JiangYue
2018年05月03日 阅读 254

ZooKeeper 使用 Java API

zkCli 工具适用于调试，不推荐使用 zkCli 工具来搭建系统。

实际开发时一般也不直接使用 ZooKeeper 的 Java API，而是使用更高层次的封装库 Curator，不过对 Java API 的学习仍然有很多益处。

本篇文章介绍通过 ZooKeeper 的 Java API 来实现创建会话、实现监视点等功能，演示主从模式。

添加依赖

```xml
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.9</version>
</dependency>
```

建立会话

启动 ZooKeeper 服务端，通过 Java API 建立会话。

```java
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)
```

其中 connectString 包含主机名及端口号，sessionTimeout 为会话超时时间，watcher 对象用于接收会话事件。

Watcher 为一个接口，实现 Watcher 需要重写 `void process(WatchedEvent event)` 方法。

当遇到网络故障时，如果连接断开，ZooKeeper 客户端会自动重新连接。

活动状态。

java

```
package com.ulyssesss.zookeeper;

import org.apache.zookeeper.*;
import org.apache.zookeeper.data.Stat;
import java.io.IOException;
import java.util.Random;

public class Master implements Watcher {

    private ZooKeeper zk;
    private String serviceId = Integer.toString(new Random().nextInt());
    private boolean isLeader = false;

    private void startZk() throws IOException {
        zk = new ZooKeeper("localhost:2181", 5000, this);
    }

    private void stopZk() throws InterruptedException {
        zk.close();
    }

    public void process(WatchedEvent watchedEvent) {
        System.out.println("event: " + watchedEvent);
    }

    public static void main(String[] args) throws Exception {
                Master master = new Master();
                master.startZk();

                master.runForMaster();

        System.out.println("serviceId: " + master.serviceId);

                if (master.isLeader) {
            System.out.println("master");
            Thread.sleep(10000);
        } else {
            System.out.println("not master");
        }

                master.stopZk();
    }
```

```
                byte data[] = zk.getData("/master", false, stat);
                isLeader = new String(data).equals(serviceId);
                return true;
            } catch (KeeperException.NoNodeException e) {
                return false;
            } catch (KeeperException e) {
                e.printStackTrace();
            }
        }
    }

    private void runForMaster() throws InterruptedException {
        while (true) {
            try {
                zk.create("/master", serviceId.getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.
                isLeader = true;
                break;
            } catch (KeeperException.NodeExistsException e) {
                isLeader = false;
                break;
            } catch (KeeperException e) {
                e.printStackTrace();
            }
            if (checkMaster()) {
                break;
            }
        }
    }
}
```

主函数执行创建一个演示实例，实例会分配一个随机整数作为 id，建立 ZooKeeper 连接后尝试创建主节点 master。

如果 master 主节点创建成功，则该实例为群首 leader；如果节点已经存在则其他实例为 leader；发生断开连接等异常时，响应信息丢失，无法确定当前进程是否为主节点，需要通过 checkMaster() 方法重新检查主节点状态。

多次执行主函数，其中第一次执行时会打印 master，在 master 断开连接前的 10 秒钟内，再次执行会打印 not master，当第一次执行的 master 断开连接后，再次执行主函数，打印 master。

主从模型的设计需要用到 /tasks、/assign 和 /workers 3 个目录，可以通过某些系统配置来创建这些目录。下面的代码示例会通过异步的方式创建出所需要的目录。

```java
private void bootstrap() {
    createParent("/workers", new byte[0]);
    createParent("/assign", new byte[0]);
    createParent("/tasks", new byte[0]);
}

private void createParent(String path, byte[] data) {
    zk.create(path, data, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT, createParentCallback,
}

AsyncCallback.StringCallback createParentCallback = new AsyncCallback.StringCallback() {
    public void processResult(int rc, String path, Object ctx, String name) {
        switch (KeeperException.Code.get(rc)) {
            case OK:
                System.out.println("parent " + path + " created");
                break;
            case NODEEXISTS:
                System.out.println("parent " + path + " already registered");
                break;
            case CONNECTIONLOSS:
                createParent(path, (byte[]) ctx);
                break;
            default:
                System.out.println("create " + path + " error");
        }
    }
};
```

注册从节点

前面的部分已经有了主节点，为了使主节点可以发号施令，现在要配置从节点，在 /workers 下创建临时节点。

```java
package com.ulyssesss.zookeeper;
```

```
private ZooKeeper zk;
private String serviceId = Integer.toString(new Random().nextInt());

private void startZk() throws IOException {
    zk = new ZooKeeper("localhost:2181", 5000, this);
}

@Override
public void process(WatchedEvent event) {
    System.out.println("event: " + event);
}

private void register() {
    zk.create("/workers/worker-" + serviceId, "Idle".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE
            , CreateMode.EPHEMERAL, createWorkerCallback, null);
}

private AsyncCallback.StringCallback createWorkerCallback = new AsyncCallback.StringCallback()
    @Override
    public void processResult(int rc, String path, Object ctx, String name) {
        switch (KeeperException.Code.get(rc)) {
            case OK:
                System.out.println("registered successfully: " + serviceId);
                break;
            case NODEEXISTS:
                System.out.println("already registered: " + serviceId);
                break;
            case CONNECTIONLOSS:
                register();
                break;
            default:
                System.out.println("error");
        }
    }
};

private String status;

public void setStatus(String status) {
    this.status = status;
    updateStatus(status);
}
```

```
    AsyncCallback.StatCallback statusUpdateCallback = new AsyncCallback.StatCallback() {
        @Override
        public void processResult(int rc, String path, Object ctx, Stat stat) {
            switch (KeeperException.Code.get(rc)) {
                case CONNECTIONLOSS:
                    updateStatus((String) ctx);
                    break;
                default:
            }
        }
    };

    public static void main(String[] args) throws Exception {
                Worker worker = new Worker();
                worker.startZk();
                worker.register();
                Thread.sleep(30000);
    }
}
```

主函数创建 worker 实例，开启会话，执行注册逻辑，创建节点时如发生连接丢失则再次执行注册逻辑，注册所创建的节点为临时节点。

从节点开始处理某些任务时，需要通过 setStatus() 方法更新节点状态。

任务队列

系统中 client 组件用于添加任务，以便从节点执行任务。以下为 client 代码：

java

```
package com.ulyssesss.zookeeper;

import org.apache.zookeeper.*;
import java.io.IOException;

public class Client implements Watcher {

    private ZooKeeper zk;
```

```
    public void process(WatchedEvent event) {
        System.out.println("event: " + event);
    }

    private String queueCommand(String command) {
        while (true) {
            try {
                String name = zk.create("/tasks/task-", command.getBytes()
                        , ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT_SEQUENTIAL);
                return name;
            } catch (Exception e) {
                System.out.println("error");
            }
        }
    }


    public static void main(String[] args) throws Exception {
                Client client = new Client();
                client.startZk();
                String name = client.queueCommand("command-1");
        System.out.println("created " + name);
    }
}
```

Client 使用有序节点 task- 标示任务，task- 后面会跟随一个递增整数，在执行 create 时如发生连接丢失，则重试 create 操作，适用于【至少执行一次】策略的应用。如要采用【至多执行一次】策略，可以将任务的唯一标识添加到节点名中。

管理客户端

管理客户端 AdminClient 用于展示系统运行状态，代码如下：

java

```
package com.ulyssesss.zookeeper;

import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.data.Stat;
```

```
    private ZooKeeper zk;

    private void startZk() throws IOException {
        zk = new ZooKeeper("localhost:2181", 5000, this);
    }

    @Override
    public void process(WatchedEvent event) {
        System.out.println("event: " + event);
    }

    private void listState() throws KeeperException, InterruptedException {
        try {
            Stat stat = new Stat();
            byte[] masterData = zk.getData("/master", false, stat);
            Date startDate = new Date(stat.getCtime());
            System.out.println("master: " + new String(masterData) + " since " + startDate);
        } catch (KeeperException.NoNodeException e) {
            System.out.println("no master");
        }

        System.out.println("workers: \n");
        for (String worker : zk.getChildren("/workers", false)) {
            byte[] data = zk.getData("/workers/" + worker, false, null);
            String state = new String(data);
            System.out.println("worker: " + state);
        }

        // ...
    }

    public static void main(String[] args) throws Exception {
                AdminClient adminClient = new AdminClient();
                adminClient.startZk();
                adminClient.listState();
    }
}
```

以上代码会简单的列出各个节点的信息。

通过 Java API 编程与 zkCli 命令非常接近，不同的是 zkCli 常用于调试，一般会在一个相对稳定的环境下使用。通过 Java API 编写的程序，需要考虑到异常情况，尤其是 ConnectionLossExceptio... 常，需要检查状态并合理恢复。

关注下面的标签，发现更多相似文章

后端 Java 算法 ZooKeeper

评论

登录 说说你的看法

相关推荐

Java公众号_芋道源码_每日更新 · 12小时前 · Java
分布式消息队列 RocketMQ 源码分析 —— Message 发送与接收
30

热 · 专栏 · 若邪 · 1天前 · Vue.js
vue权限路由实现方式总结
274 6

专栏 · ZedeChan · 5小时前 · Android / 算法
LruCache 使用及原理
12 1

专栏 · 超人汪小建 · 10小时前 · 算法 / 后端
看图轻松理解数据结构与算法系列(双向链表)
42 1

热 · 专栏 · 公众号_咖啡拿铁 · 2天前 · 后端
再有人问你分布式事务，把这篇扔给他

131 3

Java公众号_芋道源码_每日更新 · 1天前 · Java

配置中心 Apollo 源码解析 —— 客户端配置 Spring 集成（三）之外部化配置

30

热 · 专栏 · L_Xian · 2天前 · Android

又有MVP新写法了，这次我认为挺不错的。

105 26

专栏 · ztwindy · 1天前 · MySQL / 后端

MySQL探索(一):B-Tree索引

56

热 · 专栏 · 夕阳 · 6天前 · JavaScript

面试图谱：前端基础技术知识讲解

853 18