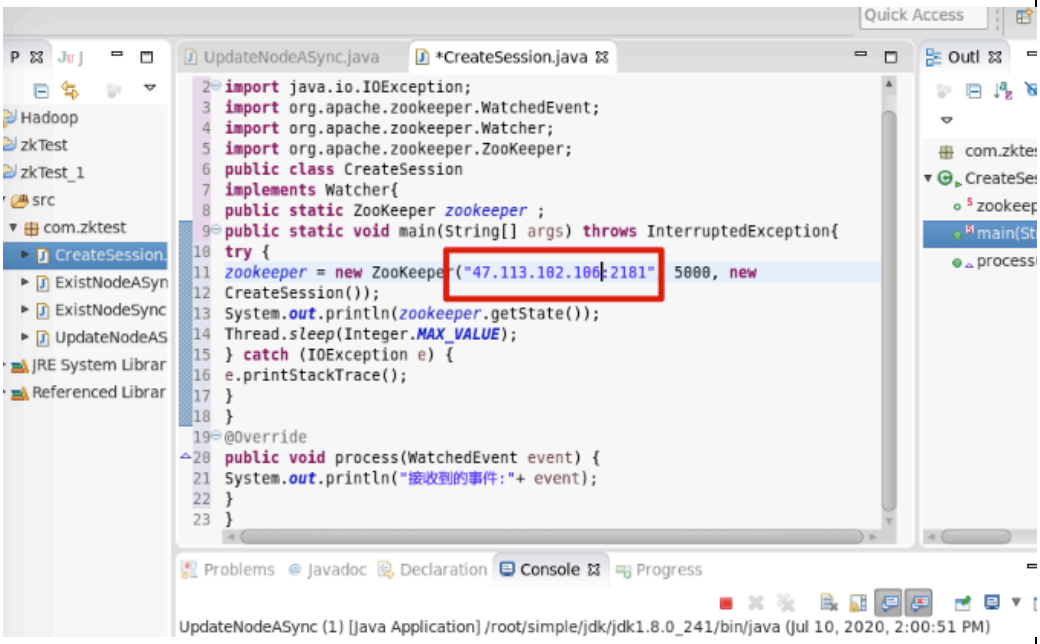

**3-3 zookeeper javaAPI-创建会话**

1.复制 zkTest 项目，并且命名为 zkTest_1，在 zkTest_1 项目 src 目录下新建一个名为 CreateSession 的类。

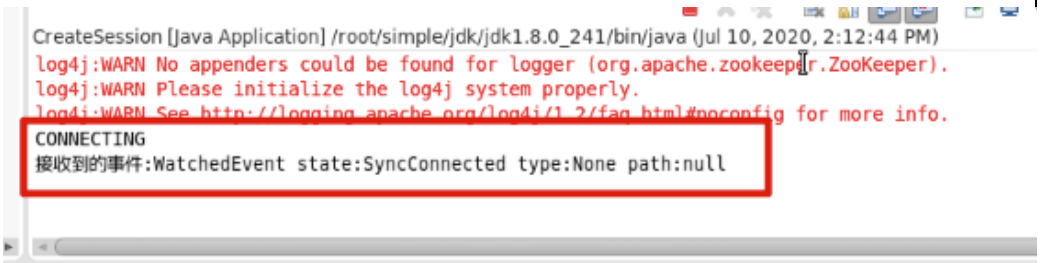


2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法中输出接收到的事件。

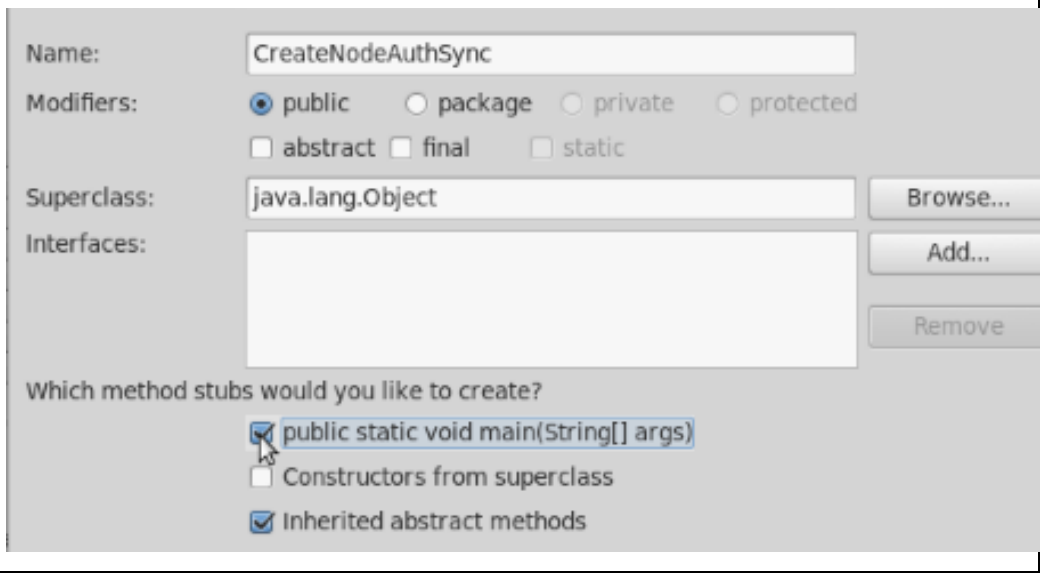



3.右键，选择 run as->1 java application 运行程序，运行结果如图。

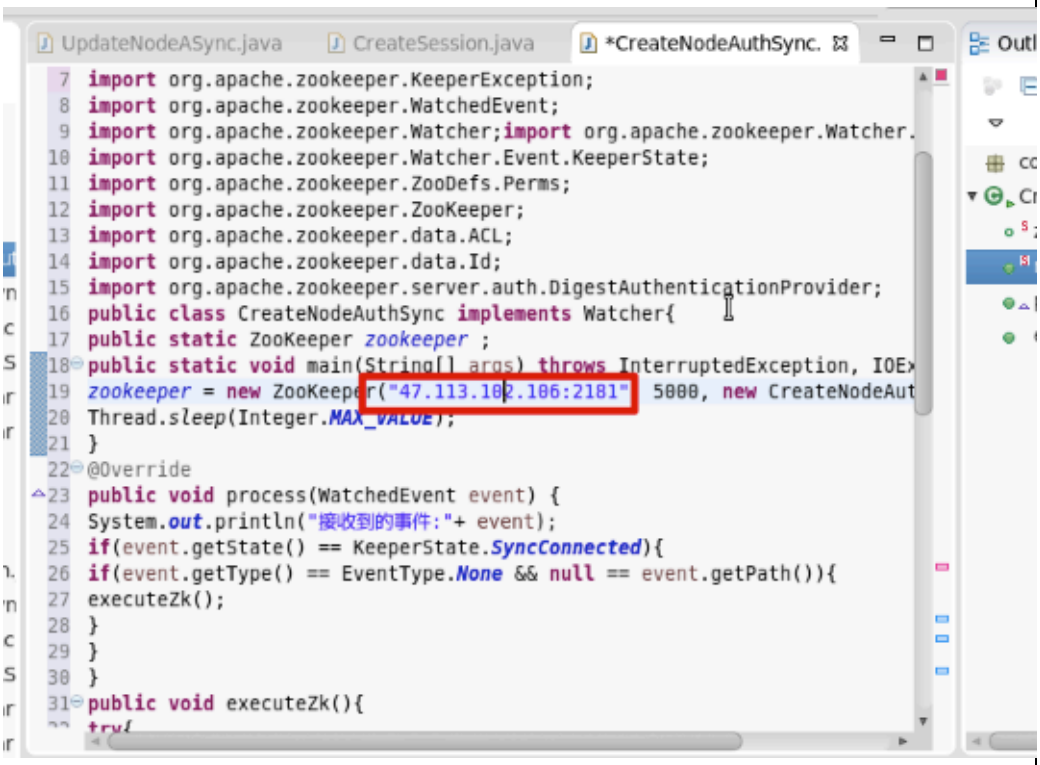



**3-4 zookeeper javaAPI-节点权限控制**

1.在 zkTest 项目 src 目录下新建一个名为 CreateNodeAuthSync 的类。



2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法中输出接收到的事件，并判断当前状态及事件类型，如果服务连接成功并且事件类型返回 None，则调用 executeZk()方法。




3.创建 public void executeZk()方法，并在其中创建节点 node07，设置此节点数据为 testdata，同时添加权限。

```
UpdateNodeASync.java   CreateSession.java   CreateNodeAuthSync.j
33 ACL aclIp = new ACL(Perms.ALL,new Id("ip","192.168.1.3"));
34 ACL
35 aclDigest
36 =
37 new
38 ACL(Perms.READ|Perms.CREATE,new
39 Id("digest",DigestAuthenticationProvider.generateDigest("abcd:123456")))
40 List<ACL> aclList = new ArrayList<ACL>();
41 aclList.add(aclIp);
42 aclList.add(aclDigest);
43 String path=zookeeper.create("/node07","testdata".getBytes(),aclList,
44         CreateMode.PERSISTENT);
45 System.out.println(path);
46 } catch (KeeperException e) {
47 // TODO Auto-generated catch block
48 e.printStackTrace();
49 } catch (InterruptedException e) {// TODO Auto-generated catch block
50     e.printStackTrace();
51     } catch (NoSuchAlgorithmException e) {
52     // TODO Auto-generated catch block
53     e.printStackTrace();
54     }
55     }
56     }
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

```
CreateNodeAuthSync [Java Application] /root/simple/jdk/jdk1.8.0_241/bin/java (Jul 10, 2020, 2:26:29 PM)
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.
接收到的事件:WatchedEvent state:SyncConnected type:None path:null
/node07
```

5.在终端的 zookeeper 服务中，查看是否创建了节点 node07，并查询节点数据。

```
[zk: 47.113.102.106:2181(CONNECTED) 24] ls /
[node07, node3, zookeeper, node1]
[zk: 47.113.102.106:2181(CONNECTED) 25] get /node07
Authentication is not valid : /node07
[zk: 47.113.102.106:2181(CONNECTED) 26]
[root]   bigdata   [eclipse]
```

6.在命令行中利用 digest 方式赋予权限后再次执行查询、修改节点等操作。可查询到节点数据为“testdata”。

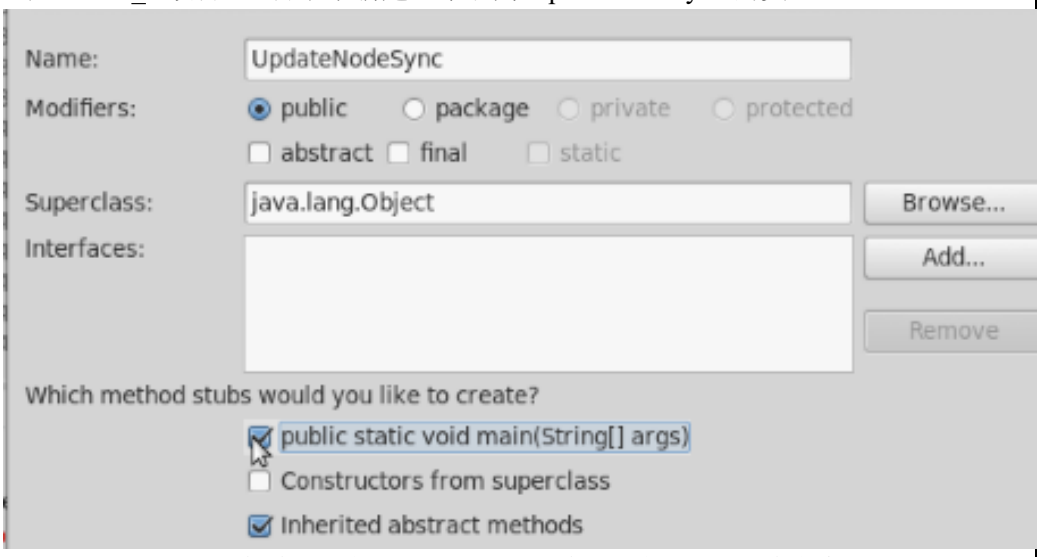
```
[zk: 47.113.102.106:2181(CONNECTED) 26] addauth digest abcd:123456
[zk: 47.113.102.106:2181(CONNECTED) 27] get /node07
testdata
cZxid = 0x200000017
ctime = Fri Jul 10 14:26:29 CST 2020
mZxid = 0x200000017
mtime = Fri Jul 10 14:26:29 CST 2020
pZxid = 0x200000017
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
[zk: 47.113.102.106:2181(CONNECTED) 28]
```

7.修改节点时，发现依然无权限，原因是在程序中利用 digest 方式赋权时未赋予修改权限。

```
[zk: 47.113.102.106:2181(CONNECTED) 28] set /node07 2342233
Authentication is not valid : /node07
```

**3-5 zookeeper javaAPI-修改节点（同步）**

1.在 zkTest_1 项目 src 目录下新建一个名为 UpdateNodeSync 的类。




2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法判断当前状态及事件类型，如果服务连接成功并且事件类型返回 None，则调用 executeZk()方法。

```java
package com.zktest;
import org.apache.zookeeper.KeeperException;
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;
import org.apache.zookeeper.data.Stat;
public class UpdateNodeSync implements Watcher{
public static ZooKeeper zookeeper = null;
public static void main(String[] args)
throws Exception{
zookeeper = new ZooKeeper("47.112.102.106:2181", 5000, new UpdateNodeSyn
Thread.sleep(Integer.MAX_VALUE);
}
@Override
public void process(WatchedEvent event) {
if(event.getState() == KeeperState.SyncConnected){
if(event.getType() == EventType.None && null == event.getPath()){
executeZk();
}
}}
public void executeZk(){
System.out.println("执行 Zookeeper 操作");
```

3.创建 public void executeZk()方法，并在其中利用 zookeeper 对象调用修改节点方法，将 nodel 节点的节点值修改为 123456。

```java
try {
Stat stat = zookeeper.setData("/node3", "123456".getBytes(), -1);
System.out.println(stat);
} catch (KeeperException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

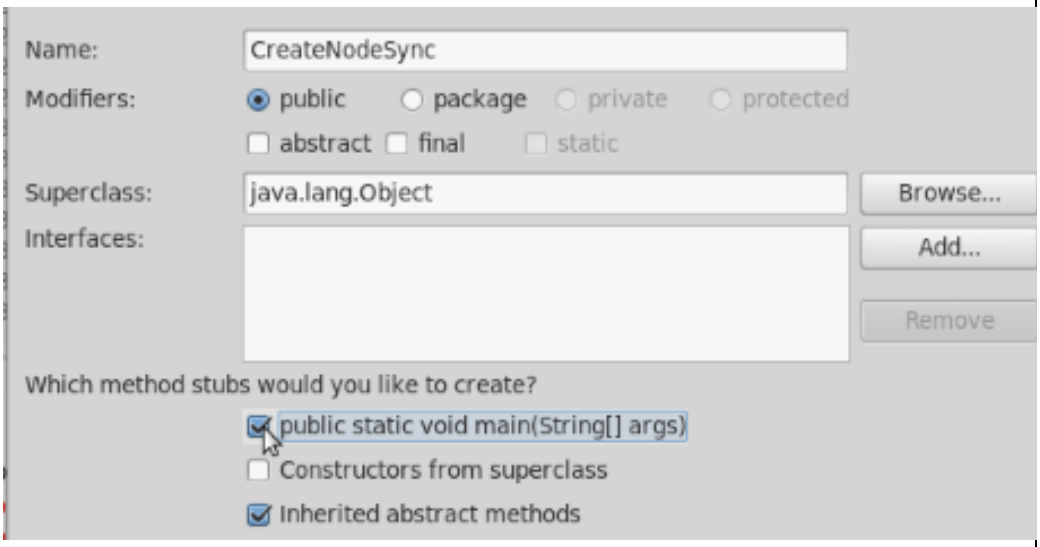
```
UpdateNodeSync [Java Application] /root/simple/jdk/jdk1.8.0_241/bin/java (Jul 10, 2020, 2:45:46
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more inf
执行 Zookeeper 操作
8589934604,8589934618,1594359607537,1594363546983,2,0,0,0,6,0,8589934604
```

5.利用 zookeeper 服务，查看节点的值是否为修改后的值。

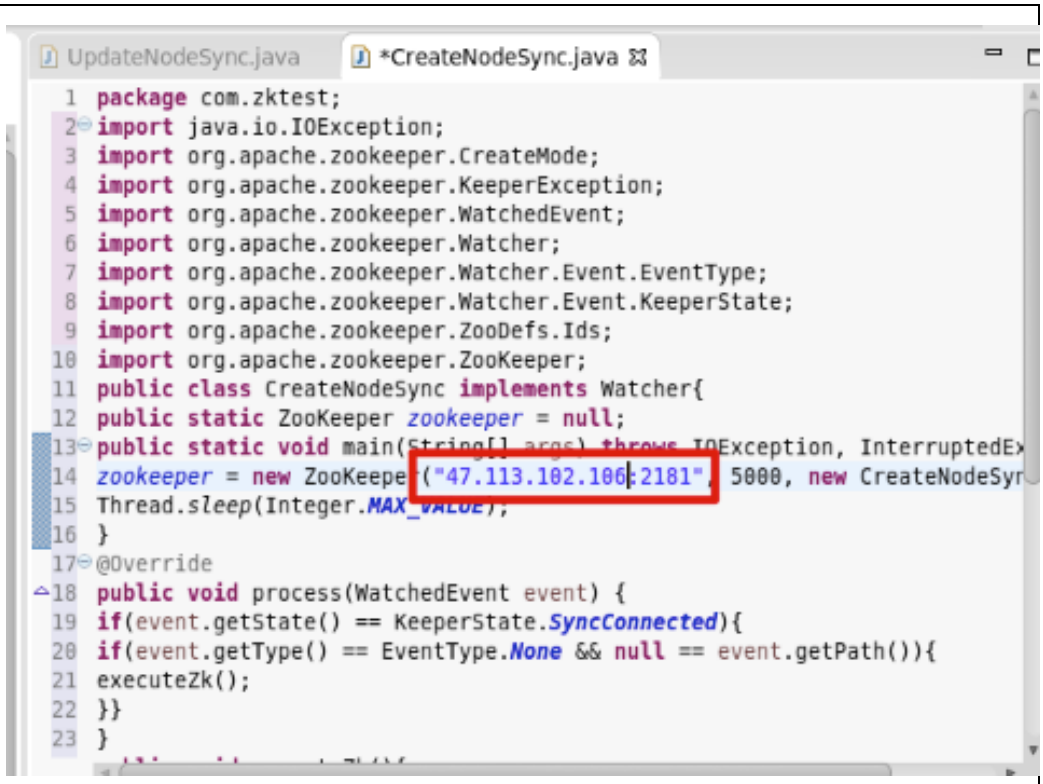
```
[zk: 47.113.102.106:2181(CONNECTED) 29] ls /
[node07, node3, zookeeper, node1]
[zk: 47.113.102.106:2181(CONNECTED) 30] get /node3
123456
cZxid = 0x20000000c
ctime = Fri Jul 10 13:40:07 CST 2020
mZxid = 0x20000001a
mtime = Fri Jul 10 14:45:46 CST 2020
pZxid = 0x20000000c
cversion = 0
dataVersion = 2
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
[zk: 47.113.102.106:2181(CONNECTED) 31]
```

**3-6 zookeeper javaAPI-创建节点（同步）**

1.在 zkTest 项目 src 目录下新建一个名为 CreateNodeSync 的类。



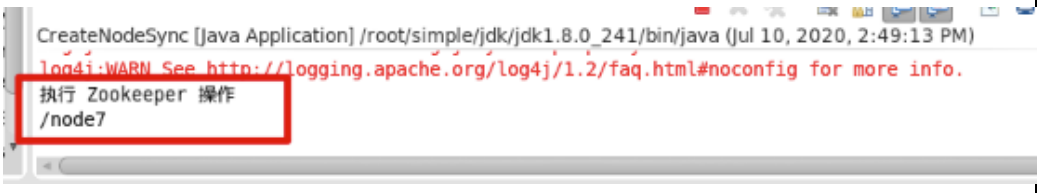

2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法判断当前状态及事件类型，如果服务连接成功并且事件类型返回 None，则调用 executeZk()方法。




3.创建 public void executeZk()方法，并在其中利用 zookeeper 对象创建节点 node7，设置新节点的节点值为 testdata。

```
public void executeZk(){
System.out.println("执行 Zookeeper 操作");
try {
String
path
=
zookeeper.create("/node7","testdata".getBytes(),
Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
System.out.println(path);
} catch (KeeperException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

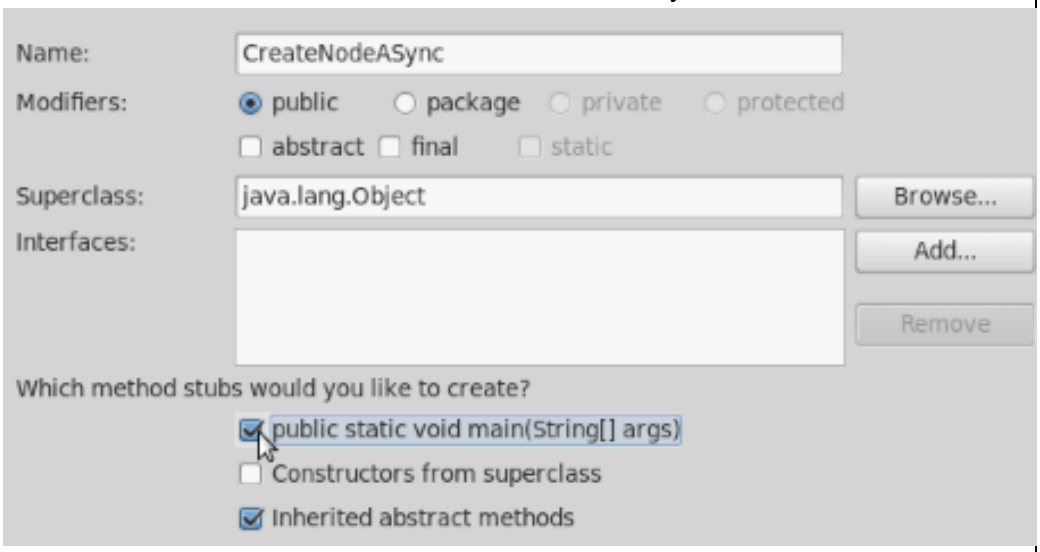



5.在终端中利用 zookeeper 服务查看根节点下是否存在 node7。

```
[zk: 47.113.102.106:2181(CONNECTED) 31] ls /
[node07, node3, zookeeper, node1, node7]
[zk: 47.113.102.106:2181(CONNECTED) 32]
```

**3-7 zookeeper javaAPI-创建节点（异步）**

1.在 zkTest 项目 src 目录下新建一个名为 CreateNodeASync 的类。




2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法判断当前状态及事件类型。利用 zookeeper 对象创建节点 node8，并设置节点值为 test。

```
1  package com.zktest;
2  import java.io.IOException;
3  import org.apache.zookeeper.AsyncCallback;
4  import org.apache.zookeeper.CreateMode;
5  import org.apache.zookeeper.KeeperException;
6  import org.apache.zookeeper.WatchedEvent;
7  import org.apache.zookeeper.Watcher;
8  import org.apache.zookeeper.ZooKeeper;
9  import org.apache.zookeeper.Watcher.Event.EventType;
10 import org.apache.zookeeper.Watcher.Event.KeeperState;
11 import org.apache.zookeeper.ZooDefs.Ids;
12 public class CreateNodeASync implements Watcher{
13 private static ZooKeeper zookeeper;
14 public static void main(String[] args) throws IOException, InterruptedEx
15 zookeeper = new ZooKeeper("47.113.102.106:2181", 5000, new CreateNodeASy
16 Thread.sleep(Integer.MAX_VALUE);
17 }
18 @Override
19 public void process(WatchedEvent event) {
20 // TODO Auto-generated method stub
21 //1、判断 zookeeper 服务是否已经连接
22 if(event.getState() == KeeperState.SyncConnected){
23 //2.判断是否是第一次连接,保证创建节点只执行一次
24 if(event.getType() == EventType.None && null == event.getPath()){
25 zookeeper.create("/node7",
   "test".getBytes()
```

3.创建一个内部类 IStringCallBack 并实现 AsyncCallback.StringCallback 接口，并且实

现其 public void processResult(int rc, String path, Object ctx, String name)方法，在此方法中输出异步接收创建节点返回的值。

```
@Override
public void process(WatchedEvent event) {
// TODO Auto-generated method stub
//1、判断 zookeeper 服务是否已经连接
if(event.getState() == KeeperState.SyncConnected){
//2.判断是否是第一次连接,保证创建节点只执行一次
if(event.getType() == EventType.None && null == event.getPath()){
zookeeper.create("/nod8","test".getBytes(),
Ids.OPEN_ACL_UNSAFE,
CreateMode.PERSISTENT,new IStringCallBack(),"创建");
}
}
}
class IStringCallBack implements AsyncCallback.StringCallback{
@Override
public void processResult(int rc, String path, Object ctx, String name
    StringBuilder sb = new StringBuilder();
    sb.append("RC:"+rc+"\n");
    sb.append("PATH:"+path+"\n");
    sb.append("CTX:"+ctx+"\n");
    sb.append("NAME:"+name);
    System.out.println(sb.toString());
    }
    }
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

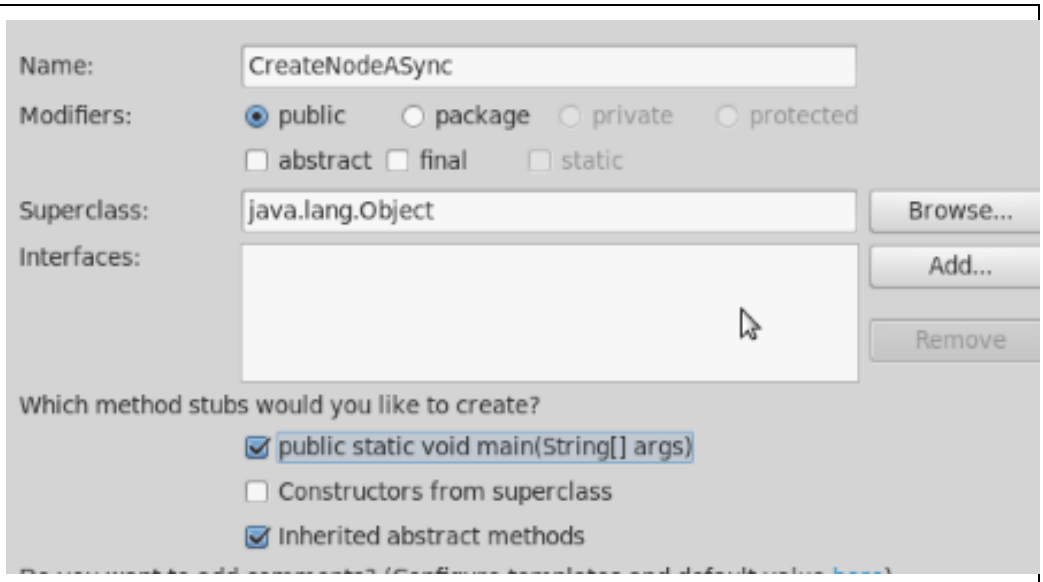
```
CreateNodeASync [Java Application] /root/simple/jdk/jdk1.8.0_241/bin/java (Jul 10, 2020, 2:57:24 PM
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
RC:0
PATH:/nod8
CTX:创建
NAME:/nod8
```

5.在终端中利用 zookeeper 服务查看根节点下是否存在 node8。

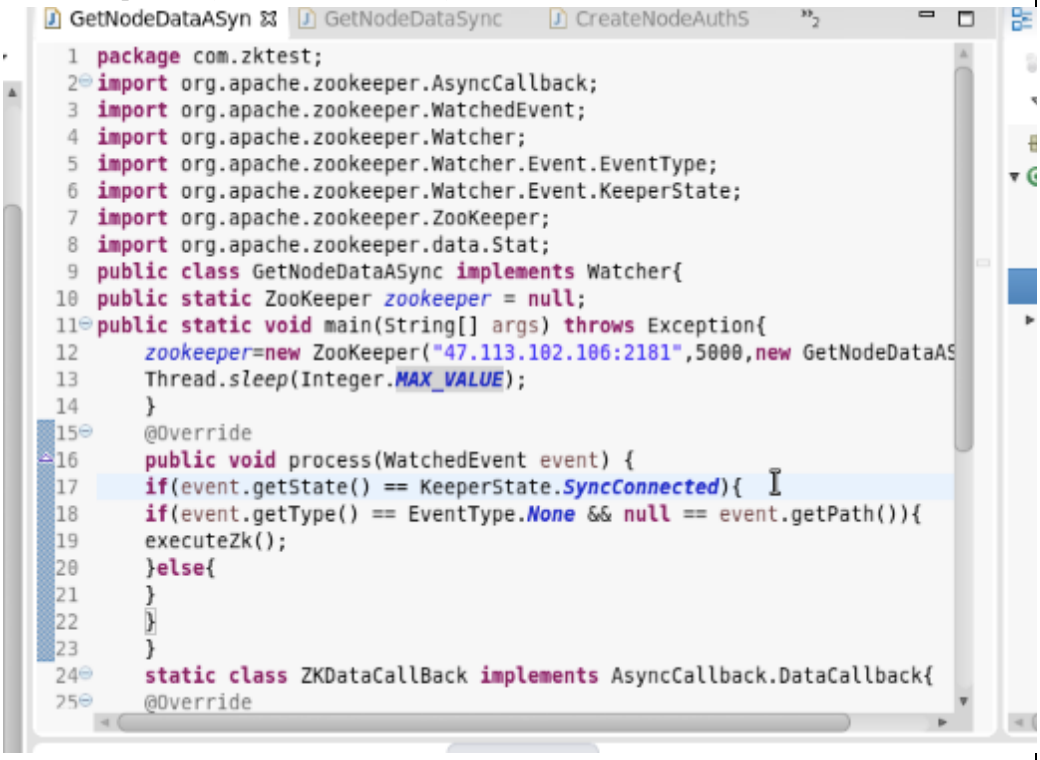
```
[zk: 47.113.102.106:2181(CONNECTED) 32] ls /
[node07, node3, zookeeper, node1, nod8, node7]
[zk: 47.113.102.106:2181(CONNECTED) 33]
```

**3-8 zookeeper javaAPI-获取节点数据（异步）**

1.在 zkTest_1 项目 src 目录下新建一个名为 CreateNodeASync 的类。




2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法判断当前状态及事件类型。




3.创建一个内部类 ZKDataCallBack 并实现 AsyncCallback.DataCallback 接口，并且实现其 public void processResult(int rc, String path, Object ctx, byte[] data, Stat stat)方法，在此方法中输出接收获取节点的值。创建 public void executeZk()方法，在方法中利用 zookeeper 调用获取节点数据方法。

```
    @Override
    public void processResult(int rc, String path, Object ctx, byte[] d
    Stat stat) {
    StringBuilder sb = new StringBuilder();
    sb.append("rc="+rc);
    sb.append("\npath="+path);
    sb.append("\nctx="+ctx);
    sb.append("\ndata="+new String(data));
    sb.append("\nstat="+stat);
    System.out.println(sb.toString());
    }
    }
    public void executeZk(){
    zookeeper.getData("/node3", false, new ZKDataCallBack(),"异步获取节点
    }
    }
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

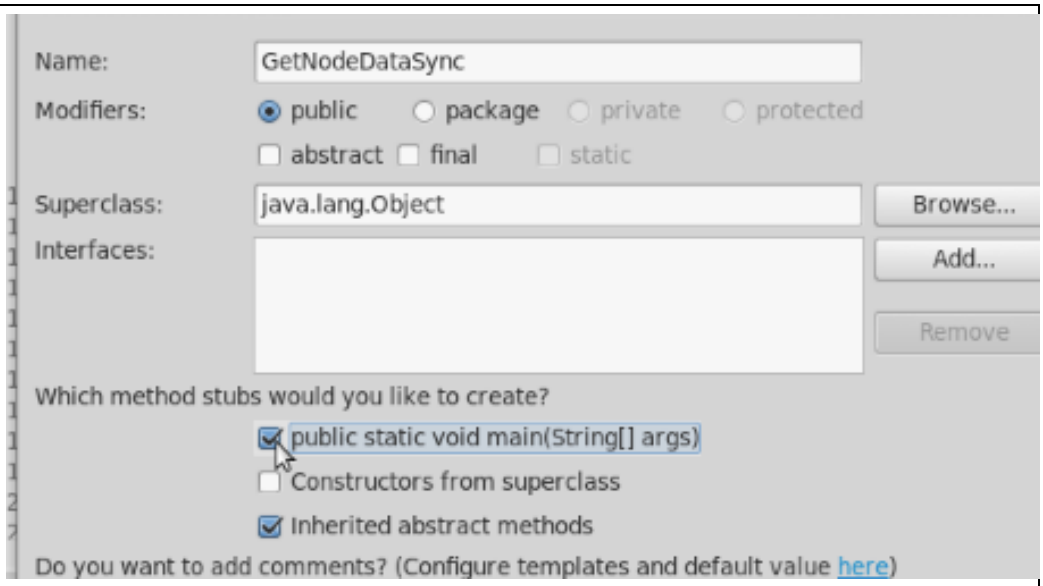
```
GetNodeDataASync [Java Application] /root/simple/jdk/jdk1.8.0_241/bin/java (Jul 10, 2020, 3
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKe
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
rc=0
path=/node3
ctx=异步获取节点数据
data=123456
stat=8589934604,8589934618,1594359607537,1594363546983,2,0,0,0,6,0,8589934604
```

5.在终端中利用 zookeeper 服务查看节点的值是否和控制台打印的节点值一致。

```
[zk: 47.113.102.106:2181(CONNECTED) 34] get /node3
123456
cZxid = 0x20000000c
ctime = Fri Jul 10 13:40:07 CST 2020
mZxid = 0x20000001a
mtime = Fri Jul 10 14:45:46 CST 2020
pZxid = 0x20000000c
cversion = 0
dataVersion = 2
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
[zk: 47.113.102.106:2181(CONNECTED) 35]
```

**3-9 zookeeper javaAPI-获取节点数据（同步）**

1.在 zkTest_1 项目 src 目录下新建一个名为 GetNodeDataSync 的类。




2.声明 zookeeper 对象静态属性，创建 main 方法，并在 main 方法中实例化 zookeeper 对象，实例化时的三个参数分别为连接字符串，超时时间和监听事件。在 zookeeper 对象后输出 zookeeper 的状态并设置程序睡眠状态。在类中实现 watcher 接口，并实现其方法 process(WatchedEvent event)，在此方法判断当前状态及事件类型，如果服务连接成功并且事件类型返回 None，则调用 executeZk()方法。

```
package com.zktest;
import java.io.IOException;
import org.apache.zookeeper.KeeperException;
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;
import org.apache.zookeeper.data.Stat;
public class GetNodeDataSync implements Watcher{
private static ZooKeeper zookeeper;
private static Stat stat;
public static void main(String[] args) throws IOException, InterruptedEx
zookeeper
=
new
ZooKeeper("47.113.102.106:2181",5000,new
GetNodeDataSync());
Thread.sleep(Integer.MAX_VALUE);
}
@Override
public void process(WatchedEvent event) {
System.out.println("调用事件:"+event);
if(event.getState() == KeeperState.SyncConnected){
if(event.getType() == EventType.None && null == event.getPath()){execute
}
```

3.创建 public void executeZk()方法，并在其中利用 zookeeper 对象获取节点数据。

```
@Override
public void process(WatchedEvent event) {
System.out.println("调用事件:"+event);
if(event.getState() == KeeperState.SyncConnected){
if(event.getType() == EventType.None && null == event.getPath()){execute
}
}
}
public void executeZk(){
System.out.println("执行 zookeeper 操作");
try {
System.out.println(new String(zookeeper.getData("/node3", false, stat)))
} catch (KeeperException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
```

4.右键，选择 run as->1 java application 运行程序，运行结果如图。

```
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.
调用事件:WatchedEvent state:SyncConnected type:None path:null
执行 zookeeper 操作
123456
```

5.在终端中利用 zookeeper 服务查看节点的值是否和控制台打印的节点值一致。

```
[zk: 47.113.102.106:2181(CONNECTED) 3] get /node3
123456
cZxid = 0x20000000c
ctime = Fri Jul 10 13:40:07 CST 2020
mZxid = 0x20000001a
mtime = Fri Jul 10 14:45:46 CST 2020
pZxid = 0x20000000c
cversion = 0
dataVersion = 2
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
[zk: 47.113.102.106:2181(CONNECTED) 34]
```

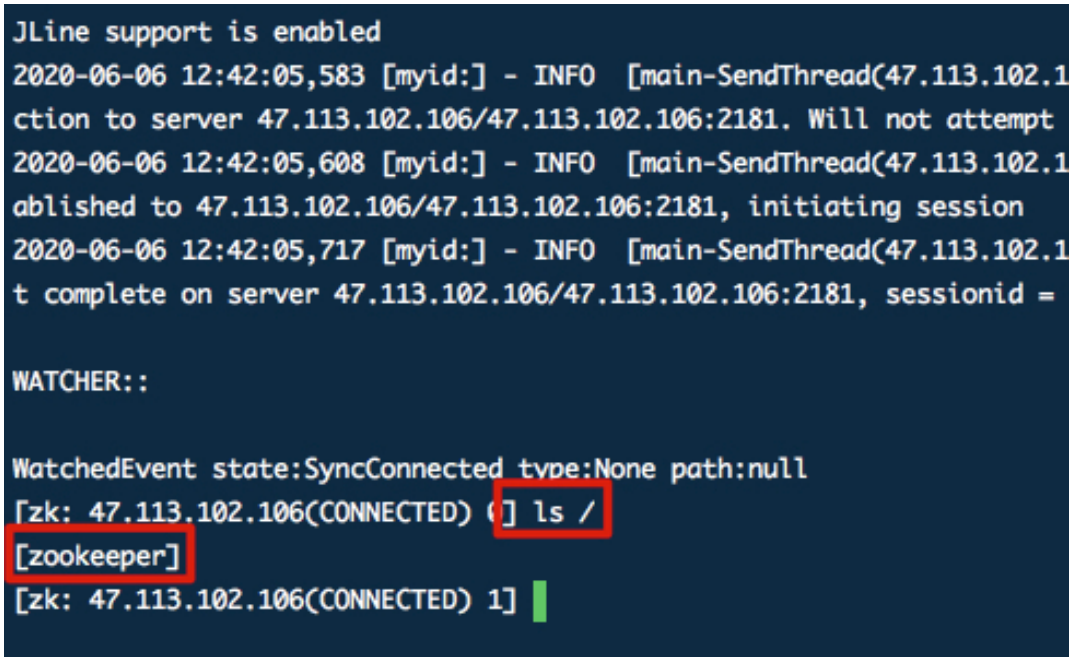
**实验结论与分析：**

**1. 粘贴最终结果图，并对结果做分析说明,.,,.....**

【4-1】

[1] Linux 平台下安装 zookeeper

对 zookeeper 进行了安装，并修改了 zookeeper 的配置文件 zoo.cfg，启动了 zookeeper 服务，并对根节点下的节点进行了查看。




[2] Zookeeper 伪集群模式配置

在该实验中，对 zookeeper 配置文件 zoo.cfg 进行了修改，添加了伪集群配置参数，并启动了集群。

```
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zoo.cfg
Starting zookeeper ... already running as process 14737.
(base) [root@ferry bin]# ./zkServer.sh start zk2.cfg
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zk2.cfg
Starting zookeeper ... STARTED
(base) [root@ferry bin]# jps
14737 QuorumPeerMain
32694 QuorumPeerMain
470 Jps
(base) [root@ferry bin]# ./zkServer.sh start zk3.cfg
ZooKeeper JMX enabled by default
Using config: /root/simple/zookeeper-3.4.14/bin/../conf/zk3.cfg
Starting zookeeper ... STARTED
(base) [root@ferry bin]# jps
14737 QuorumPeerMain
32694 QuorumPeerMain
873 QuorumPeerMain
1018 Jps
(base) [root@ferry bin]#
```

[3] zookeeper 服务端和客户端常用命令

在该实验中，执行了服务端的常用命令如查看服务状态及集群命令、重启 zookeeper 命令、停止 zookeeper 命令；执行了客户端的常用命令如创建节点、修改节点数据、获取节点数据及节点其他信息、查询子节点个数、查看当前节点概况、设置配额和查看配额。

```
[zk: 47.113.102.106(CONNECTED) 10] listquota /nodel
absolute path is /zookeeper/quota/nodel/zookeeper_limits
Output quota for /nodel count=2,bytes=-1
Output stat for /nodel count=1,bytes=2
[zk: 47.113.102.106(CONNECTED) 11]
```

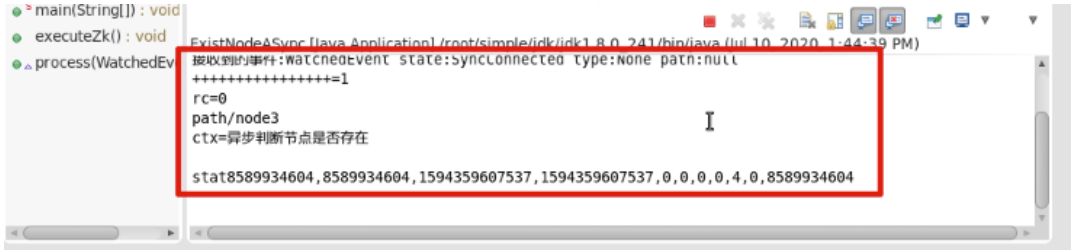
**【4-2】zookeeper javaAPI**

[1] Zookeeper JavaAPI-判断节点是否存在（同步和异步）

启动了 zookeeper 单机服务，新建了 zkTest 项目并对项目进行了相应的 jar 包配置，并分别编写程序同步判断节点是否存在及异步判断节点是否存在，之后用终端的 zookeeper 服务验证程序运行结果的正确性。

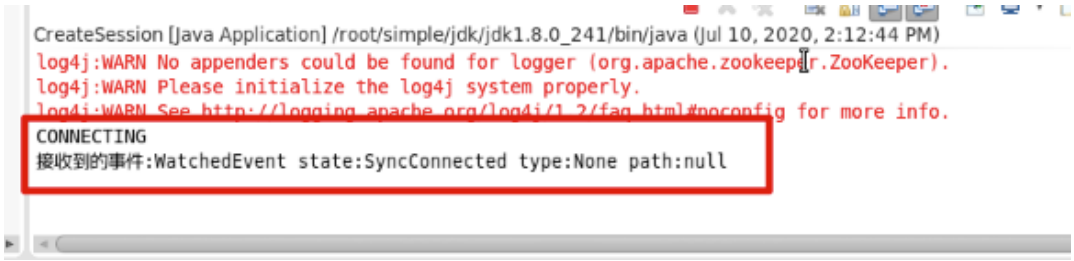

[2] Zookeeper JavaAPI-修改节点（异步）

编写程序对 zookeeper 已存在节点中的数据进行异步修改，并用终端的 zookeeper 服务对程序运行结果进行验证。



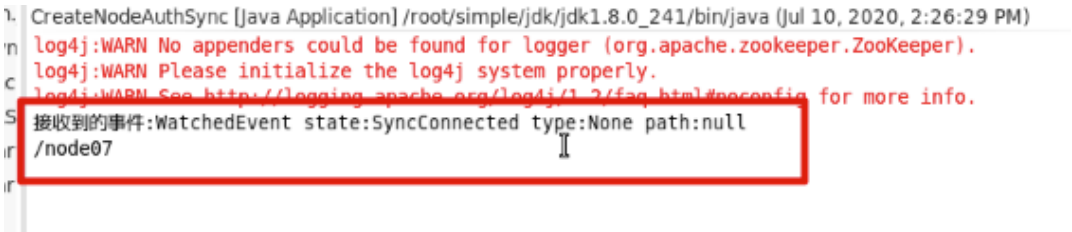

[3] Zookeeper JavaAPI-创建会话

编写程序输出接收到的事件。




[4] Zookeeper JavaAPI - 节点权限控制

编写程序用 ACL 方式进行权限控制，创建新节点并输出创建节点的路径；在 zookeeper 服务中查询节点数据、修改节点数据，并利用 digest 方式赋予权限后再次查询节点数据、修改节点数据。




后面的具体看实验过程:

[5] Zookeeper JavaAPI-修改节点（同步）

[6] Zookeeper JavaAPI-创建节点（同步）

[7] Zookeeper JavaAPI-创建节点（异步）

[8] Zookeeper JavaAPI-获取节点数据（异步）

[9] Zookeeper JavaAPI-获取节点数据（同步）

[10] Zookeeper JavaAPI-获取子节点（异步）

[11] Zookeeper JavaAPI-获取子节点（同步）

[12] Zookeeper JavaAPI-删除节点（异步）

[13] Zookeeper JavaAPI-删除节点（同步）

**2. 谈谈自己对这个实验的体会感受 xxxxx**

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等。Zookeeper 是 hadoop 的一个子项目。在大数据环境下，zookeeper 主要用于保存数据和协调服务。本次实验的集群安装需要再多个节点解压 zookeeper 的安装包，除了配置数据保存的目录，还要配置一个一个节点的 id。在这个过程遇到最主要的问题是：阿里云后台的端口由于没有开放，造成 zookeeper 无法正常启动，最后尝试了很久才找到问题所在。但也学习到了更多的知识。对于 zookeeper javaAPI 操作实验过程也需要自己的配置修改代码， 但也可以更好地理解 zookeeper，

指导教师批阅意见：

成绩评定：