**Curator**

Curator 是 Netflix 公司开源的一个 Zookeeper 客户端，与 Zookeeper 提供的原生客户端相比，Curator 的抽象层次更高，简化了 Zookeeper 客户端的开发量。现在已是 apache 的顶级开源框架，Fluent 编程风格的实现。

官网[:http://cu](http://curator.apache.org/)r[ator.apache.org](http://curator.apache.org/)

Maven 依赖基础框架

<dependency>

<groupId>org.apache.curator</groupId> 基础框架

<artifactId>curator-framework</artifactId>

<version></version>

</dependency>

<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-recipes</artifactId> 功能 jar 分布式锁、队列等

<version></version>

</dependency>

<dependency> 客户端重试策略

<groupId>org.apache.curator</groupId>

<artifactId>curator-client</artifactId>

<version></version>

</dependency>

官网：<http://curator.apache.org/>

**使用:**

Curator 框架提供了一种流式接口,通过 builder 串起来，传递参数都是调方法。

Curator 框架通过 CuratorFrameworkFactory 以工厂模式和 builder 模式创建CuratorFramework 实 例。 CuratorFramework 实例都是线程安全的，你应该在你的应用中共享同一个.

工厂方法 newClient()提供了一个简单方式创建实例。 而 Builder 提供了更多的参数控制。一旦你创建了一个 CuratorFramework 实例，你必须调用它的 start()启动，在应用退出时调用 close()方法关闭.

1.创建 Curator 连接实例

ExponentialBackoffRetry(1000, 3));//重试机制

client.start();

new

CuratorFrameworkFactory.newClient(address,

=

client

CuratorFramework

String address = "localhost:2181";

注意：一个 Zookeeper 集群只需要构造一个 CuratorFramework 实例对象即可。

CuratorFramework 使用之前必须先调用

client.start();

**CuratorFramework 提供的方法：**

|  |  |
| --- | --- |
| **方法名** | **描述** |
| create | 开始创建操作， 可以调用额外的方法(比如方式 mode 或者后台执行  background) 并在最后调用 forPath()指定要操作的 ZNode |

|  |  |
| --- | --- |
| Delete | 开始删除操作. 可以调用额外的方法(版本或者后台处理 version or background)并在最后调用 forPath()指定要操作的 ZNode |
| checkExists | 开始检查 ZNode 是否存在的操作. 可以调用额外的方法(监控或者后台处理)并在最后调用 forPath()指定要操作的 ZNode |
| getData | 开始获得 ZNode 节点数据的操作. 可以调用额外的方法(监控、后台处理或者获取状态 watch, background or get stat) 并在最后调用 forPath()指定要操作的 ZNode |
| setData | 开始设置 ZNode 节点数据的操作. 可以调用额外的方法(版本或者后台处理) 并在最后调用 forPath()指定要操作的 ZNode |
| getChildren | 开始获得 ZNode 的子节点列表。 以调用额外的方法(监控、后台处理或者获取状态 watch, background or get stat) 并在最后调用 forPath()指定要操作的 ZNode |
| inTransaction | 开始是原子 ZooKeeper 事务. 可以复合 create, setData, check, and/or delete 等操作然后调用 commit()作为一个原子操作提交 |

**事件类型以及事件的方法如下：**

|  |  |
| --- | --- |
| **Event Type** | **Event Methods** |
| CREATE | getResultCode() and getPath() |
| DELETE | getResultCode() and getPath() |
| EXISTS | getResultCode(), getPath() and getStat() |
| GETDATA | getResultCode(), getPath(), getStat() and getData() |
| SETDATA | getResultCode(), getPath() and getStat() |
| CHILDREN | getResultCode(), getPath(), getStat(), getChildren() |
| WATCHED | getWatchedEvent |

**监听器**

Curator 提供了三种 Watcher(Cache)来监听结点的变化：

**Path Cache**：监视一个路径下子结点的创建、删除，以及结点数据的更新。产生的

事件会传递给注册的 PathChildrenCacheListener。

**Node Cache**：监视一个结点的创建、更新、删除，并将结点的数据缓存在本地。

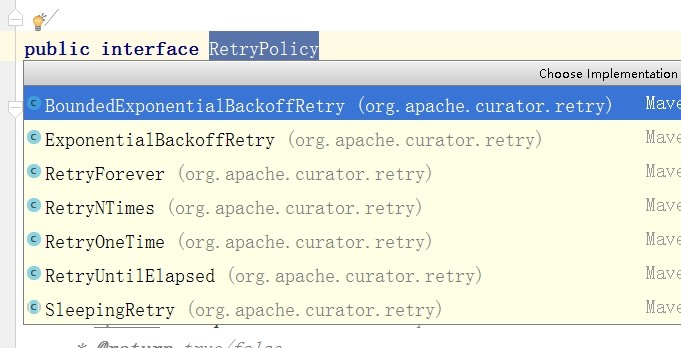
**Tree Cache**：Path Cache 和 Node Cache 的“合体”，监视路径下的创建、更新、删除事件，并缓存路径下所有子结点的数据。

# 重试机制

Curator 内部实现的几种重试策略:

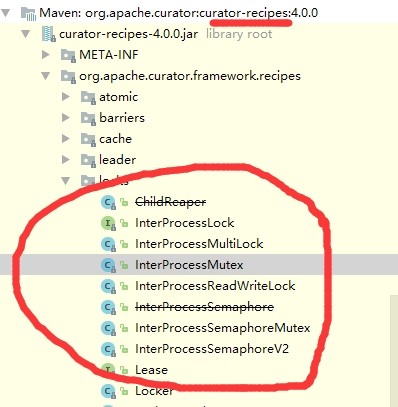
1.ExponentialBackoffRetry:重试指定的次数, 且每一次重试之间停顿的时间逐渐增加. 2.RetryNTimes:指定最大重试次数的重试策略

1. RetryOneTime:仅重试一次
2. RetryUntilElapsed:一直重试直到达到规定的时间



# 分布式锁

用分布式锁或者原子操作、队列等功能需引入



<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-recipes</artifactId>

<version>${version}</version>

</dependency>