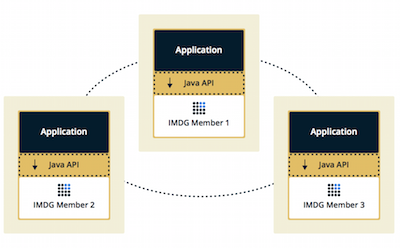
# Hazelcast源码分析

Hazelcast有两种运行模式：P2P(点对点模式)和CS模式（扩展P2P）。P2P模式的拓扑结构如下图所示：



在P2P模式下，所有的节点都是集群中的服务节点，提供相同的功能和计算能力，数据访问延迟较低，适合于构建计算密集型的集群。由于每个节点都分担集群的总体性能，每增加一个节点都可以线性增加集群能力。

Hazelcast可以将多个应用服务器组成一个分布式环境，在Cluster中选举出Master对外提供工作，服务器之间需要进行数据同步和数据备份机制来避免单节点挂掉而导致的数据丢失和Cluster失效。数据存储在分布式内存中，可以保证各个节点的均匀分布。

# Hazelcast构成集群

1. Hazelcast节点启动后根据配置的寻找机制查找其他节点，主要包括：

* multicast，多播的方式
* TCP/IP

1. 如果查找不到其他节点，则选举自己为master。
2. 后面加入的节点向已存在的cluster的Master发出join请求
3. Master检测是否符合集群的加入要求，符合则会发送最新的成员类表给新加入的成员，同时更新集群中的数据
4. 集群中每个成员都有相同的成员类表，按加入的顺序排序，如果master挂掉，那么其他成员会收到通知，然后从成员类表中选举下一个作为master

## Hazelcast Node

Node是Hazelcast 节点的抽象，其中包含节点引擎、客户端引擎、分区服务、集群服务、组播服务、连接管理、命令管理、组播属性、节点配置、本地成员、管理中心、安全上下文等各功能模块，其各模块如下所示

*public final HazelcastInstanceImpl hazelcastInstance;*

*public final DynamicConfigurationAwareConfig config;*

*public final NodeEngineImpl nodeEngine;*

*pulic final ClientEngineImpl clientEngine;*

*public final InternalPartitionServiceImpl partitionService;*

*public final ClusterServiceImpl clusterService;*

*public final MulticastService multicastService;*

*public final DiscoveryService discoveryService;*

*public final TextCommandServiceImpl textCommandService;*

*public final LoggingServiceImpl loggingService;*

*public final ConnectionManager connectionManager;*

*public final Address address;*

*public final SecurityContext securityContext;*

*private final ILogger logger;*

*private final AtomicBoolean shuttingDown = new AtomicBoolean(false);*

*private final NodeShutdownHookThread shutdownHookThread;*

*private final PhoneHome phoneHome = new PhoneHome();*

*private final InternalSerializationService serializationService;*

*private final ClassLoader configClassLoader;*

*private final NodeExtension nodeExtension;*

*private final HazelcastProperties properties;*

*private final BuildInfo buildInfo;*

*private final HealthMonitor healthMonitor;*

*private final Joiner joiner;*

*private ManagementCenterService managementCenterService;*

*private volatile NodeState state;*

*private final MemberVersion version;*

1. **Config,** 包含GroupConfig：集群用户名和密码; NetworkConfig: 网络相关配置；MapConfig、TopicConfig等
2. **ClusterServiceImpl**，用户维护集群中各个成员节点，实例化时将本地节点加入到Member Map中
3. **MulticastService，**Node在实例化时根据MulticastConfig使用组播加入组，基于组播传递，使用监听模式每当接收到其他节点传播的消息调用监听器的onMessage，传递的参数为JoinMessage；默认的监听器为NodeMulticastListener监听器，对集群节点进行校验，是否为同集群的用户名和密码
4. **Joiner**，用于加入集群节点，创建JoinRequest对象用于发送给其他成员，发送请求到集群的Master，其实现包括MultiJoiner,TcpIPJoiner及DiscoveryJoiner。集群Master节点的获取是通过循环发送JoinRequest消息向组内发送，如果已加入状态且是master节点接收到后向外组播JoinMessage，告诉其他节点组成员信息，还没有加入集群的成员则将自己节点的master地址设置为master发出的JoinMessage中的地址。
5. **SerializationService**，序列化转换模块，提供各种类型的序列化支持，toData提供Object到Data的转化，toObject为反向转化操作。
6. **ConnectionManager**，用于管理连接，与其他Member进行通信
7. **ManagementCenterService**，用于向ManagementCenter发送统计信息
8. **OperationService**，用于Member接收及发送请求
9. **NodeEngine**，节点引擎包含许多服务，例如事件服务、operation服务、执行服务、内置Service(例如Map/Queue等)管理服务等。其中核心是ServiceManageImpl用于管理所有远程启动是注册的常用服务，

*private void registerDefaultServices(ServicesConfig servicesConfig) {*

*registerService(MapService.SERVICE\_NAME, createService(MapService.class));*

*registerService(LockService.SERVICE\_NAME, new LockServiceImpl(nodeEngine));*

*registerService(QueueService.SERVICE\_NAME, new QueueService(nodeEngine));*

*registerService(TopicService.SERVICE\_NAME, new TopicService());*

*registerService(ReliableTopicService.SERVICE\_NAME, new ReliableTopicService(nodeEngine));*

*registerService(MultiMapService.SERVICE\_NAME, new MultiMapService(nodeEngine));*

*registerService(ListService.SERVICE\_NAME, new ListService(nodeEngine));*

*registerService(SetService.SERVICE\_NAME, new SetService(nodeEngine));*

*registerService(DistributedExecutorService.SERVICE\_NAME, new DistributedExecutorService());*

*registerService(DistributedDurableExecutorService.SERVICE\_NAME, new DistributedDurableExecutorService(nodeEngine));*

*registerService(AtomicLongService.SERVICE\_NAME, new AtomicLongService());*

*registerService(AtomicReferenceService.SERVICE\_NAME, new AtomicReferenceService());*

*registerService(CountDownLatchService.SERVICE\_NAME, new CountDownLatchService());*

*registerService(SemaphoreService.SERVICE\_NAME, new SemaphoreService(nodeEngine));*

*registerService(IdGeneratorService.SERVICE\_NAME, new IdGeneratorService(nodeEngine));*

*registerService(FlakeIdGeneratorService.SERVICE\_NAME, new FlakeIdGeneratorService(nodeEngine));*

*registerService(MapReduceService.SERVICE\_NAME, new MapReduceService(nodeEngine));*

*registerService(ReplicatedMapService.SERVICE\_NAME, new ReplicatedMapService(nodeEngine));*

*registerService(RingbufferService.SERVICE\_NAME, new RingbufferService(nodeEngine));*

*registerService(XAService.SERVICE\_NAME, new XAService(nodeEngine));*

*registerService(CardinalityEstimatorService.SERVICE\_NAME, new CardinalityEstimatorService());*

*registerService(PNCounterService.SERVICE\_NAME, new PNCounterService());*

*registerService(CRDTReplicationMigrationService.SERVICE\_NAME, new CRDTReplicationMigrationService());*

*registerService(DistributedScheduledExecutorService.SERVICE\_NAME, new DistributedScheduledExecutorService());*

*registerCacheServiceIfAvailable(); readServiceDescriptors();}*

# 1.2 节点加入集群及选择Master

Hazelcast集群由运行Hazelcast Node的实例构成，当节点启动后自动Join到集群中，节点启动后，会先寻找其他节点，寻找的机制在hazelcast.xml中进行配置，配置如下所示：

*<join>*

*<multicast enabled="true">*

*<multicast-group>224.2.2.3</multicast-group>*

*<multicast-port>54327</multicast-port>*

*</multicast>*

*<tcp-ip enabled="false">*

*<interface>127.0.0.1</interface>*

*<member-list>*

*<member>127.0.0.1</member>*

*</member-list>*

*</tcp-ip>*

*<aws enabled="false">*

*......*

*</aws>*

*<discovery-strategies>*

*</discovery-strategies>*

*</join>*

目前主要支持的Join机制有：

* Multicast，默认配置多播的方式，但是在生产环境中不使用这种方式。
* Tcp-Ip，通过在配置文件中配置member的IP来实现
* Discovery，在Hazelcast中可以通过配置服务发现机制来构成集群，包括Zookeeper 、Consul、etcd、Eureka等服务发现机制

### 1.2.1 Discovering Members by TCP

在配置文件中配置所有成员的hostname或者IP地址，配置示例如下：

*<tcp-ip enabled="true">*

*<member>machine1</member>*

*<member>machine2</member>*

*<member>machine3:5799</member>*

*<member>192.168.1.0-7</member>*

*<member>192.168.1.21</member>*

*</tcp-ip>*

配置tcp-ip项为true，通过<member>配置节点成员，也可以配置IP区域段。如果不配置端口，则尝试使用5701/5702等端口。

在Node启动后，调用join加入集群中，执行如下：

*public final void join() {*

*blacklistedAddresses.clear();*

*doJoin();*

*if (!clusterService.isJoined() && shouldResetHotRestartData()) {*

*node.getNodeExtension().getInternalHotRestartService().resetHotRestartData();*

*reset();*

*doJoin();*

*}*

*postJoin();*

*}*

在doJoin中进行的操作顺序如下：

1）根据hazelcast-default.xml中的配置，构建Member列表：possibleAddresses

*addressMatcher = AddressUtil.getAddressMatcher(addressHolder.getAddress());*

*for (String matchedAddress : matchedAddresses) {*

*addPossibleAddresses(possibleAddresses, null,*

*InetAddress.getByName(matchedAddress), port, count);*

*}*

*return possibleAddress*

1. Node与集群中的Member交互，获取Master Address

*for (Address address : possibleAddresses) {*

*if (clusterJoinManager.sendMasterQuestion(address)) {*

*sent = true;*

*}*

*}*

1. Node#ClusterJoinManager，向Member发送MasterQuestion

*public boolean sendMasterQuestion(Address toAddress) {*

*.....*

*return nodeEngine.getOperationService().send(new WhoisMasterOp(joinMessage), toAddress);*

*}*

OutboundOperationHandler#send 向toAddress发送joinMessage，获取是否为master

*Connection connection = node.getConnectionManager().getOrConnect(target);*

*return send(op, connection);*

其执行的过程如下：

*public boolean send(Operation op, Connection connection) {*

*byte[] bytes = serializationService.toBytes(op); //对WhoisMasterOp进行序列化*

*int partitionId = op.getPartitionId();*

*Packet packet = new Packet(bytes, partitionId).setPacketType(Packet.Type.OPERATION);*

*if (op.isUrgent()) {*

*packet.raiseFlags(FLAG\_URGENT);*

*}*

*return node.getConnectionManager().transmit(packet, connection);*

*//将信息传递给目标Member*

*}*

1. 目标Member收到请求后，进行响应

*public void run() {*

*ClusterServiceImpl cm = getService();*

*cm.getClusterJoinManager().answerWhoisMasterQuestion(joinMessage, getConnection());*

*}*

响应为：

*public class MasterResponseOp extends AbstractClusterOperation {  
 protected Address masterAddress;*

*}*

目标Member将自己已知的Master Address发送给新启动的Node。

1. Master的选举由ClusterJoinManager完成

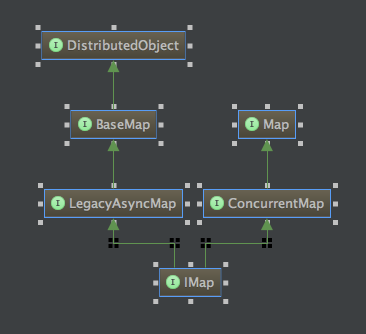
# 分布式Map

Hazelcast是一个高度可扩展的数据分发和集群平台，其提供的功能之一是数据结构的分布式实现，Distributed Map的使用方法如下：

*HazelcastInstance instance = Hazelcast.newHazelcastInstance(config);*

*Map<Integer,String> clusterMap = instance.getMap("MyMap");*

数据分布式保存在集群的某节点上，而且会有备份，节点down掉，数据也不会丢失。IMap的类结构如下：



IMap继承了JDK的ConcurrentMap，而且还继承DistributedObject接口，其具有PartitionKey，Hazelcast使用该字段进行分区。

### **IMap的初始化**

getMap的过程其实就是从集群内部Node获取其保存在哪里过程，由于IMap其实就是DistributedObject，本质上就是getDistributedObject过程：

*public <K, V> IMap<K, V> getMap(String name) {checkNotNull(name, "Retrieving a map instance with a null name is not allowed!");*

*return getDistributedObject(MapService.SERVICE\_NAME, name);*

*}*

*public <T extends DistributedObject> T getDistributedObject(String serviceName, String name) {  
 ProxyService proxyService = node.nodeEngine.getProxyService();  
 return (T) proxyService.getDistributedObject(serviceName, name);  
}*

使用代理ProxyService获取DistributedObject的IMap，其通过注册在内存中的ProxyRegistry来处理，如下所示：

*public DistributedObject getDistributedObject(String serviceName, String name) {*

*ProxyRegistry registry = getOrCreateRegistry(serviceName);*

*return registry.getOrCreateProxy(name, true);*

*}*

在ProxyRegistry内部包含RemoteService，进行远程或者本地调用来获取DistributedObject，进入MapRemoteService逻辑：

*public DistributedObject createDistributedObject(String name) {*

*Config config = nodeEngine.getConfig();*

*MapConfig mapConfig = config.findMapConfig(name);*

*MergePolicyProvider mergePolicyProvider =mapServiceContext.getMergePolicyProvider();*

*Object mergePolicy = mergePolicyProvider.getMergePolicy(mapConfig.getMergePolicyConfig().getPolicy());*

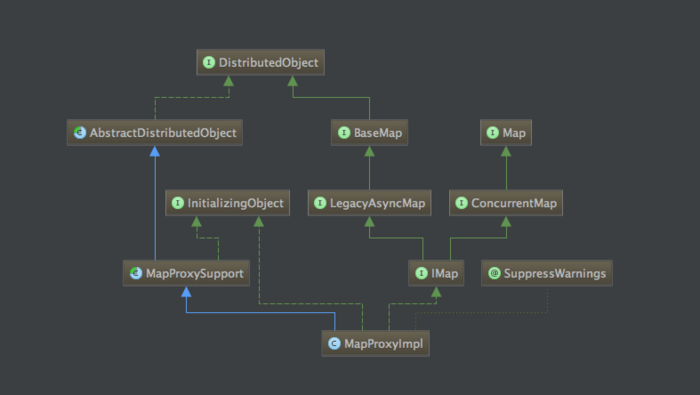
*......*

*return new MapProxyImpl(name, mapServiceContext.getService(), nodeEngine, mapConfig);*

*}*

*}*

DistributedObject其实现为MapProxyImpl，类图如下：



### **数据的写入及读取**

数据的获取及写入都是通过MapProxySupport内部实现的

*protected Data putInternal(Object key, Data value, long ttl, TimeUnit timeunit) {*

*Data keyData = toDataWithStrategy(key);  
 long timeInMillis = getTimeInMillis(ttl, timeunit);  
 MapOperation operation = operationProvider.createPutOperation(name, keyData, value, timeInMillis);  
 return (Data) invokeOperation(keyData, operation);*

*}*

在invokeOperation中执行写入：

*private Object invokeOperation(Data key, MapOperation operation) {*

*int partitionId = partitionService.getPartitionId(key);*

*operation.setThreadId(getThreadId());*

*.....*

*Future future = operationService*

*.createInvocationBuilder(SERVICE\_NAME, operation, partitionId)*

*.setResultDeserialized(false)*

*.invoke();*

*result = future.get();*

*}*

1. 根据key，从PartitionService中获取PartitionId
2. OperationService，根据partitionId获取所要操作的主机

*public Address getTarget() {*

*IPartition partition = context.partitionService.getPartition(op.getPartitionId());*

*return partition.getReplicaAddress(op.getReplicaIndex());*

*}*

1. 将数据写入对应的主机中，PartitionInvocation#invoke

*private void doInvokeRemote() {*

*Connection connection = context.connectionManager.getOrConnect(invTarget);*

*this.connection = connection;*

*if (!context.outboundOperationHandler.send(op, connection)) {*

*....*

*}*

*}*

将操作发送至Target主机。

http://itfish.net/article/25133.html

https://blog.csdn.net/wangyangzhizhou/article/details/52677826

https://blog.csdn.net/wangyangzhizhou/article/details/47258457