zookeeper详细功能介绍与客户端框架使用

鲁班学院-周瑜

曾参与大型电商平台、互联网金融产品等多家互联网公司的开发，曾就职于大众点评，任项目经理等职位，参与并主导千万级并发电商网站与系统架构搭建

> 学海无涯，我们一起勉力前行
>
> 课程讲师：**周瑜老师QQ：3413298904**
>
> 往期课程资料：**木兰老师QQ: 2746251334**
>
> VIP课程咨询：**安其拉老师QQ: 3164703201**

节点类型(znode)

1. 持久节点，所谓持久节点，是指在节点创建后，就一直存在，直到有删除操作来主动清除这个节点。
2. 临时节点，和持久节点不同的是，临时节点的生命周期和客户端会话绑定。也就是说，如果客户端会话失效，那么这个节点就会自动被清除掉。注意，这里提到的是会话失效，而非连接断开。另外，在临时节点下面不能创建子节点。
3. 持久顺序节点，这类节点的基本特性和持久节点是一致的。额外的特性是，在ZK中，每个父节点会为他的第一级子节点维护一份时序，会记录每个子节点创建的先后顺序。基于这个特性，在创建子节点的时候，可以设置这个属性，那么在创建节点过程中，ZK会自动为给定节点名加上一个数字后缀，作为新的节点名。这个数字后缀的范围是整型的最大值。
4. 临时顺序节点，类似临时节点和顺序节点

> zookeeper默认对每个结点的最大数据量有一个上限是1M

Stat

ZooKeeper命名空间中的每个znode都有一个与之关联的stat结构，类似于Unix/Linux文件系统中文件的stat结构。 znode的stat结构中的字段显示如下，各自的含义如下：

- cZxid：创建znode的事务ID。
- mZxid：最后修改znode的事务ID。
- pZxid：最后修改添加或删除子节点的事务ID。
- ctime：表示从1970-01-01T00:00:00Z开始以毫秒为单位的znode创建时间。
- mtime：表示从1970-01-01T00:00:00Z开始以毫秒为单位的znode最近修改时间。
- dataVersion：表示对该znode的数据所做的更改次数。
- cversion：这表示对此znode的子节点进行的更改次数。
- aclVersion：表示对此znode的ACL进行更改的次数。
- ephemeralOwner：如果znode是ephemeral类型节点，则这是znode所有者的 session ID。 如果znode不是ephemeral节点，则该字段设置为零。
- dataLength：这是znode数据字段的长度。
- numChildren：这表示znode的子节点的数量。

Zxid-后面讲zab算法会着重讲

类似于RDBMS中的事务ID，用于标识一次更新操作的Proposal ID。为了保证顺序性，该zkid必须单调递增。因此ZooKeeper使用一个64位的数来表示，高32位是Leader的epoch，从1开始，每次选出新的Leader，epoch加一。低32位为该epoch内的序号，每次epoch变化，都将低32位的序号重置。这样保证了zkid的全局递增性。

Watch

一个zk的节点可以被监控，包括这个目录中存储的数据的修改，子节点目录的变化，一旦变化可以通知设置监控的客户端，这个功能是zookeeper对于应用最重要的特性，通过这个特性可以实现的功能包括配置的集中管理，集群管理，分布式锁等等。

watch机制官方说明：一个Watch事件是一个**一次性**的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。

可以注册watcher的方法：getData、exists、getChildren。

可以触发watcher的方法：create、delete、setData。连接断开的情况下触发的watcher会丢失。

一个Watcher实例是一个回调函数，被回调一次后就被移除了。如果还需要关注数据的变化，需要再次注册watcher。

New ZooKeeper时注册的watcher叫default watcher，它不是一次性的，只对client的连接状态变化作出反应。

	event For "/path"	**event For "/path/child"**
create(“/path”)	EventType.NodeCreated	无
delete(“/path”)	EventType.NodeDeleted	无
setData(“/path”)	EventType.NodeDataChanged	无
create(“/path/child”)	EventType.NodeChildrenChanged（getChild）	EventType.NodeCreated
delete(“/path/child”)	EventType.NodeChildrenChanged（getChild）	EventType.NodeDeleted
setData(“/path/child”)	无	EventType.NodeDataChanged

event For “/path”	**Default Watcher**	**exists(“/path”)**	**getData(“/path”)**	**getChildren(“/path”)**
EventType.None	√	√	√	√
EventType.NodeCreated		√	√	
EventType.NodeDeleted		√	√	
EventType.NodeDataChanged		√	√	
EventType.NodeChildrenChanged				√

> exits和getData设置数据监视，而getChildren设置子节点监视

常用命令

创建节点（znode）

用给定的路径创建一个节点。flag参数指定创建的节点是临时的，持久的还是顺序的。默认情况下，所有节点都是持久的。

当会话过期或客户端断开连接时，临时节点（flag：-e）将被自动删除。

顺序节点保证节点路径将是唯一的。

ZooKeeper集合将向节点路径填充10位序列号。例如，节点路径 /myapp 将转换为/myapp0000000001，下一个序列号将为/myapp0000000002。如果没有指定flag，则节点被认为是持久的。

语法：

```
create /path data
```

示例：

```
create /FirstNode first
```

输出：

```
[zk: localhost:2181(CONNECTED) 3] create /FirstNode first
Created /FirstNode
```

要创建顺序节点，请添加flag：-s，如下所示。

语法：

```
create -s /path data
```

示例：

```
create -s /FirstNode second
```

输出：

```
[zk: localhost:2181(CONNECTED) 4] create -s /FirstNode second
Created /FirstNode0000000018
```

要创建临时节点，请添加flag：-e ，如下所示。

语法：

```
create -e /path data
```

示例：

```
create -e /FirstNode-ephemeral ephemeral
```

输出：

```
[zk: localhost:2181(CONNECTED) 6] create -e /FirstNode-ephemeral ephemeral
Created /FirstNode-ephemeral
```

记住当客户端断开连接时，临时节点将被删除。你可以通过退出ZooKeeper CLI，然后重新打开CLI来尝试。

获取数据

它返回节点的关联数据和指定节点的元数据。你将获得信息，例如上次修改数据的时间，修改的位置以及数据的相关信息。此CLI还用于分配监视器以显示数据相关的通知。

语法：

```
get /path
```

示例：

`get /FirstNode`

输出：

```
[zk: localhost:2181(CONNECTED) 7] get /FirstNode
first
cZxid = 0xa2
ctime = Wed Dec 12 13:29:14 CST 2018
mZxid = 0xa2
mtime = Wed Dec 12 13:29:14 CST 2018
pZxid = 0xa2
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 5
numChildren = 0
```

要访问顺序节点，必须输入znode的完整路径。

示例：

`get /FirstNode0000000018`

输出：

```
[zk: localhost:2181(CONNECTED) 9] get /FirstNode0000000018
second
cZxid = 0xa3
ctime = Wed Dec 12 13:30:44 CST 2018
mZxid = 0xa3
mtime = Wed Dec 12 13:30:44 CST 2018
pZxid = 0xa3
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

设置数据

设置指定znode的数据。完成此设置操作后，你可以使用 get CLI命令检查数据。

语法：

`set /path /data`

示例：

`set /FirstNode first_update`

输出：

```
[zk: localhost:2181(CONNECTED) 12] set /FirstNode first_update

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/FirstNode
cZxid = 0xa2
ctime = Wed Dec 12 13:29:14 CST 2018
mZxid = 0xa6
mtime = Wed Dec 12 13:51:39 CST 2018
pZxid = 0xa2
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 0
```

创建子节点

创建子节点类似于创建新的znode。唯一的区别是，子znode的路径也将具有父路径。

语法：

`create /parent/path/subnode/path data`

示例：

`create /FirstNode/Child firstchildren`

输出：

```
[zk: localhost:2181(CONNECTED) 13] create /FirstNode/Child firstchildren
Created /FirstNode/Child
[zk: localhost:2181(CONNECTED) 14] create /FirstNode/Child2 secondchildren
Created /FirstNode/Child2
```

列出子节点

此命令用于列出和显示znode的子项。

语法：

`ls /path`

实例：

`ls /FirstNode`

输出：

```
[zk: localhost:2181(CONNECTED) 15] ls /FirstNode
[Child2, Child]
```

检查状态

状态描述指定的znode的元数据。它包含时间戳，版本号，ACL，数据长度和子znode等细项。

语法：

`stat /path`

示例：

`stat /FirstNode`

输出：

```
[zk: localhost:2181(CONNECTED) 16] stat /FirstNode
cZxid = 0xa2
ctime = Wed Dec 12 13:29:14 CST 2018
mZxid = 0xa6
mtime = Wed Dec 12 13:51:39 CST 2018
pZxid = 0xa8
cversion = 2
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 2
```

移除Znode

移除指定的znode并递归其所有子节点。只有在znode可用的情况下才会发生。

语法：

`rmr /path`

示例：

`rmr /FirstNode`

输出：

```
[zk: localhost:2181(CONNECTED) 17] rmr /FirstNode
[zk: localhost:2181(CONNECTED) 18] get /FirstNode
Node does not exist: /FirstNode
```

删除(delete /path)命令类似于 remove 命令，但是只适用于没有子节点的znode。

ACL

zk做为分布式架构中的重要中间件，通常会在上面以节点的方式存储一些关键信息，默认情况下，所有应用都可以读写任何节点，在复杂的应用中，这不太安全，ZK通过ACL机制来解决访问权限问题。

- ZooKeeper的权限控制是基于每个znode节点的，需要对每个节点设置权限
- 每个znode支持设置多种权限控制方案和多个权限
- 子节点不会继承父节点的权限，客户端无权访问某节点，但可能可以访问它的子节点

ACL 权限控制，使用：schema:id:permission 来标识，主要涵盖 3 个方面：

- 权限模式（Schema）：鉴权的策略
- 授权对象（ID）
- 权限（Permission）

schema

- world：只有一个用户：anyone，代表所有人（默认）
- ip：使用IP地址认证
- auth：使用已添加认证的用户认证
- digest：使用“用户名:密码”方式认证

id

授权对象ID是指，权限赋予的用户或者一个实体，例如：IP 地址或者机器。授权模式 schema 与 授权对象 ID 之间关系：

- world：只有一个id，即anyone
- ip: 通常是一个ip地址或地址段，比如192.168.0.110或192.168.0.1/24
- auth：用户名
- digest：自定义：通常是"username:BASE64(SHA-1(username:password))"

权限

- CREATE, 简写为c，可以创建子节点
- DELETE，简写为d，可以删除子节点（仅下一级节点），注意不是本节点
- READ，简写为r，可以读取节点数据及显示子节点列表
- WRITE，简写为w，可设置节点数据
- ADMIN，简写为a，可以设置节点访问控制列表

查看ACL

查看ACL

```
getAcl /parent
```

返回

```
[zk: localhost:2181(CONNECTED) 122] getAcl /parent
'world,'anyone
: cdrwa
```

默认创建的节点的权限是最开放的，所有都可以增删查改管理。

设置ACL

设置节点对所有人都有写和管理权限

```
setAcl /parent world:anyone:wa
```

所以去读取数据的时候会提示

```
[zk: localhost:2181(CONNECTED) 124] get /parent
Authentication is not valid : /parent
```

先添加用户：

```
addauth digest zhangsan:12345
```

再设置权限，这个节点只有zhangsan这个用户拥有所有权限

```
setAcl /parent auth:zhangsan:123456:rdwca
```

Curator客户端

1. Recipes：Zookeeper典型应用场景的实现，这些实现是基于Curator Framework。
2. Framework：Zookeeper API的高层封装，大大简化Zookeeper客户端编程，添加了例如Zookeeper连接管理、重试机制等。
3. Utilities：为Zookeeper提供的各种实用程序。
4. Client：Zookeeper client的封装，用于取代原生的Zookeeper客户端（ZooKeeper类），提供一些非常有用的客户端特性。
5. Errors：Curator如何处理错误，连接问题，可恢复的例外等。

Curator主要解决了三类问题

1. 封装ZooKeeper client与ZooKeeper server之间的连接处理
2. 提供了一套Fluent风格的操作API
3. 提供ZooKeeper各种应用场景(recipe, 比如共享锁服务, 集群领导选举机制)的抽象封装

原生Zookeeper客户端存在的问题

客户端在连接服务端是会设置一个sessionTimeout（session过期时间），并且客户端会给服务端发送心跳以刷新服务端的session时间。

当网络断开后，服务端无法接受到心跳，会进行session倒计时，判断是否超过了session过期时间，一旦超过了过期时间，就发送了Session过期，就算后来网络通了，客户端从新连接上了服务端，就会接受session过期的事件，从而删除临时节点和watcher等等。原生客户端不会重建session。