ZooKeeper API

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程

ZooKeeper 类

ZooKeeper Java 代码主要使用 org.apache.zookeeper.ZooKeeper 这个类使用 ZooKeeper 服务。

```
ZooKeeper(connectString, sessionTimeout, watcher)
```

connectString：使用逗号分隔的列表，每个 ZooKeeper 节点是一个 host:port 对，host 是机器名或者 IP 地址，port 是 ZooKeeper 节点使用的端口号。 会任意选取 connectString 中的一个节点建立连接。

sessionTimeout：session timeout 时间。

watcher: 用于接收到来自 ZooKeeper 集群的所有事件。

ZooKeeper 主要方法

- create(path, data, flags)：创建一个给定路径的 znode，并在 znode 保存 data[]的数据，flags 指定 znode 的类型。
- delete(path, version)：如果给定 path 上的 znode 的版本和给定的 version 匹配，删除 znode。
- exists(path, watch)：判断给定 path 上的 znode 是否存在，并在 znode 设置一个 watch。
- getData(path, watch)：返回给定 path 上的 znode 数据，并在 znode 设置一个 watch。
- setData(path, data, version)：如果给定 path 上的 znode 的版本和给定的 version 匹配，设置 znode 数据。
- getChildren(path, watch)：返回给定 path 上的 znode 的孩子 znode 名字，并在 znode 设置一个 watch。
- sync(path)：把客户端 session 连接节点和 leader 节点进行同步。

方法说明

- 所有获取 znode 数据的 API 都可以设置一个 watch 用来监控 znode 的变化。
- 所有更新 znode 数据的 API 都有两个版本：无条件更新版本和条件更新版本。如果 version 为 -1，更新为条件更新。否则只有给定的 version 和 znode 当前的 version 一样，才会进行更新，这样的更新是条件更新。
- 所有的方法都有同步和异步两个版本。同步版本的方法发送请求给 ZooKeeper 并等待服务器的响应。异步版本把请求放入客户端的请求队列，然后马上返回。异步版本通过 callback 来接受来自服务端的响应。

ZooKeeper 代码异常处理

所有同步执行的 API 方法都有可能抛出以下两个异常：

- KeeperException：表示 ZooKeeper 服务端出错。 KeeperException 的子类 ConnectionLossException 表示客户端和当前连接的 ZooKeeper 节点断开了连接。网络分区和 ZooKeeper 节点失败都会导致这个异常出现。发生此异常的时机可能是在 ZooKeeper 节点处理客户端请求之前，也可能是在 ZooKeeper 节点处理客户端请求之后。出现 ConnectionLossException 异常之后，客户端会进行自动重新连接，但是我们必须要检查我们以前的客户端请求是否被成功执行。

- InterruptedException：表示方法被中断了。我们可以使用 Thread.interrupt() 来中断 API 的执行。

数据读取 API 示例-getData

有以下三个获取 znode 数据的方法：

1.`byte[] getData(String path, boolean watch, Stat stat)`

同步方法。如果 `watch` 为 `true`，该 `znode` 的状态变化会发送给构建 `ZooKeeper` 是指定的 `watcher`。

2.`void getData(String path, boolean watch, DataCallback cb, Object ctx)`

异步方法。`cb` 是一个 `callback`，用来接收服务端的响应。`ctx` 是提供给 `cb` 的 `context`。`watch` 参数的含义和方法 `1` 相同。

3.`void getData(String path, Watcher watcher, DataCallback cb, Object ctx)`

异步方法。 `watcher` 用来接收该 `znode` 的状态变化。

数据写入 API 示例 -setData

```
1. Stat setData(String path, byte[] data, int version)
```

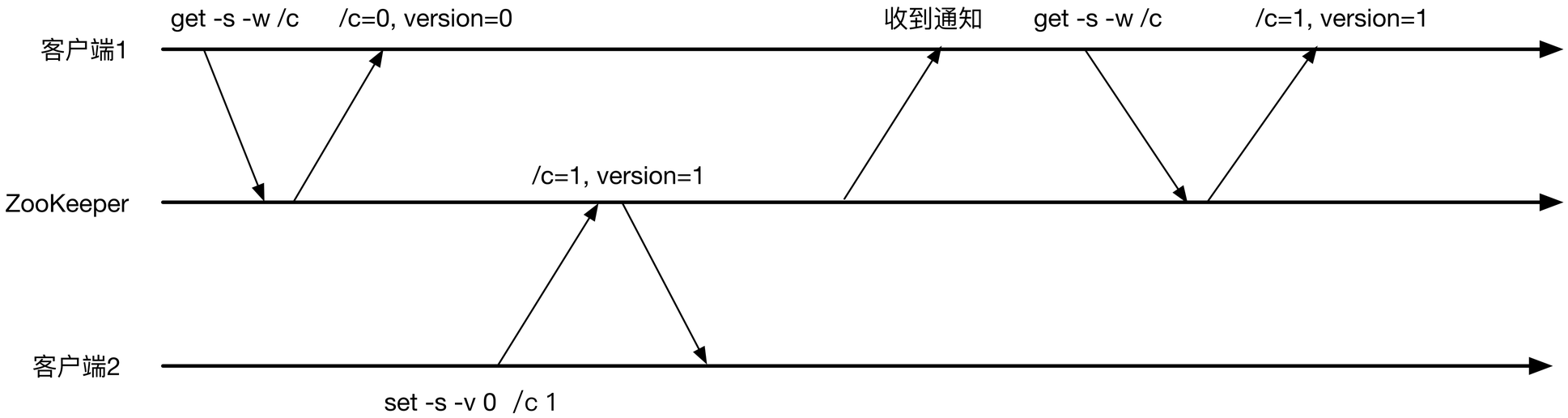
同步版本。如果 `version` 是 `-1`，做无条件更新。如果 `version` 是非负整数，做条件更新。

```
2. void setData(String path,
                byte[] data,
                int version,
                StatCallback cb,
                Object ctx)
```

异步版本。

watch

watch 提供一个让客户端获取最新数据的机制。如果没有 watch 机制，客户端需要不断的轮询 ZooKeeper 来查看是否有数据更新，这在分布式环境中是非常耗时的。客户端可以在读取数据的时候设置一个 watcher，这样在数据更新时，客户端就会收到通知。

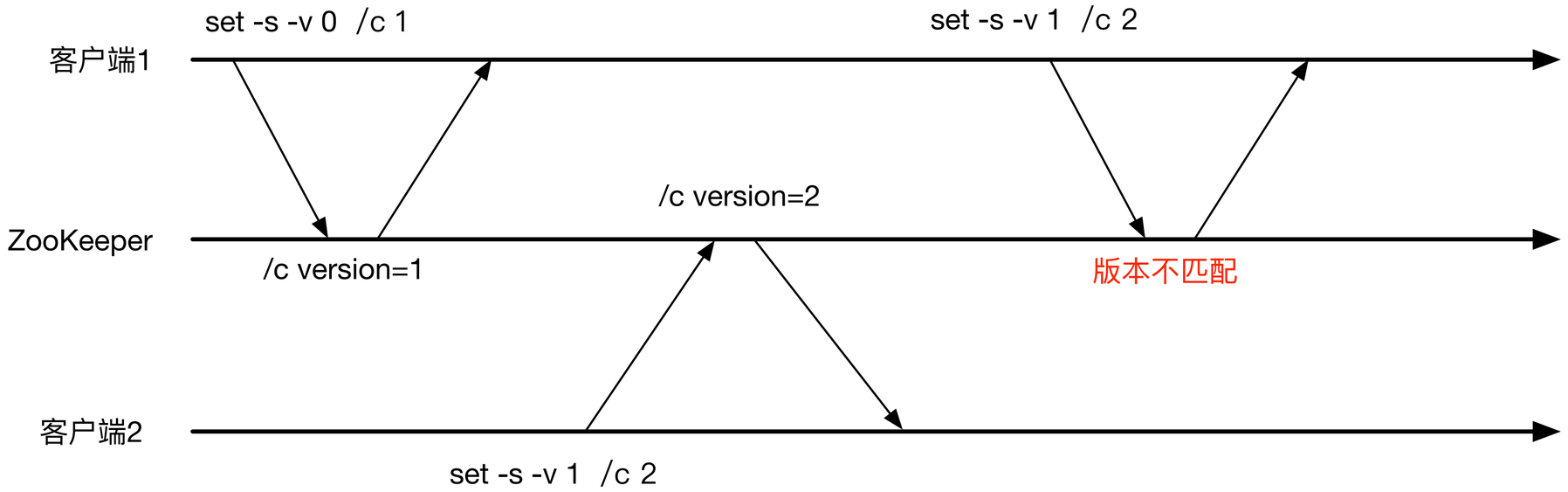


条件更新

设想用 znode /c 实现一个 counter，使用 set 命令来实现自增 1 操作。条件更新场景：

1. 客户端 1 把 /c 更新到版本 1，实现 /c 的自增 1 。
2. 客户端 2 把 /c 更新到版本 2，实现 /c 的自增 1 。
3. 客户端 1 不知道 /c 已经被客户端 2 更新过了，还用过时的版本 1 是去更新 /c，更新失败。如果客户端 1 使用的是无条件更新，/c 就会更新为 2，没有实现自增 1 。

使用条件更新可以避免对数据基于过期的数据进行数据更新操作。



Javadoc API 演示

ZooKeeper API-Watch 示例

设置 CLASSPATH

使用 ZooKeeper Java 代码需要使用依赖的Jar包，下面的命名把 ZooKeeper 依赖的 JAR 加到 CLASSPATH 环境变量：

```
ZOOBINDIR="<path_to_distro>/bin”
. "$ZOOBINDIR"/zkEnv.sh
```

但是因为我使用的是 ZSH，我不能使用上面的办法。我使用 scripts/executor.sh 来运行 ZooKeeper Java 代码，使用 Gradle 来编译 ZooKeeper Java 代码。

代码讲解

运行示例

ZooKeeper Recipes-分布式队列

ZooKeeper 源代码简介

如果你使用的是 macOS，使用以下命令获取 ZooKeeper 3.5.5 分支上的代码，并生成 Eclipse 项目文件。其他环境方法类似。

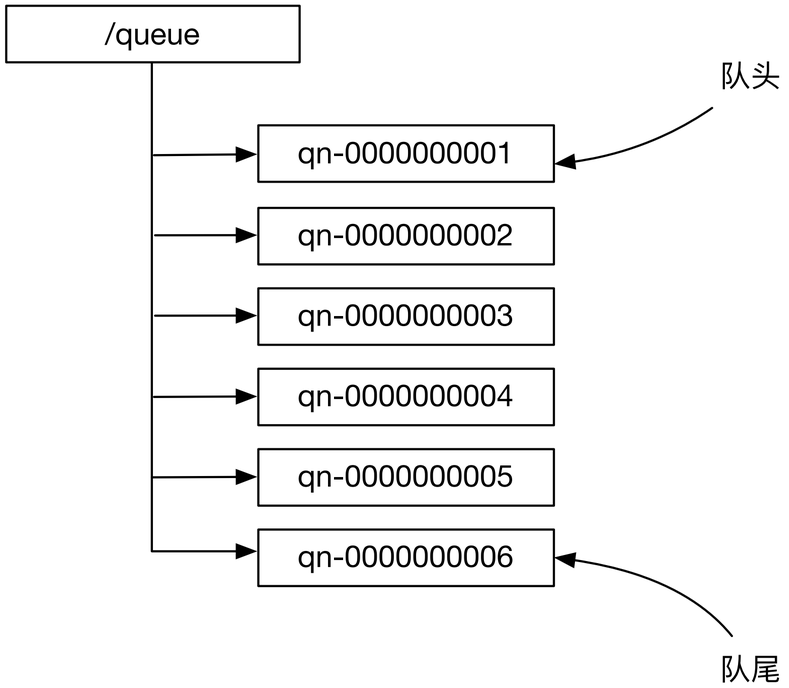
```
git clone https://github.com/apache/zookeeper.git

cd zookeeper

git checkout branch-3.5.5

brew install ant

ant eclipse
```

如果使用的是 Eclipse 环境，直接打开 ZooKeeper 项目就行。如果使用的是 Idea，使用 Idea 导入 Eclipse 项目的功能打开 ZooKeeper 项目。

ZooKeeper 的 build 工具是 Ant+Ivy。

设计

使用路径为 /queue 的 znode 下的节点表示队列中的元素。/queue 下的节点都是顺序持久化 znode。这些 znode 名字的后缀数字表示了对应队列元素在队列中的位置。Znode 名字后缀数字越小，对应队列元素在队列中的位置越靠前。Recipe 说明：Queues。



offer 方法

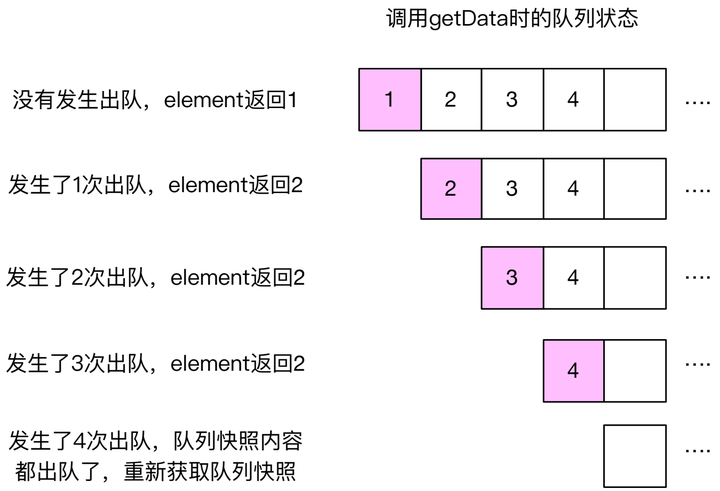
offer 方法在 /queue 下面创建一个顺序 znode。因为 znode 的后缀数字是/queue 下面现有 znode 最大后缀数字加 1，所以该 znode 对应的队列元素处于队尾。

element 方法

element 方法有以下两种返回的方式，我们下面说明这两种方式都是正确的。

1. throw new NoSuchElementException()：因为 element 方法读取到了队列为空的状态，所以抛出 NoSuchElementException 是正确的。

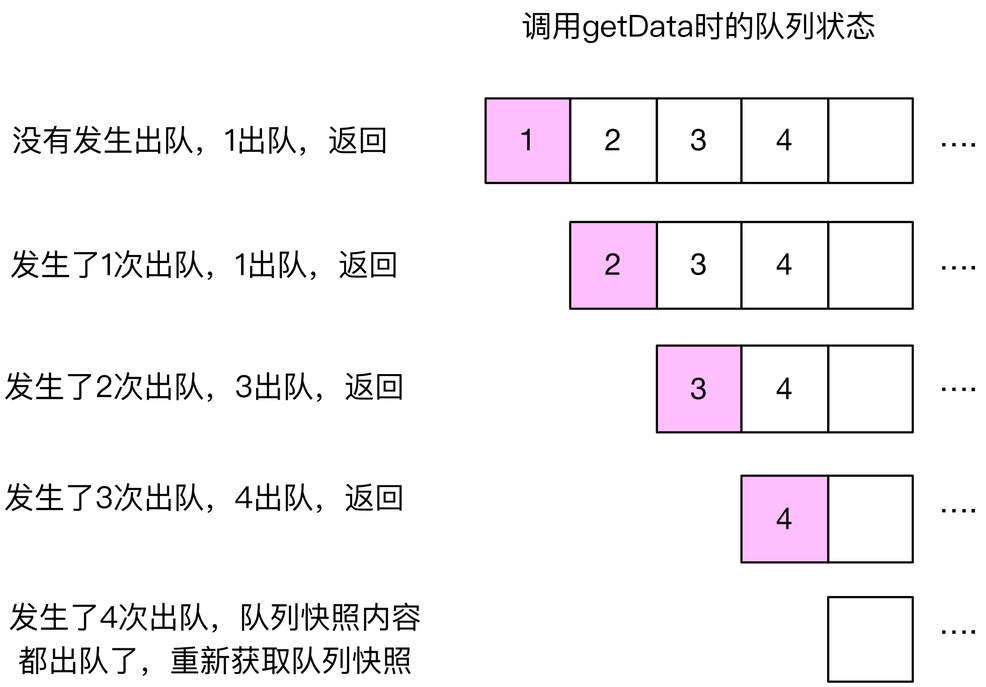
2. return zookeeper.getData(dir+"/"+headNode, false, null)：childNames 保存的是队列内容的一个快照。这个 return 语句返回快照中还没出队。如果队列快照的元素都出队了，重试。



take 方法

take 方法和 element 方法类似。take 只有一种返回方式。值得注意的是getData的成功执行不意味着出队成功，原因是该队列元素可能会被其他用户出队。

```
byte[] data = zookeeper.getData(path, false, null);

zookeeper.delete(path, -1);
```

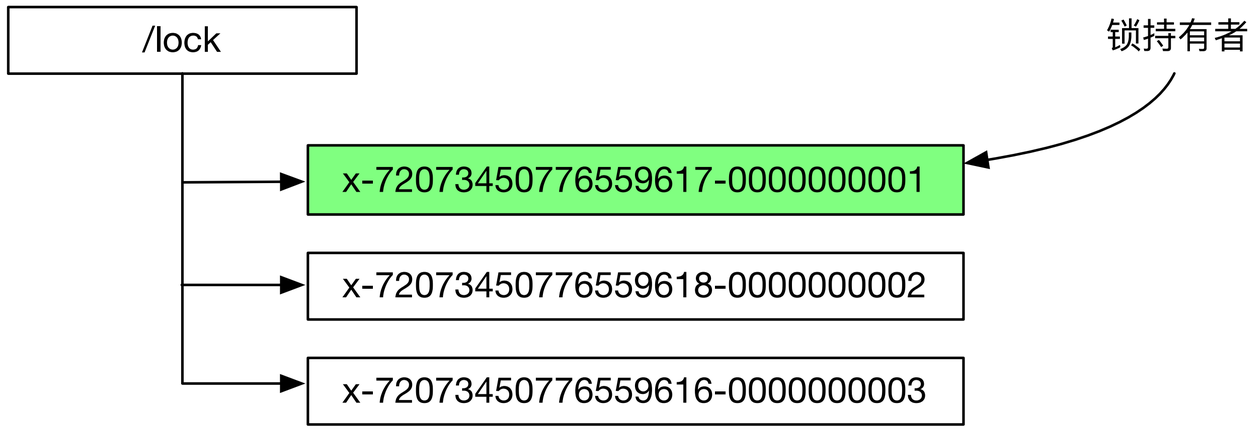


运行测试用例

ZooKeeper Recipes-分布式锁

设计

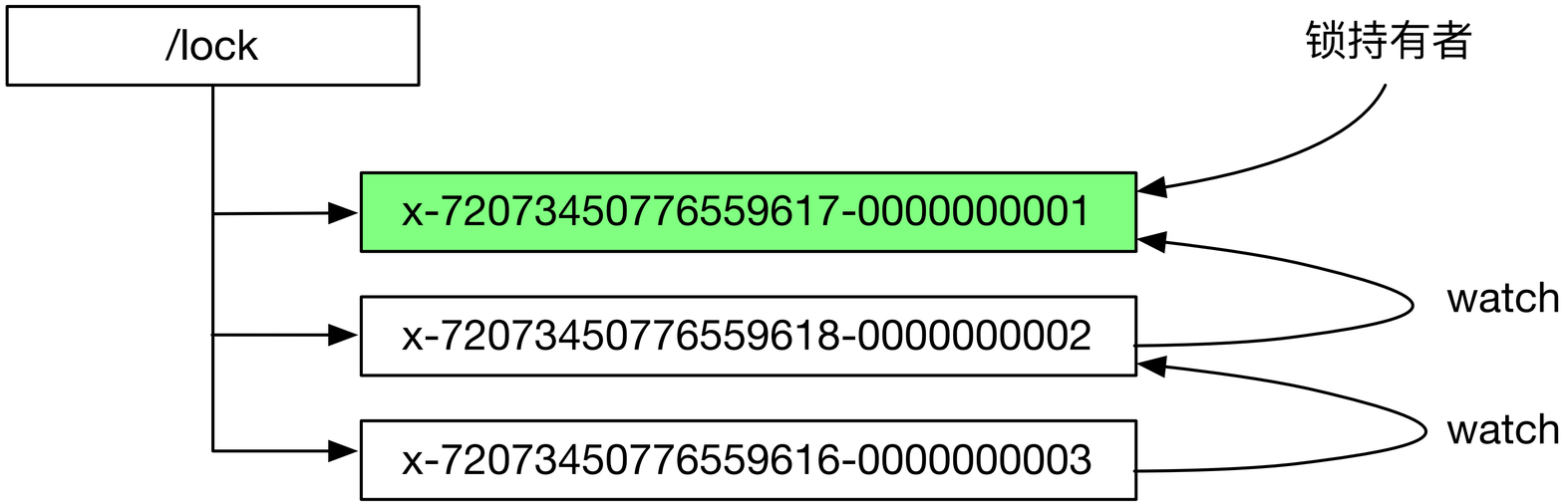
使用临时顺序 `znode` 来表示获取锁的请求，创建最小后缀数字 `znode` 的用户成功拿到锁。 `Recipe` 说明：[Locks](#)。



避免羊群效应（herd effect）

把锁请求者按照后缀数字进行排队，后缀数字小的锁请求者先获取锁。如果所有的锁请求者都 `watch` 锁持有者，当代表锁请求者的 `znode` 被删除以后，所有的锁请求者都会通知到，但是只有一个锁请求者能拿到锁。这就是羊群效应。

为了避免羊群效应，每个锁请求者 `watch` 它前面的锁请求者。每次锁被释放，只会有一个锁请求者会被通知到。这样做还让锁的分配具有公平性，锁定的分配遵循先到先得的原则。



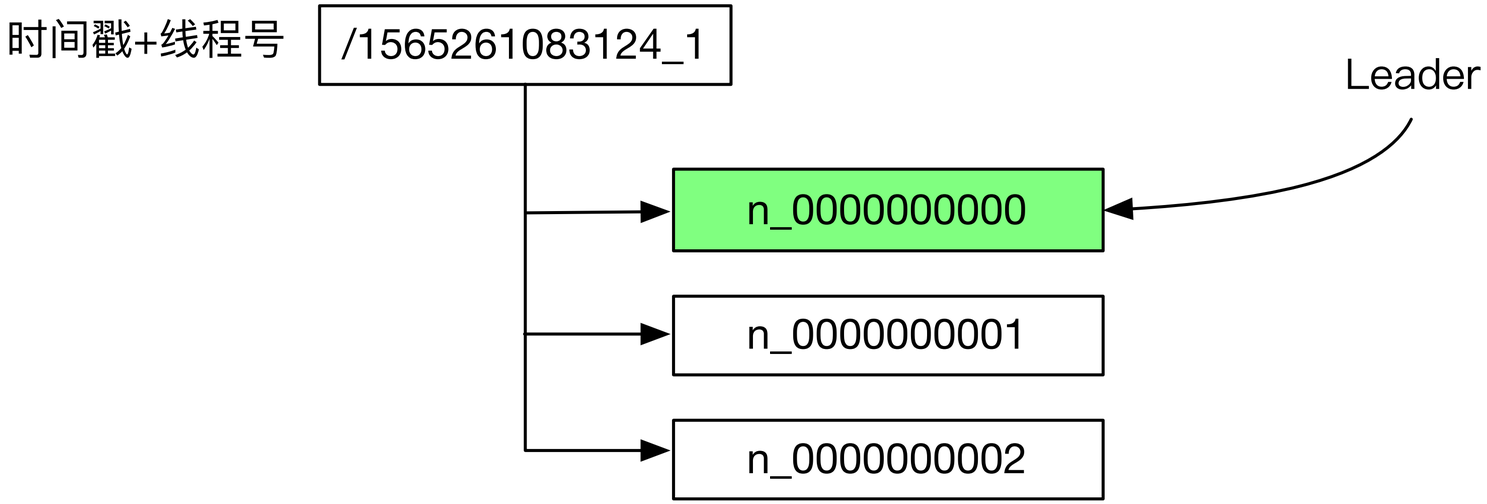
代码展示

运行测试用例

ZooKeeper Recipes-选举

设计

使用临时顺序 znode 来表示选举请求，创建最小后缀数字 znode 的选举请求成功。在协同设计上和分布式锁是一样的，不同之处在于具体实现。不同于分布式锁，选举的具体实现对选举的各个阶段做了监控。Recipe 说明：Leader Election 。



代码展示

运行测试用例

使用 Apache Curator 简化 ZooKeeper 开发

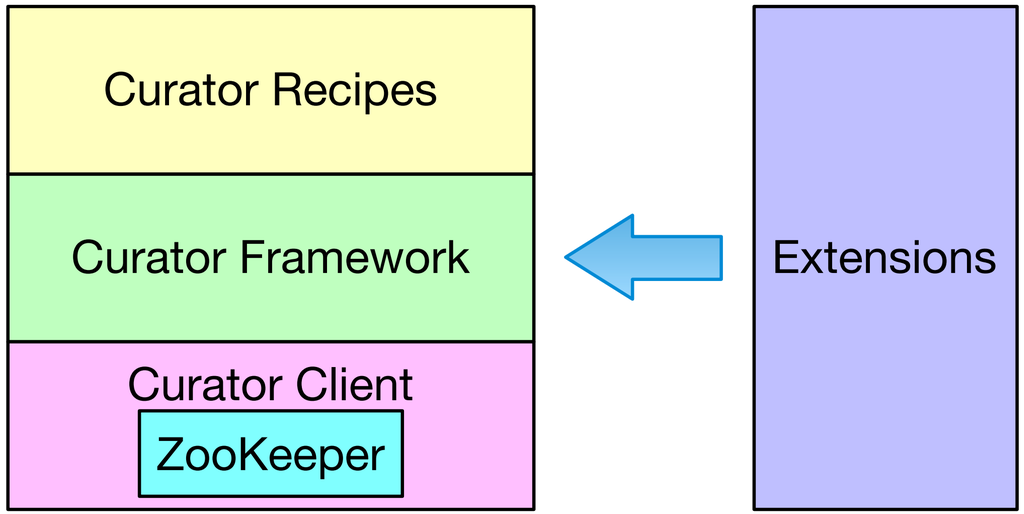
概述

Apache Curator 是 Apache ZooKeeper 的 Java 客户端库。Curator 项目的目标是简化 ZooKeeper 客户端的使用。例如，在以前的代码展示中，我们都要自己处理 ConnectionLossException 。另外 Curator 为常见的分布式协同服务提供了高质量的实现。

Apache Curator 最初是 Netflix 研发的，后来捐献给了 Apache 基金会，目前是 Apache 的顶级项目。

Curator 技术栈

- Client：封装了 ZooKeeper 类，管理和 ZooKeeper 集群的连接，并提供了重建连接机制。
- Framework：为所有的 ZooKeeper 操作提供了重试机制，对外提供了一个 Fluent 风格的 API 。
- Recipes：使用 framework 实现了大量的 ZooKeeper 协同服务。
- Extensions：扩展模块。



Client

初始化一个 client 分成两个步骤：1.创建 client 。2.启动 client 。以下是两种创建 client 的方法：

```
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
// 使用Factory方法
CuratorFramework zkc = CuratorFrameworkFactory.newClient(connectString, retryPolicy);

// Fluent风格
CuratorFramework zkc = CuratorFrameworkFactory.buidler()
                                              .connectString(connectString)
                                              .retryPolicy(retryPolicy)
                                              .build()
```

启动 client：

```
zkc.start();
```

Fluent 风格 API

```
// 同步版本
client.create().withMode(CreateMode.PERSISTENT).forPath(path, data);

// 异步版本
client.create().withMode(CreateMode.PERSISTENT).inBackground().forPath(path, data);

// 使用watch
client.getData().watched().forPath(path);
```

Curator 示例代码讲解

Curator Recipes 示例 - 选举

我们使用 Curator 的源代码来讲解，按照下面的步骤把 Curator 的源代码导入到 Idea：

1. 从 Curator 的主页 http://curator.apache.org/ 下载最新的版本，目前是 4.2.0。
2. 然后把 apache-curator-4.2.0-source-release.zip 解压到一个本地目录。
3. 然后用 Idea 的导入 Maven 项目的功能导入这个项目。

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程