


价格参考 (<http://www.fwqtg.net/%e4%bb%b7%e6%a0%bc%e5%8f%82%e8%80%83>)

数据中心 (<http://www.fwqtg.net/our-portfolio>) | 联系我们 (<http://www.fwqtg.net/contact-us>)

QQ咨询 (<http://wpa.qq.com/msgrd?v=3&uin=93663045&site=qq&menu=yes>)

 (<http://www.fwqtg.net>)

ZooKeeper示例 实时更新server列表

ZooKeeper示例 实时更新server列表

 分享         

☆ 特别推荐: 希望有缘来到小站的用户们, ^{在线咨询}如果平时需要服务器主机托管, 云服务, 机房机柜带宽租用等, 可以交个朋友, 我叫董礼 QQ 93663045 真诚交友, 用心服务, 价格最低!

转自: <http://coolxing.iteye.com/blog/1871520>

通过之前的3篇博文, 讲述了ZooKeeper的基础知识点. 可以看出, ZooKeeper提供的核心功能是非常简单, 且易于学习的. 可能会给人留下ZooKeeper并不强大的印象, 事实并非如此, 基于ZooKeeper的核心功能, 我们可以扩展出很多非常有意思的应用. 接下来的几篇博文, 将陆续介绍ZooKeeper的典型应用场景.

场景描述

在分布式应用中, 我们经常同时启动多个server, 调用方(client)选择其中之一发起请求. 分布式应用必须考虑高可用性和可扩展性: server的应用进程可能会崩溃, 或者server本身也可能会宕机. 当server不够时, 也有可能增加server的数量. 总而言之, server列表并非一成不变, 而是一直处于动态的增减中.

那么client如何才能实时的更新server列表呢? 解决的方案很多, 本文将讲述利用ZooKeeper的解决方案.


思路

启动server时, 在zookeeper的某个znode(假设为/sgroup)下创建一个子节点. 所创建的子节点的类型应该为ephemeral, 这样一来, 如果server进程崩溃, 或者server宕机, 与zookeeper连接的session就结束了, 那么其所创建的子节点会被zookeeper自动删除. 当崩溃的server恢复后, 或者新增server时, 同样需要在/sgroup节点下创建新的子节点.

对于client, 只需注册/sgroup子节点的监听, 当/sgroup下的子节点增加或减少时, zookeeper会通知client, 此时client更新server列表.

实现AppServer

AppServer的逻辑非常简单, 只须在启动时, 在zookeeper的"/sgroup"节点下新增一个子节点即可.

Java代码 

```
public class AppServer {
    private String groupNode = "sgroup";
    private String subNode = "sub";

    /**
     * 连接zookeeper
     * @param address server的地址
     */
    public void connectZookeeper(String address) throws Exception {
        ZooKeeper zk = new ZooKeeper("localhost:4180,localhost:4181,localhost:4182");
        public void process(WatchedEvent event) {
            // 不做处理
        }
    };
    // 在"/sgroup"下创建子节点
    // 子节点的类型设置为EPHEMERAL_SEQUENTIAL, 表明这是一个临时节点, 且在子节点的名称
    上一串数字后缀
    // 将server的地址数据关联到新创建的子节点上
    String createdPath = zk.create("/") + groupNode + "/" + subNode, address.g
    8"),
        Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);
    System.out.println("create: " + createdPath);
}

/**
 * server的工作逻辑写在这个方法中
 * 此处不做任何处理, 只让server sleep
 */
public void handle() throws InterruptedException {
    Thread.sleep(Long.MAX_VALUE);
}

public static void main(String[] args) throws Exception {
    // 在参数中指定server的地址
    if (args.length == 0) {
        System.err.println("The first argument must be server address");
        System.exit(1);
    }
}
```

```

        AppServer as = new AppServer();
        as.connectZookeeper(args[0]);

        as.handle();
    }
}


```

将其打成appserver.jar后待用, 生成jar时别忘了指定入口函数. 具体的教程请自行搜索.

实现AppClient

AppClient的逻辑比AppServer稍微复杂一些, 需要监听”/sgroup”下子节点的变化事件, 当事件发生时, 需要更新server列表.

注册监听”/sgroup”下子节点的变化事件, 可在getChildren方法中完成. 当zookeeper回调监听器的process方法时, 判断该事件是否是”/sgroup”下子节点的变化事件, 如果是, 则调用更新逻辑, 并再次注册该事件的监听.

Java代码 

```

public class AppClient {
    private String groupNode = "sgroup";
    private ZooKeeper zk;
    private Stat stat = new Stat();
    private volatile List<String> serverList;

    /**
     * 连接zookeeper
     */
    public void connectZookeeper() throws Exception {
        zk = new ZooKeeper("localhost:4180,localhost:4181,localhost:4182", 5000,
            public void process(WatchedEvent event) {
                // 如果发生了”/sgroup”节点下的子节点变化事件, 更新server列表, 并重新注册!
                if (event.getType() == EventType.NodeChildrenChanged
                    && ("/" + groupNode).equals(event.getPath())) {
                    try {
                        updateServerList();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        });

        updateServerList();
    }

    /**

```

```

* 更新server列表
*/
private void updateServerList() throws Exception {
    List<String> newServerList = new ArrayList<String>();

    // 获取并监听groupNode的子节点变化
    // watch参数为true, 表示监听子节点变化事件.
    // 每次都需要重新注册监听, 因为一次注册, 只能监听一次事件, 如果还想继续保持监听, 必须
    注册
    List<String> subList = zk.getChildren("/") + groupNode, true);
    for (String subNode : subList) {
        // 获取每个子节点下关联的server地址
        byte[] data = zk.getData("/") + groupNode + "/" + subNode, false, stat
        newServerList.add(new String(data, "utf-8"));
    }

    // 替换server列表
    serverList = newServerList;

    System.out.println("server list updated: " + serverList);
}

/**
 * client的工作逻辑写在这个方法中
 * 此处不做任何处理, 只让client sleep
 */
public void handle() throws InterruptedException {
    Thread.sleep(Long.MAX_VALUE);
}

public static void main(String[] args) throws Exception {
    AppClient ac = new AppClient();
    ac.connectZookeeper();

    ac.handle();
}
}

```

将其打包成appclient.jar后待用, 别忘了指定入口函数.

运行


在运行jar包之前, 需要确认zookeeper中是否已经存在"/sgroup"节点了, 没有不存在, 则创建该节点. 如果存在, 最好先将其删除, 然后再重新创建. ZooKeeper的相关命令可参考我的另一篇博文.

运行appclient.jar: `java -jar appclient.jar` 开启多个命令行窗口, 每个窗口运行appserver.jar 进程: `java -jar appserver.jar server0000`. "server0000"表示server的地址, 别忘了给每个

server设定一个不同的地址. 观察appclient的输出.

依次结束appserver的进程, 观察appclient的输出.

appclient的输出类似于:

Bash代码 

```
server list updated: []
server list updated: [server0000]
server list updated: [server0000, server0001]
server list updated: [server0000, server0001, server0002]
server list updated: [server0000, server0001, server0002, server0003]
server list updated: [server0000, server0001, server0002]
server list updated: [server0000, server0001]
server list updated: [server0000]
server list updated: []
```

<iframe style=" font-size: 12px; line-height: 18px;"
src=" http://coolxing.iteye.com/iframe_ggbd/794" frameborder=" 0" scrolling=" no"
width=" 468" height=" 60" ></iframe>

通过之前的3篇博文, 讲述了ZooKeeper的基础知识点. 可以看出, ZooKeeper提供的核心功能是非常简单, 且易于学习的. 可能会给人留下ZooKeeper并不强大的印象, 事实并非如此, 基于ZooKeeper的核心功能, 我们可以扩展出很多非常有意思的应用. 接下来的几篇博文, 将陆续介绍ZooKeeper的典型应用场景.

场景描述

在分布式应用中, 我们经常同时启动多个server, 调用方(client)选择其中之一发起请求.

分布式应用必须考虑高可用性和可扩展性: server的应用进程可能会崩溃, 或者server本身可能会宕机. 当server不够时, 也有可能增加server的数量. 总而言之, server列表并非一成不变, 而是一直处于动态的增减中.

那么client如何才能实时的更新server列表呢? 解决的方案很多, 本文将讲述利用ZooKeeper的解决方案.

思路

启动server时, 在zookeeper的某个znode(假设为/sgroup)下创建一个子节点. 所创建的子节点的类型应该为ephemeral, 这样一来, 如果server进程崩溃, 或者server宕机, 与zookeeper连接的session就结束了, 那么其所创建的子节点会被zookeeper自动删除. 当崩溃的server恢复后, 或者新增server时, 同样需要在/sgroup节点下创建新的子节点.

对于client, 只需注册/sgroup子节点的监听, 当/sgroup下的子节点增加或减少时, zookeeper会通知client, 此时client更新server列表.

实现AppServer

AppServer的逻辑非常简单, 只须在启动时, 在zookeeper的"/sgroup"节点下新增一个子节点即可.

Java代码 ☆

```
public class AppServer {
    private String groupNode = "sgroup";
    private String subNode = "sub";

    /**
     * 连接zookeeper
     * @param address server的地址
     */
    public void connectZookeeper(String address) throws Exception {
        ZooKeeper zk = new ZooKeeper("localhost:4180,localhost:4181,localhost:4182");
        public void process(WatchedEvent event) {
            // 不做处理
        }
    });
    // 在"/sgroup"下创建子节点
    // 子节点的类型设置为EPHEMERAL_SEQUENTIAL, 表明这是一个临时节点, 且在子节点的名称
    上一串数字后缀
    // 将server的地址数据关联到新创建的子节点上
    String createdPath = zk.create("/") + groupNode + "/" + subNode, address.g
    8"),
        Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);
    System.out.println("create: " + createdPath);
}

/**
 * server的工作逻辑写在这个方法中
 * 此处不做任何处理, 只让server sleep
 */
public void handle() throws InterruptedException {
    Thread.sleep(Long.MAX_VALUE);
}

public static void main(String[] args) throws Exception {
    // 在参数中指定server的地址
    if (args.length == 0) {
        System.err.println("The first argument must be server address");
        System.exit(1);
    }

    AppServer as = new AppServer();
    as.connectZookeeper(args[0]);


    as.handle();
}
}
```

将其打成appserver.jar后待用, 生成jar时别忘了指定入口函数. 具体的教程请自行搜索.

实现AppClient

AppClient的逻辑比AppServer稍微复杂一些, 需要监听”/sgroup”下子节点的变化事件, 当事件发生时, 需要更新server列表.

注册监听”/sgroup”下子节点的变化事件, 可在getChildren方法中完成. 当zookeeper回调监听器的process方法时, 判断该事件是否是”/sgroup”下子节点的变化事件, 如果是, 则调用更新逻辑, 并再次注册该事件的监听.

Java代码 

```
public class AppClient {
    private String groupNode = "sgroup";
    private ZooKeeper zk;
    private Stat stat = new Stat();
    private volatile List<String> serverList;

    /**
     * 连接zookeeper
     */
    public void connectZookeeper() throws Exception {
        zk = new ZooKeeper("localhost:4180,localhost:4181,localhost:4182", 5000,
            new Watcher() {
                public void process(WatchedEvent event) {
                    // 如果发生了”/sgroup”节点下的子节点变化事件, 更新server列表, 并重新注册!
                    if (event.getType() == EventType.NodeChildrenChanged
                        && ("/" + groupNode).equals(event.getPath())) {
                        try {
                            updateServerList();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        );
        updateServerList();
    }

    /**
     * 更新server列表
     */
    private void updateServerList() throws Exception {
        List<String> newServerList = new ArrayList<String>();

        // 获取并监听groupNode的子节点变化
        // watch参数为true, 表示监听子节点变化事件.
        // 每次都需要重新注册监听, 因为一次注册, 只能监听一次事件, 如果还想继续保持监听, 必须
        // 注册
        List<String> subList = zk.getChildren("/", groupNode, true);
        for (String subNode : subList) {
```

```

// 获取每个子节点下关联的server地址
byte[] data = zk.getData("/") + groupNode + "/" + subNode, false, stat
newServerList.add(new String(data, "utf-8"));
}

// 替换server列表
serverList = newServerList;

System.out.println("server list updated: " + serverList);
}

/**
 * client的工作逻辑写在这个方法中
 * 此处不做任何处理, 只让client sleep
 */
public void handle() throws InterruptedException {
    Thread.sleep(Long.MAX_VALUE);
}

public static void main(String[] args) throws Exception {
    AppClient ac = new AppClient();
    ac.connectZookeeper();

    ac.handle();
}
}

```

将其打包成appclient.jar后待用, 别忘了指定入口函数.

运行

在运行jar包之前, 需要确认zookeeper中是否已经存在"/sgroup"节点了, 没有不存在, 则创建该节点. 如果存在, 最好先将其删除, 然后再重新创建. ZooKeeper的相关命令可参考我的另一篇博文.

运行appclient.jar: `java -jar appclient.jar` 开启多个命令行窗口, 每个窗口运行appserver.jar 进程: `java -jar appserver.jar server0000`. "server0000"表示server的地址, 别忘了给每个

server设定一个不同的地址. 观察appclient的输出.

依次结束appserver的进程, 观察appclient的输出.

appclient的输出类似于:

Bash代码 ☆


```

server list updated: []
server list updated: [server0000]
server list updated: [server0000, server0001]
server list updated: [server0000, server0001, server0002]
server list updated: [server0000, server0001, server0002, server0003]
server list updated: [server0000, server0001, server0002]

```



```
server list updated: [server0000, server0001]
server list updated: [server0000]
server list updated: []
```

 系统运维 (<http://www.fwqtg.net/category/zxgx/yunwei>)

发表回复

电子邮件地址不会被公开。 必填项已用*标注

名字 *

邮箱 *

网址

评论 *

文章评论

关于我们

北京中联互动科技发展有限公司（以下简称“中联互动”）成立于2004年，是一家专业的互联网基础建设服务商。其核心团队成员拥有多年互联网专业经验和先进的服务理念，为客户提供优质互联网基础服务。

友情链接

[北京服务器托管 \(http://www.fwqtg.net\)](http://www.fwqtg.net) [北京兆维机房 \(http://www.e-1.cn\)](http://www.e-1.cn)

最新资讯

[Android]自定义控件LoadMoreRecyclerView
(<http://www.fwqtg.net/android%E8%87%AA%E5%AE%9A%E4%B9%89%E6%8E%A7%E4%BB%B6loadmorerecyclerview.html>)
2016年2月28日
RecyclerView是加强版的Lis...

Linux Mint （应用软件— 虚拟机：Virtualbox） (<http://www.fwqtg.net/linux-mint-%EF%BC%88%E5%BA%94%E7%94%A8%E8%BD%AF%E4%BB%B6-%E8%99%9A%E6%8B%9F%E6%9C%BA%E4%B8%8B%E6%96%87%E5%AE%B9%E5%99%A8.html>)
2016年2月28日
最近想自己折腾一下Linux系统本身，比...

SpringMVC源码解析 - HandlerAdater - ModelAndViewContainer上下文容器
(<http://www.fwqtg.net/springmvc%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90-handleradater-modelandviewcontainer%E4%B8%8A%E4%B8%8B%E6%96%87%E5%AE%B9%E5%99%A8.html>)
2016年2月28日
HandlerAdapter在处理请求时...



联系方式

- 🏠 北京市石景山区重聚园甲18号2层
- ☎ 010-68833838
- ☎ 010-68813838
- 📠 13051898268
- 📠 QQ 93663045
- ✉ Dongli@linux.cn (<mailto:Dongli@linux.cn>)