Hazelcast（Executor 模块）源码阅读报告

（未完成，待补充）

2022k8009925009 王敬华

Part 0.什么是 Hazelcast?

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。

Hazelcast 提供了一个分布式内存计算框架，可以通过内存来存储和计算数据，从而加速应用程序的响应和处理速度。它最常见的功能包括分布式缓存、数据存储、任务执行等，用于提高系统的性能和可扩展性。

Hazelcast 提供了对很多 Java 接口的分布式实现，如 Map,ExecutorService, Queue, Lock 以及 JCache。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。

下面简单介绍 Hazelcast 的主要特点：

1.分布式缓存：Hazelcast 提供了基于内存的分布式缓存功能，可以在集群节点间共享数据，提高访问速度。

2.分布式数据结构：提供了多种分布式集合，如分布式 Map、Queue、List 等。

3.分布式任务执行：支持分布式的任务执行功能，可以在多个节点上并行处理任务。

4.数据分区和容错：支持将数据分区存储在不同节点上，并进行复制，确保数据在节点故障时不会丢失。

下面简单介绍在 Java 中如何使用 Hazelcast

在 Java 中使用 Hazelcast 的具体步骤如下：

1. 引入依赖：

在 Java 项目中，需要添加 Hazelcast 的依赖项。可以使用 Maven 或 Gradle 等构建工具来管理依赖关系。以下是 Maven 的示例配置：

```
<dependency>
    <groupId>com.hazelcast</groupId>
    <artifactId>hazelcast</artifactId>
    <version>4.2</version></dependency>
```

2. 创建 Hazelcast 实例：

在代码中，首先需要创建 Hazelcast 实例来启动一个 Hazelcast 集群。以下是一个创建 Hazelcast 实例的示例代码：

```
import com.hazelcast.core.Hazelcast;import com.hazelcast.core.HazelcastInstance;
 public class HazelcastExample {
    public static void main(String[] args) {
        // 创建Hazelcast 实例        HazelcastInstance hazelcastInstance =
Hazelcast.newHazelcastInstance();
```

```
        // 执行其他操作...    }}
```

3. 使用分布式数据结构：

Hazelcast 提供了各种分布式数据结构，如 Map、Queue、List 等。可以使用这些数据结构来存储和访问分布式数据。以下是一个使用 Hazelcast Map 的示例代码：

```
import com.hazelcast.core.Hazelcast;import
com.hazelcast.core.HazelcastInstance;import com.hazelcast.core.IMap;
 public class HazelcastExample {
    public static void main(String[] args) {
        // 创建Hazelcast 实例        HazelcastInstance hazelcastInstance =
          Hazelcast.newHazelcastInstance();

        // 获取分布式Map         IMap<String, String> map =
          hazelcastInstance.getMap("myMap");

        // 存储数据         map.put("key1", "value1");
        map.put("key2", "value2");

        // 获取数据         String value1 = map.get("key1");
        String value2 = map.get("key2");

        System.out.println("Value1: " + value1);
        System.out.println("Value2: " + value2);
    }}
```

4. 运行应用程序：

可以通过运行上述代码来启动你的应用程序，并使用 Hazelcast 进行分布式数据存储和访问。根据具体需求可以使用其他 Hazelcast 的功能和 API 来完成更多的任务。

我们选择分析的模块是 Hazelcast 的 Executor 模块，这是一个分布式任务执行器，允许你在 Hazelcast 集群中的不同节点上并行执行任务。它类似于 Java 的 ExecutorService，但 Hazelcast 的 Executor 是分布式的，支持将任务发送到集群的一个或多个节点上执行。这个模块特别适用于需要利用多节点的计算能力来处理大量任务的场景，比如大规模数据处理、实时分析等。

Part 1.功能分析和建模

1. Hazelcast Executor 模块概述

Hazelcast 的 Executor 模块提供了一个分布式任务执行器，支持在 Hazelcast 集群中并行执行任务。它允许开发者将任务提交到集群中的某一节点或一组节点上执行，从而实现分布式计算、任务调度、并发控制等。该模块在大规模分布式系统和实时数据处理中尤为重要，能够有效利用集群节点的资源。

其主要功能包括：
分布式任务执行：将任务分发到多个节点上并行执行，提高计算性能。
任务调度与管理：可以指定任务在特定节点或所有节点执行，灵活控制任务分配。
容错处理：如果某个节点出现故障，Executor 模块可以重新分配任务，保证任务的最终完成。

2.Hazelcast Executor 模块

Hazelcast Executor 模块不是单纯的一个调库 hazelcast.executorservice 的独到之处是:它实现了分布式环境中的异步任务。
在提异步任务之前，我们首先来探究一下 executor 框架：Executor 框架是 Java 中一个非常酷的特性，可以让我们轻松的异步执行任务，比如数据库查询，复杂计算和图像渲染等。ThreadPoolExecutor 是 Executor 框架的默认实现，它主要为在单 JVM 内运行而设计。在分布式系统中，ThreadPoolExecutor 可能不是期望的 Executor 实现，因为提交任务和执行任务的 JVM 是不同的。
为了解决这个问题，Hazelcast 提供了 IExecutorService 用于在分布式环境中执行异步任务，其中依然实现了 java.util.concurrent.ExecutorService 接口，为需要计算和数据处理能力的应用程序提供服务。为了达到异步分布式执行的目的，IExecutorService 中的任务必须支持序列化以在不同的 JVM 之间通过网络传输（代码中也有一部分涉及到了序列化处理）
重点就是分布式的实现，具体实现方式会在之后的源码分析中给出序列化也是这一部分的重点。

3. Hazelcast Executor 模块需求建模

围绕着 Hazelcast 的 Executor 模块的主要功能，我们对其进行功能建模分析（暂时可能还不完善，后面可能进行补充）

[用例名称]
分布式任务执行
[用例描述]
通过 Hazelcast 的 Executor 模块将计算任务分发到 Hazelcast 集群的不同节点上并行执行，完成高效的分布式计算，并能够在任务执行后收集结果。
[场景]
Who：分布式系统的开发者、数据处理工程师、后台服务。
Where：在 Hazelcast 集群环境中运行，包括分布式缓存、数据处理、微服务集群等场景。
When：需要执行并行计算任务时，如批量数据处理、数据分析、实时数据监控等。
[用例价值]
Hazelcast 的 Executor 模块提供了分布式任务执行的能力，使开发者可以在集群中的多个节点上并行处理计算任务，最大化资源利用率，降低单点压力，提高系统的整体处理效率。同时，模块还支持自动处理节点故障，使得任务执行更具容错性，增强系统的可靠性。
[约束和限制]
任务可序列化：所有提交的任务必须实现 Serializable 接口，以便任务可以在网络上传输到不同的节点。

任务执行环境依赖：任务所需的环境（如特定的库、资源）必须在所有节点上都可用，否则可能导致执行失败。
节点负载均衡：需要合理的负载均衡策略，以确保任务在节点间的合理分配，避免部分节点过载。
故障重试机制：当节点故障或任务失败时，需要设计合理的重试和重新分配机制，以确保任务最终完成。
同步与异步结果处理：使用 Future 对象时，应处理好异步任务的结果，避免因阻塞导致性能下降。

[示例用例名称]
特定节点任务分配
[用例描述]
通过 Hazelcast 的 Executor 模块，将一个计算任务分配到 Hazelcast 集群中的指定节点上执行。
[场景]
Who：系统管理员、任务分发管理者。
Where：在 Hazelcast 集群中指定的节点上执行，例如对特定数据源的数据处理或节点健康检查。
When：需要将任务仅分配给特定节点时。
[用例价值]
指定任务执行节点可以实现任务的精准调度，适用于需要特定节点资源或环境的任务，提高集群的可控性和任务分配的灵活性。
[约束和限制]
节点选择：需要确保目标节点在线，且具备执行任务的资源，防止任务失败。
节点状态检测：在任务执行前，需要确认节点的健康状态，以保证任务能够成功执行。

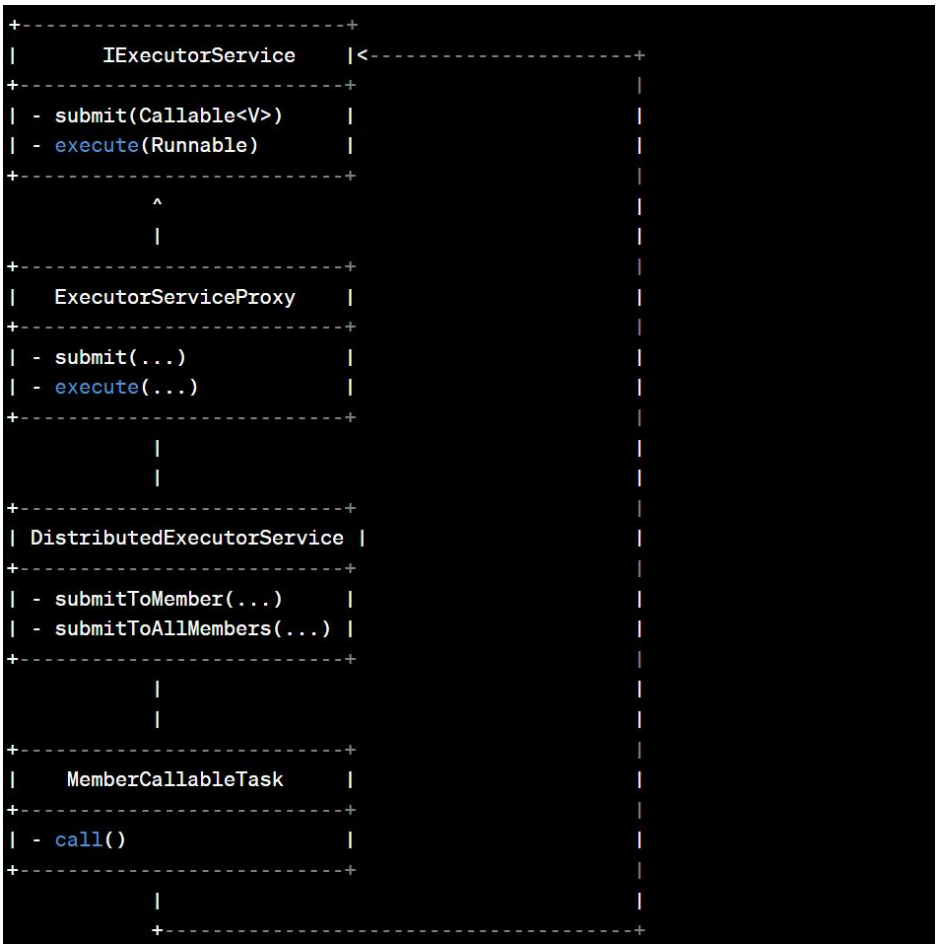
4. Hazelcast Executor 模块的工作流程

Hazelcast Executor 模块的工作流程可以分为以下几个阶段：
（1）任务提交：
客户端通过 IExecutorService 提交一个 Callable 或 Runnable 任务。任务可以指定执行节点(submitToMember)或执行策略(例如:所有节点都执行 submitToAllMembers)。
（2）任务分发：
ExecutorServiceProxy 将任务请求转发到 DistributedExecutorService，由后者确定合适的节点并将任务分发。
（3）任务执行：
任务到达目标节点后，节点启动该任务。MemberCallableTask 或 RunnableTask 执行 call() 或 run() 方法，完成分布式逻辑处理。
（4）结果返回与收集：
如果任务有返回值 (Callable)，节点会将结果发送回客户端。Hazelcast 使用 Future 机制管理异步任务的结果收集，客户端可以通过 Future.get() 等方法获取结果。
（5）故障处理：
如果执行节点发生故障，Hazelcast 具备自动重试的机制，将任务重新分发到其他节点，

保证任务的最终执行。

5. Hazelcast Executor 模块的建模

在建模过程中，可以将 IExecutorService 视作集群范围内的执行管理器，DistributedExecutorService 则是具体的执行调度器，而 MemberCallableTask 是执行的具体逻辑。以下是一个简化的 UML 类图的描述，用于理解模块的整体结构和调用关系：（时间原因先草草画一个 hhh 后面再改）



Part 2.核心流程设计分析

Part 3.高级设计意图分析

Part 4.源码阅读总结