# Hazelcast Client设计分析

伊力哈木江·玉苏扑 2017K8009929039

## 项目简介

Hazelcast 是由Hazelcast公司开发的一款开源的分布式内存级别的缓存数据库，可以为基于JVM环境运行的各种应用提供分布式集群和分布式缓存服务。利用Hazelcast可以满足“分布式”、“集群服务”、“网格式内存数据”、“分布式缓存“、“弹性可伸缩服务”等的要求。

Hazelcast提供了对很多 Java 接口的分布式系统的实现，如Map, Queue, ExecutorService, Lock以及 JCache等接口使分布式计算变得更加简单。除了在内存中存储数据外，Hazelcast还提供了一组方便的api来访问集群中的cpu，以获得最大的处理速度。轻量化和简单易用是Hazelcast的设计目标。Hazelcast以Jar包的方式发布，因此除Java语言外Hazelcast没有任何依赖。Hazelcast可以轻松地内嵌已有的项目或应用中，并提供分布式数据结构和分布式计算工具。

Hazelcast 具有高可扩展性和高可用性。分布式应用程序可以使用Hazelcast进行分布式缓存、同步、集群、处理、发布/订阅消息等，它以一个 JAR 包的形式提供服务，并且提供了 Java, C/C++, .NET 以及 REST 客户端。在应用时，可以将Hazelcast的jar包直接嵌入到任何使用Java、C++、.NET开发的产品中，我们只需要在应用中引入一个jar包，进行简单的配置和编码就可以实现。

Hazelcast的整体架构如下：  框架

### 特点：

简单：Hazelcast基于Java语言编写，没有其他依赖。Hazelcast具有的Java util包对外相同的API和接口。只要将Hazelcast的jar包添加到classpath中，便可以快速使用JVM集群，并开始构建可扩展的应用程序。

节点对等：Hazelcast集群中的节点是对等的，集群中没有主备角色之分，可以把Hazelcast内嵌到已有的应用程序中或使用客户端服务器模式。

可扩展：Hazelcast被设计为可以扩展到成百上千个节点，新加入的节点可以自动发现集群，集群的内存存储能力和计算能力可以维持线性增加。集群内每两个节点之间都有一条TCP连接，所有的交互都通过该TCP连接。

快速处理：数据都存储在内存中，Hazelcast支持快速写和更新操作

## 项目功能支持

### 基础功能

·开放式的客户端协议 能跟多种编程语言编写的客户端能进行数据处理任务

·智能客户端 这是一个启动模式及路由模式，该模式下客户端能跟该分布式系统中所有客户端能同时通信 ·unisocket客户端 该启动模式及路由模式，该模式下客户端能跟该分布式系统中能跟单一客户端能通信，客户端启动时默认为智能模式，如需unisocket模式需要修改配置参数

·运行生命周期 检查客户端是否正在运行 正常关闭客户端 恶意终止客户端（强制关机） 添加/删除生命周期侦听器。

·备份到客户端确认 ·诊断程序

### 客户端连接

·连接策略 ·重试连接 ·蓝色/绿色部署和灾难恢复

### 分布式处理

·分布式执行 ·事件监听 ·地图侦听器的自侦听器接口 ·进入处理器

### 传输方式

·Txn地图 ·TxnMultiMap ·TxnQueue ·列表 ·TxnSet

### 邻近缓存

·支持临近缓存 ·支持HD内存 ·从上次使用位置预加载缓存

### 安全性

·SSL ·XA传输 ·相互认证 ·授权书认证 ·自定义身份验证模块

### 管理中心

·客户端邻近缓存统计信息 ·客户端运行时间统计 ·客户端操作系统统计

### 云

·Hazelcast云 ·Google云 ·Docker ·Azure ·AWS ·Apache jclouds

## 源码分析

#### 客户端主干部分

客户端的项目的结构图为如下： 

下面详细介绍几个类 client.java

```
public interface Client extends Endpoint {
     * Returns a unique UUID for this client.
```

```
    UUID getUuid();

     * Returns the socket address of this client.
    InetSocketAddress getSocketAddress();

    * Return the type of this client
    String getClientType();

     * Return the name of this client if provided, null otherwise

    String getName();

     * Return read only set of all labels of this client.
    Set<String> getLabels();
}
```

HazelcastClient.java  uml HazelcastClient.java的具体分析

```
public static Collection<HazelcastInstance> getAllHazelcastClients() {
        Set<HazelcastInstance> result = createHashSet(CLIENTS.size());
        for (InstanceFuture<HazelcastClientProxy> f : CLIENTS.values()) {
            result.add(f.get());
        }
        return Collections.unmodifiableCollection(result);
    }
```

此类中返回所有不可变更的client信息

```
  public static HazelcastInstance getHazelcastClientByName(String instanceName) {
        InstanceFuture<HazelcastClientProxy> future = CLIENTS.get(instanceName);
        if (future == null) {
            return null;
        }
```

此类中会返回instancename

```
    public static void shutdown(HazelcastInstance instance) {
        if (instance instanceof HazelcastClientProxy) {
            final HazelcastClientProxy proxy = (HazelcastClientProxy) instance;
```

```
            HazelcastClientInstanceImpl client = proxy.client;
            if (client == null) {
                return;
            }
            proxy.client = null;
            CLIENTS.remove(client.getName());

            try {
                client.shutdown();
            } catch (Throwable ignored) {
                EmptyStatement.ignore(ignored);
            } finally {
                OutOfMemoryErrorDispatcher.deregisterClient(client);
            }
        }
    }
```

此类方法中主要是完成将关闭当前客户端并把它从管理清单中清除。

```

```

**故障检测器**

Hazelcast Java客户端具有两个故障检测器：截止日期故障检测器和Ping故障检测器。

**截止时间故障检测器**

使用绝对超时来丢失/丢失心跳。超时后，成员被视为崩溃/不可用并标记为可疑。

```
    public static final HazelcastProperty HEARTBEAT_TIMEOUT
            = new HazelcastProperty("hazelcast.client.heartbeat.timeout", 60000, MILLISECONDS);

    public static final HazelcastProperty HEARTBEAT_INTERVAL
            = new HazelcastProperty("hazelcast.client.heartbeat.interval", 5000, MILLISECONDS);
```

上述代码中可以设置多长时间间隔中跟其他客户端发送heartbeat消息， 超时时间定义为6000，这时候可以怀疑集群成员，因为它尚未将任何响应发送回客户端请求。并进行进一步检查是否陷入故障。

**Ping故障检测器**

它在OSI协议的第3层上运行，并提供对硬件和其他较低级别事件的更快，更确定的检测。该检测器可能配置为在其他检测器之一怀疑某个成员后执行额外检查，或者它可以并行工作，这是默认设置。这样，可以更快地检测到硬件和网络级别的问题。

```
    private void handleIcmpPing(Node node, ClientNetworkConfig clientNetworkConfig) {
        ClientIcmpPingConfig icmpPingConfig = clientNetworkConfig.getClientIcmpPingConfig();

        Node enabledNode = getNamedItemNode(node, "enabled");
        boolean enabled = enabledNode != null && getBooleanValue(getTextContent(enabledNode).trim());
        icmpPingConfig.setEnabled(enabled);

        for (Node child : childElements(node)) {
            String nodeName = cleanNodeName(child);
            if (matches("timeout-milliseconds", nodeName)) {
                icmpPingConfig.setTimeoutMilliseconds(Integer.parseInt(getTextContent(child)));
            } else if (matches("interval-milliseconds", nodeName)) {
                icmpPingConfig.setIntervalMilliseconds(Integer.parseInt(getTextContent(child)));
            } else if (matches("ttl", nodeName)) {
                icmpPingConfig.setTtl(Integer.parseInt(getTextContent(child)));
            } else if (matches("max-attempts", nodeName)) {
                icmpPingConfig.setMaxAttempts(Integer.parseInt(getTextContent(child)));
            } else if (matches("echo-fail-fast-on-startup", nodeName)) {
                icmpPingConfig.setEchoFailFastOnStartup(Boolean.parseBoolean(getTextContent(child)));
            }
        }
        clientNetworkConfig.setClientIcmpPingConfig(icmpPingConfig);
    }
```

当时间截止故障检测器中检测到客户端长时间没有给予heartbeat回复，则启动ping故障检测器， enabled：启用旧的ICMP检测模式，与现有的故障检测器协同工作，并且仅在经过预定义的时间段且没有成员发出的心跳信号后才启动。默认值为false， timeout-milliseconds：如果没有答复，则直到尝试执行ping操作为止的毫秒数。其默认值为1000毫秒。 max-attempts：在检测器怀疑成员之前，最多可尝试ping一次。其默认值为3 interval-milliseconds：两次ping尝试之间的间隔（以毫秒为单位）。1000ms（1秒）也是允许的最小间隔。其默认值为1000毫秒。

ttl：数据包应经过的最大跳数。其默认值为255。您可以设置为0以使用系统的默认TTL。

在Ping故障检测器每秒尝试2次ping，并等待长达1秒才能完成。如果2秒钟后没有成功ping，则该成员会受到怀疑。

### 本项目中所用到的面向对象思想：

遵循面向对象的开闭原则该系统下提供接口，一个客户端能监听其他客户端的状态，但是不能对于内部的信息不是能被其他客户端修改因此它遵循了。 遵循面向对象的依赖倒转原则，该项目中具有明显的继承的面向对象思想，直接用接口调用并与具体类的上层接口交互。 遵循面向对象的接口隔离原则

桥接模式，该项目中每一个对象都是单独实现单独变化配置结束之后才能进入路由和连接状态，相当于之后的都是通过继承关系 简单工厂模式，定义抽象对象client之后通过其他对象的继承和最后能实现一个完整的功能所以可以说是具有简单工厂模式。

参考文献：


[1]https://blog.csdn.net/wy0123/article/details/79267543?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control

[2]https://blog.csdn.net/u011814346/article/details/71080847?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_baidulandingword-2&spm=1001.2101.3001.4242

[3]https://blog.csdn.net/u011814346/article/details/71081510

[4]https://blog.csdn.net/liumiao1128/article/details/53230822?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-6&spm=1001.2101.3001.4242

[5]https://docs.hazelcast.org/docs/4.1.1/manual/html-single/index.html#java-client-failure-detectors

[6]https://docs.hazelcast.org/docs/latest/javadoc/index.html?help-doc.html