01 Watch

Zookeeper Watches：https://zookeeper.apache.org/doc/current/zookeeperProgrammers.html#ch_zkWatches

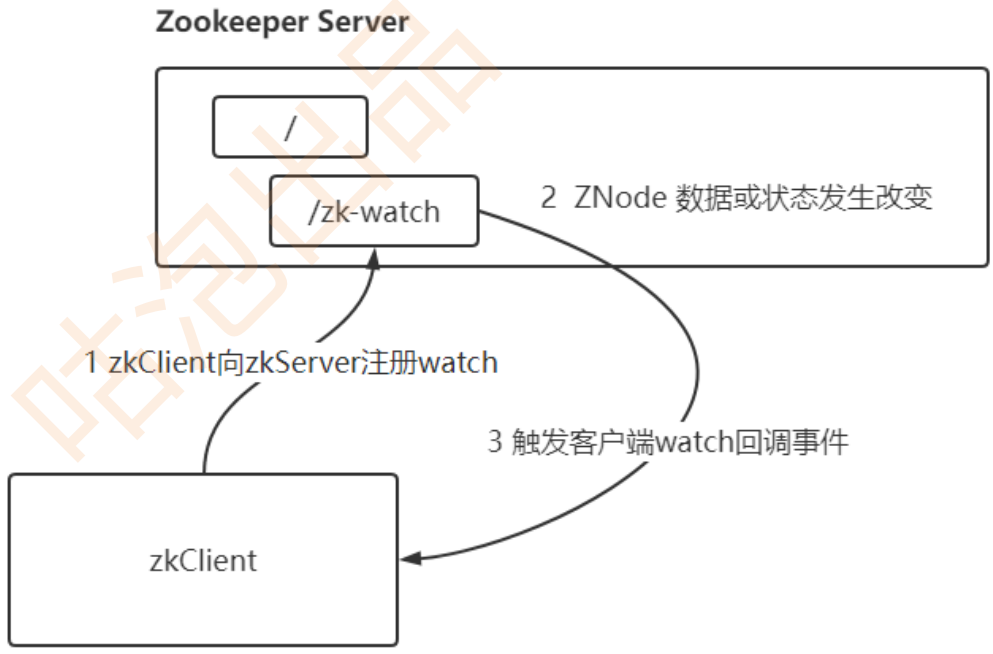
All of the read operations in ZooKeeper - **getData()**, **getChildren()**, and **exists()** - have the option of setting a watch as a side effect. Here is ZooKeeper's definition of a watch: a watch event is one-time trigger, sent to the client that set the watch, which occurs when the data for which the watch was set changes.

Conditional updates and watches：https://zookeeper.apache.org/doc/current/zookeeperOver.html#Conditional+updates+and+watches

ZooKeeper supports the concept of **watches**. Clients can set a watch on a znode. A watch will be triggered and removed when the znode changes. When a watch is triggered, the client receives a packet saying that the znode has changed. If the connection between the client and one of the ZooKeeper servers is broken, the client will receive a local notification.

News in 3.6.0 Clients can also set permanent, recursive watches on a znode that are not removed when triggered and that trigger for changes on the registered znode as well as any children znodes recursively.

1.1 图解watch



1.2 支持watch的命令

> 可以发现watch的注册基本上都是读事件

```
help
config [-c] [-w] [-s]
get [-s] [-w] path
ls [-s] [-w] [-R] path
stat [-w] path
```

1.3 watch体验之旅

> 分类:
>
> get stat 监听节点数据的变化
>
> ls (-R) 针对(子)节点的变化

1.3.1 监听节点数据变化

```
# 创建节点并添加监听
create /zk-watch 111
get -w /zk-watch

# 来到另外一个客户端(其实用当前客户端也行,只是为了便于理解)
set /zk-watch 222

# 观察第一个zkClient的变化，发现收到通知
WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/zk-watch

# 收到watch通知之后，就可以进行对应的业务逻辑处理。但如果再修改/zk-watch的值，发现就不会收到watch通知了，因为该命令下的watch通知是一次性的，要想再收到，得继续添加watch监听
get -w /zk-watch
```

1.3.2 监听(子)节点的创建和删除

```
# 创建子节点并对父节点添加watch
create /zk-watch/sub1
get -w /zk-watch

# 修改/zk-watch/sub1节点的数据值，发现watch并没有生效，因为get只监听单个节点
set /zk-watch/sub1 111

# 通过ls添加对(子)节点的增加和删除监听
ls -w /zk-watch
create /zk-watch/sub2 111

# 继续添加zk-watch节点的子节点，发现并没有收到通知，因为ls也是一次性的
create /zk-watch/sub3 111
```

1.3.3 添加永久监听

> addWatch [-m mode] path # optional mode is one of [PERSISTENT, PERSISTENT_RECURSIVE] - **default is PERSISTENT_RECURSIVE**

```
create /zk-watch-update 666
addWatch /zk-watch-update

set /zk-watch-update 999
set /zk-watch-update 888
create /zk-watch-update/sub1
create /zk-watch-update/sub2
delete /zk-watch-update/sub1
set /zk-watch-update/sub2 222
create /zk-watch-update/sub2/sub1 111
delete /zk-watch-update/sub2/sub1
delete /zk-watch-update/sub2
delete /zk-watch-update
```

02 ACL

Zookeeper access control using ACLs：https://zookeeper.apache.org/doc/current/zookeeperProgrammers.html#sc_ZooKeeperAccessControl

2.1 ACL的组成

scheme：id ： permission

（1）scheme：表示策略

（2）id：表示允许访问的用户

（3）permission：表示访问的权限

2.2 ACL Permissions

- CREATE: you can create a child node
- READ: you can get data from a node and list its children.
- WRITE: you can set data for a node
- DELETE: you can delete a child node
- ADMIN: you can set permissions

2.3 ACL schemes

（1）world

该scheme只有一个id，为anyone，表示所有人，格式为 world:anyone:permission

（2）auth

该scheme表示需要认证登录，也就是对应注册的用户需拥有权限才可以访问，格式为 auth:user:password:permission

（3）digest

该scheme表示需要密码加密才能访问，格式为 digest:username:BASE64(password):permission

（4）ip

该scheme表示指定的ip才能访问，格式为 ip:localhost:permission

（5）super

该scheme表示超管，拥有所有权限

2.4 ACL体验之旅

```
# 创建节点并查看权限    'world,'anyone
: cdrwa
create /zk-acl 111
getAcl /zk-acl

# 设置某个用户对某个节点的权限
create /zk-jack 666
setAcl /zk-jack auth:jack:123:cdrwa
# 表示该用户还没有在zk中注册，注册一下
addauth digest jack:123
setAcl /zk-jack auth:jack:123:cdrwa
getAcl /zk-jack

# 这样一来，对于/zk-jack节点的操作，就需要先登录一下，打开另外一个客户端，执行如下命令，提示:
Insufficient permission : /zk-jack
ls /zk-jack
get /zk-jack

# 授权
addauth digest jack:123
get /zk-jack
```

03 Monitoring

3.1 The Four Letter Words

The Four Letter Words：https://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_4lw

ZooKeeper responds to a small set of commands. Each command is composed of four letters.

New in 3.5.3: Four Letter Words need to be explicitly white listed before using. Please refer to **4lw.commands.whitelist** described in cluster configuration section for details. Moving forward, Four Letter Words will be deprecated, please use AdminServer instead.

说白了就是由4个字母组成的命令，可以通过telnet或ncat使用客户端向zkServere发出命令。

（1）修改zoo.cfg文件

```
（1）打开zoo.cfg文件
（2）添加一行配置:
    4lw.commands.whitelist=*
    echo "4lw.commands.whitelist=*" >> zoo.cfg
（3）重启zk服务
    zkServer.sh restart
```

（2）体验一下

```
（1）安装ncat:  yum install -y nc

（2）查看节点是否正常
    echo ruok | ncat localhost 2181

（3）查看节点相关配置
    echo conf | ncat localhost 2181

（4）查看节点更详细的状态
    echo stat | ncat localhost 2181

（5）查看节点更详细的状态
    echo srvr | ncat localhost 2181

（6）查看临时节点
    echo dump| ncat localhost 2181

（7）查看watch
    get -w /zk-watch
    echo wchc| ncat localhost 2181

（8）查看server的env
    echo envi| ncat localhost 2181
```

3.2 The AdminServer

The AdminServer：https://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_adminserver

New in 3.5.0: The AdminServer is an embedded Jetty server that provides an HTTP interface to the four-letter word commands. By default, the server is started on port 8080, and commands are issued by going to the URL "/commands/[command name]", e.g., http://localhost:8080/commands/stat.

3.3 JMX

JMX：Java Management Extensions

（1）zkServer.sh中配置JMX

```
ZOOMAIN="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=8888
-Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false
 -Djava.rmi.server.hostname=192.168.0.8
 -Dcom.sun.management.jmxremote.local.only=false
org.apache.zookeeper.server.quorum.QuorumPeerMain"
```

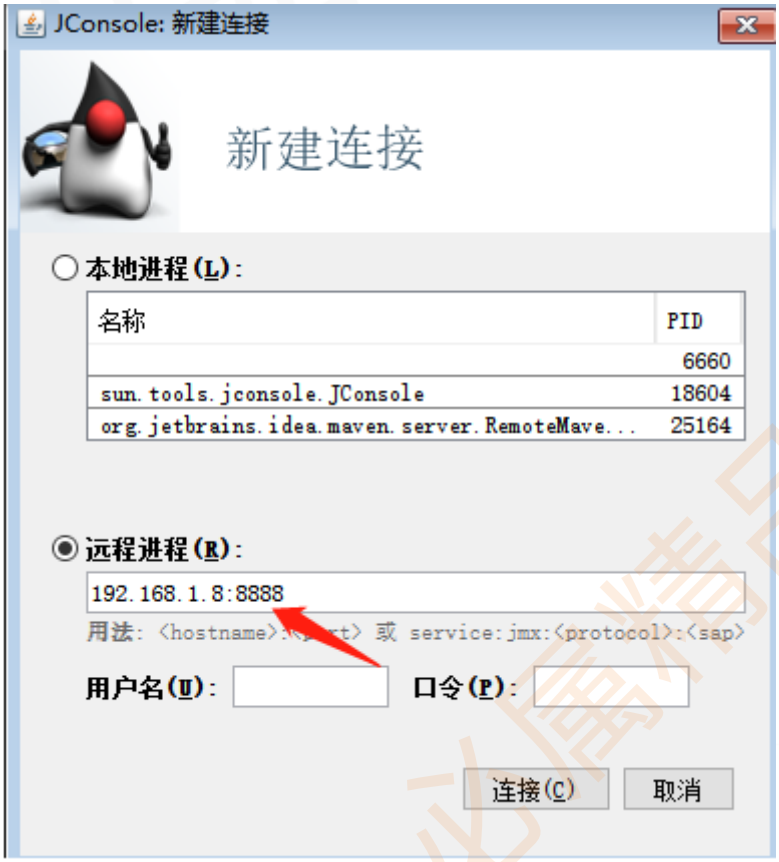
（2）重启zkServer

```
zkServer.sh restart
```

（3）查看8888端口监听

```
lsof -i:8888
```

（4）打开本地jconsole，连接指定JMX的ip和port：192.168.0.8:8888



04 序列化和反序列化

4.1 常见的序列化方式

json、protobuf、thrif、avro等

4.2 Zookeeper序列化方式

Zookeeper使用的序列化方式是jute，Java类需要实现Record接口，底层使用的是DataOutput和DataInput。

（1）引入依赖

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.24</version>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.apache.zookeeper</groupId>
```

```
    <artifactId>zookeeper</artifactId>
    <version>3.7.1</version>
</dependency>
```

(2) 定义Java类并测试

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Person implements Record {
    private String username;
    private Integer age;
    @Override
    public void serialize(OutputArchive archive, String tag) throws IOException
{
        archive.startRecord(this, tag);
        archive.writeString(username, "username");
        archive.writeInt(age, "age");
        archive.endRecord(this, tag);
    }
    @Override
    public void deserialize(InputArchive archive, String tag) throws IOException
{
        archive.startRecord(tag);
        username = archive.readString("username");
        age = archive.readInt("age");
        archive.endRecord(tag);
    }

    public static void main(String[] args) throws IOException {
        // 序列化
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
        BinaryOutputArchive binaryOutputArchive =
BinaryOutputArchive.getArchive(byteArrayOutputStream);
        new Person("Jack", 16).serialize(binaryOutputArchive, "person");
        ByteBuffer byteBuffer =
ByteBuffer.wrap(byteArrayOutputStream.toByteArray());

        // 反序列化
        ByteBufferInputStream byteBufferInputStream = new
ByteBufferInputStream(byteBuffer);
        BinaryInputArchive binaryInputArchive =
BinaryInputArchive.getArchive(byteBufferInputStream);
        Person person = new Person();
        person.deserialize(binaryInputArchive, "person");
        System.out.println(person.toString());

        // 关闭资源
        byteArrayOutputStream.close();
        byteBufferInputStream.close();
    }
}
```

05 快照数据与事务日志

快照数据：记录所有ZNode节点及数据某一时刻的快照，保存在zoo.cfg文件配置项的dataDir目录的version-2中，格式为snapshot.zxid

事务日志：记录每一次事务操作的记录，保存在dataLogDir[dataDir]目录的version-2中，格式为log.zxid

```
zkSnapShotToolkit.sh snapshot.zxid
zkTxnLogToolkit.sh log.zxid
```

5.1 zk server第一次启动

zk server第一次启动时，因为此时zxid的值为0，会生成一个snapshot.0的数据快照文件，对应的源码是FileTnxSnapLog#save方法。

文件的大小不会进行预分配，而是取决于内存DataTree的大小。

```
snapLog.serialize(dataTree, sessionsWithTimeouts, snapshotFile, syncSnap);
    new SnapshotInfo(Util.getZxidFromName(snapShot.getName(),
SNAPSHOT_FILE_PREFIX),snapShot.lastModified() / 1000)
```

查看快照日志文件内容

```
zkSnapShotToolkit.sh snapshot.0
```

5.2 第一次创建事务日志文件

使用prettyZoo连接zk server，发现生成了一个log.1的事务日志文件，对应的源码是FileTxnLog#append方法。

文件的大小会进行预分配，也就是FilePadding#preAllocaSize = 65536 * 1024 Byte = 64M 也就是说每个事务日志文件的默认大小是64M，可以直接在windows中查看文件。

如果事务日志文件的空间剩余不足4KB，则会再次预分配64M的磁盘空间。

```
zkTxnLogToolkit.sh log.1
```

5.3 新建数据快照文件和事务日志文件

每进行一次事务操作，事务日志中都会增加一条记录，当经过snapCount的过半随机次数的事务写入之后，就会触发一次快照数据文件生成，同时也会新生成一个事务日志文件。

注意：每一次重新启动zk server，如果之前有zxid的变化，则也会创建一个新的快照数据文件，同时在后续的事务操作中，也会新建一个新的事务日志文件。

具体源码见SyncRequestProcessor#shouldSnapshot()方法

```
private boolean shouldSnapshot() {
    int logCount = zks.getZKDatabase().getTxnCount();
    long logSize = zks.getZKDatabase().getTxnSize();
    return (logCount > (snapCount / 2 + randRoll))
           || (snapSizeInBytes > 0 && logSize > (snapSizeInBytes / 2 +
randSize));
}
```

5.4 数据快照文件和事务日志文件的清理

> QuorumPeerConfig

purgeInterval=0	触发自动清理的时间间隔，单位是小时，默认值为0，表示不开启自动清理的功能
snapRetainCount=3	自动清理保留3个事务日志和快照数据

5.5 总结

针对每一次事务操作，都会将其保存到事务日志文件中，同时会将数据的变化应用到内存DataTree中。当经过了一定次数的事务操作后，则会将内存DataTree中的全量数据保存到数据快照文件中。