基于空间的软件架构(云架构模式)

——Apache Geode浅析

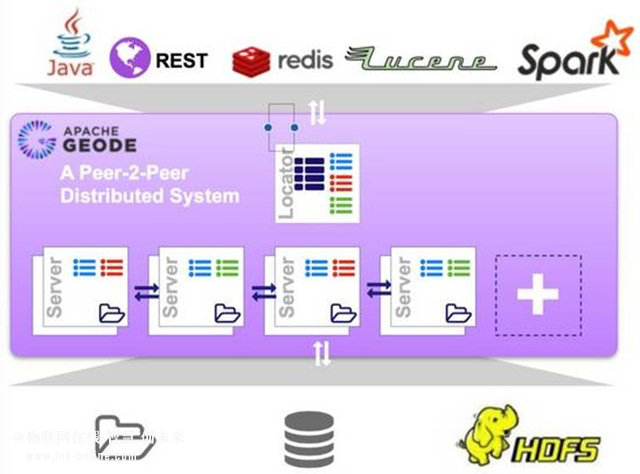
基于空间的架构模型（The space-based pattern，也称为云架构模型），旨在减少限制应用伸缩的因素。The Space-Based Pattern来源于分布式共享内存中的元组空间（Tuple Space）的概念，其通过数据网格来替代数据库来实现高伸缩性，应用需要的数据保存并复制给所有运行的进程，并且这些进程动态的随着请求数量的增减而启动或结束，通过这种方式可以避免中心数据库的瓶颈问题，同时提供近乎无限的扩展能力。数据管理平台Apache Geode使用了该架构模型，下面介绍该平台的使用。

# **1.系统架构与概念**

Apache Geode是一个数据管理平台，提供实时、一致及贯穿整个云架构地访问数据关键型应用，其实基于JVM的NoSQL分布式数据处理平台，同时集中间件、缓存、消息队列、事件处理引擎、NoSQL数据库于一身的分布式数据处理平台。可用来进行完成分布式缓存、数据持久化、分布式事务和动态扩展等功能。

## 1.1系统架构

系统总体架构如下图所示：

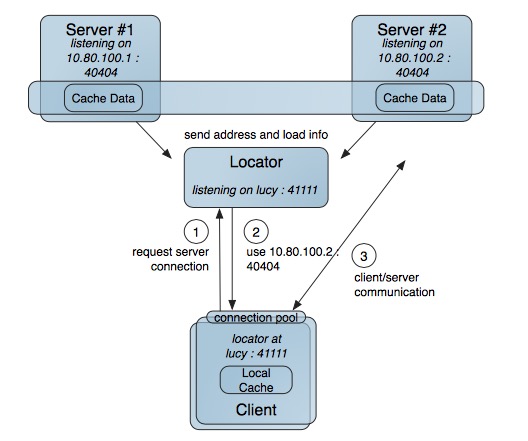


系统核心理念是跨集群节点分布数据、完全复制数据到所选节点或者跨所节点的分区数据，当数据条目在内存中更新时同步到一个或者多个内存池中的节点，以此来保障数据一致性和可靠性。Apache Geode更是提供跨网络，及时地Delta更新传播技术并通过使用内存访问来替代磁盘I/O访问，在高可靠和日益增长的带宽下以池化的形式处理数据。在Geode中核心的服务进程包括

* Locator（定位器），用于为新连接提供正在运行的成员位置，并为服务器的使用提供负载均衡。默认情况下，定位器启动了JMX Manager，用于集群的监视和管理。集群配置服务使用定位器来保持集群配置并将配置分配给集群成员
* Server（服务器），分成集群和组两个级别，组级别可以覆盖集群级别的配置

## 1.2基本流程

在Apache Geode中，成员之间以点对点的网络形式互相连接形成分布式系统，集群支持成员的动态加入和离开而不影响其他成员的运行。在较高的SLA要求下，可以将其内置在有状态的应用中，集群拓扑图如下所示：



在集群中启动多个Cache Server来为客户端提供缓存服务，在集群中以Region的形式组织数据，数据在集群节点之间复制或者切分，其基本的访问流程：

1. Server启动后，将地址和加载信息发送到Locator
2. 客户端将请求发送给Locator后，其返回在Locator中保存的Server信息，如果启动了负载均衡机制，则周期性的查询Server信息
3. 客户端在启动时订阅事件，这些事件路由到Server后将返回信息返回给客户端Listener
4. 客户端数据不满足需要，则向Server发送数据更新请求来获取数据

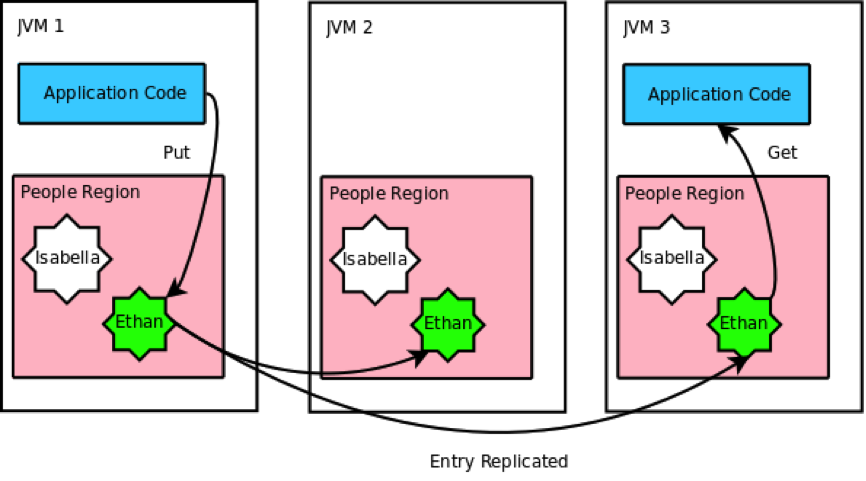
Apache Geode支持Server分组，可以将数据集合维护在某个Server组中，以此提高系统的性能和扩展性。Locator中运行了基于Tcp的位置服务，每个新的VM连接到已有的Peer上。

## 1.3基本概念

Apache Geode是内存数据管理系统，可提供可靠的异步事件通知和有保证的消息传递，其涉及的主要概念：

1. GemFire Distributed System，运行在Geode上的分布式系统，每个VM都作为GemFire对等体存在，在每个VM上创建一个缓存，缓存中间的管理通过UDP或者TCP位置服务进行互相发现
2. Regions，在分布式系统之上的抽象概念，一个Region运行在系统的多个VM上，通过接口可以透明的从分布式系统上获取数据
3. Replicated Regions，保存着所有分区的数据拷贝
4. Partitioned Regions，保存分区的一部分的数据拷贝
5. Client Caching，在Geode中分布式系统是一个网格，所有的Peer直接互相连接，也支持客户端接入分布式系统，在客户端中有本地数据缓存，并且通过注册服务器来获取最新数据
6. Shared-Nothing Persistence，支持非共享的持久化，每个Peer持久化化数据到本地磁盘

Geodo Region是键值集合，每个Replicated Region的peer都存储了整个Region的拷贝，对region所做的任何改变都同步发送到所有的Peer中



# 2.使用入门

## 2.1安装

1) 下载安装包

*wget https://mirrors.tuna.tsinghua.edu.cn/apache/geode/1.12.0/apache-geode-1.12.0.tgz*

*tar -zxvf apache-geode-1.12.0.tgz*

*cd /opt/apache-geode-1.12.0*

2）添加bin到path

*vi /etc/profile*

*export PATH="$PATH:/opt/apache-geode-1.12.0/bin"*

*source /etc/profile*

3) 测试是否安装成功

*# gfsh version --full*

*Build-Date: 2020-03-27 11:09:15 -0700*

*Build-Id: echobravo 0*

*Build-Java-Vendor: AdoptOpenJDK*

*Build-Java-Version: 1.8.0\_242*

*......*

*Running on: /10.139.12.159, 4 cpu(s), amd64 Linux 3.10.0-514.el7.x86\_64*

## **2.2启动**

1) 启动Locator，请求转发器，负责将各个查询转发到各个Server上，默认端口：10334

*# gfsh*

*Monitor and Manage Apache Geode*

*gfsh>start locator --name=fyscmh3-locator*

*Starting a Geode Locator in /opt/apache-geode-1.12.0/fyscmh3-locator...*

*...............*

*Successfully connected to: JMX Manager [host=fyscmh3, port=1099]*

2) 启动Server

*gfsh>start server --name=fyscmh3-server --server-port=40411*

*Starting a Geode Server in /opt/apache-geode-1.12.0/fyscmh3-server...*

*.......*

*Server in /opt/apache-geode-1.12.0/fyscmh3-server on fyscmh3[40411] as fyscmh3-server is currently online.*

*Process ID: 2723*

1. 查看集群的情况

*gfsh>list members*

*Member Count : 2*

*Name | Id*

*--------------- | -----------------------------------------------------------------------*

*fyscmh3-locator | 10.139.12.159(fyscmh3-locator:1317:locator)<ec><v0>:41000 [Coordinator]*

*fyscmh3-server | 10.139.12.159(fyscmh3-server:2723)<v1>:41001*

## **2.3使用**

1) 使用connect命令连接locator

*gfsh>connect --locator=fyscmh3[10334]*

*Connecting to Locator at [host=fyscmh3, port=10334] ..*

*Connecting to Manager at [host=fyscmh3, port=1099] ..*

*Successfully connected to: [host=fyscmh3, port=1099]*

1. 创建region，可以认为是数据库中的表

*gfsh>create region --name=hiA --type=REPLICATE\_PERSISTENT*

*Member | Status | Message*

*-------------- | ------ | -----------------------------------------*

*fyscmh3-server | OK | Region "/hiA" created on "fyscmh3-server"*

*Cluster configuration for group 'cluster' is updated.*

1. 查看region列表

*gfsh>list regions*

*List of regions*

*---------------*

*hiA*

1. 插入数据

*gfsh>put --region=hiA --key="Hello" --value="Geode"*

*Result : true*

*Key Class : java.lang.String*

*Key : Hello*

*Value Class : java.lang.String*

*Old Value : null*

1. 查询数据

*gfsh>query --query="select \* from /hiA"*

*Result : true*

*Limit : 100*

*Rows : 1*

*Result*

*------*

*Geode*

1. Java API

* 在pom中添加依赖

*<dependency>  
 <groupId>org.apache.geode</groupId>  
 <artifactId>geode-core</artifactId>  
 <version>1.12.0</version>  
</dependency>*

* 测试程序

*public class HelloGeode {*

*public static void main(String[] args) throws Exception {*

*ClientCache cache = new ClientCacheFactory()*

*.addPoolLocator("fyscmh3", 10334)*

*.create();*

*Region<String, String> region = cache.*

*<String, String>createClientRegionFactory(ClientRegionShortcut.CACHING\_PROXY)*

*.create("hiA");*

*region.put("1", "Hello");*

*region.put("2", "world");*

*for (Map.Entry<String,String> entry : region.entrySet()) {*

*System.out.format("key = %s, value = %s\n", entry.getKey(), entry.getValue());*

*}*

*cache.close();*

*}*

*}*

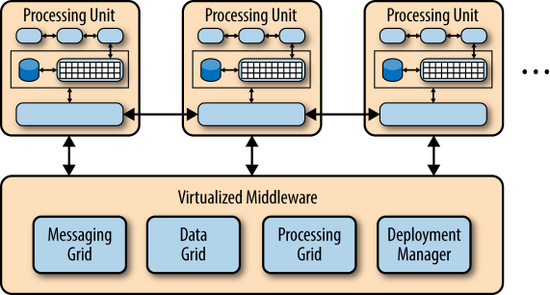
* 执行后，输出内容如下：

*key = 1, value = Hello*

*key = 2, value = world*

# **深入解析**

Apache Geode是基于JVM的NoSQL分布式数据处理平台，同时集中间件、缓存、消息队列、事件处理引擎、NoSQL数据库等功能，可用于完成分布式缓存、数据持久化、分布式事务、动态扩展等功能，其是贯穿整个云架构的访问数据关键型应用，对于云架构的核心组件如下图:



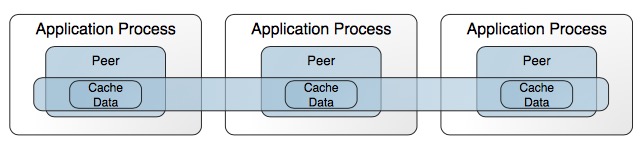
## **3.1 消息通信系统**

Apache Geode设计的核心理念是跨集群节点来分布数据，完全复制数据到所选节点或者跨所选节点的分区数据。这些成员之间以点对点的网络形式互相连接形成分布式系统，并且支持动态成员关系，集群中的成员能够随时加入和离开分布式系统而不影响其他成员的正常运行。成员之间通过多播或者基于TCP/IP的位置发现服务来进行组网，如果没有启用多播，系统将自动选举成员协调者，协调者负责授权其他成员加入到分布式系统中，并把成员关系的改变通知给系统的所有成员。Apache Geode提供各种缓存拓扑：

* 所有系统的核心是单个对等集群
* 对于水平和垂直扩展，可以将系统集合到客户端/服务器和多站点（WAN）拓扑中
* 在客户端/服务器系统中，少数服务器进程管理更大的客户端组的数据和事件处理
* 在多站点系统中，几个地理上不同的系统松散地耦合到单个、内聚的处理单元中

1. 对等集群

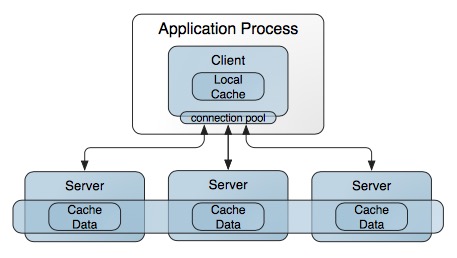
单独对等式最简单的拓扑，每个缓存实例或成员直接与集群中每个其他成员通信，此缓存配置主要是为需要在应用程序进程空间中嵌入缓存并参与集群的应用程序而设计的。典型示例是应用程序服务器集群，而其中应用程序和缓存位于同一进程并共享JVM堆



2) 客户端/服务器

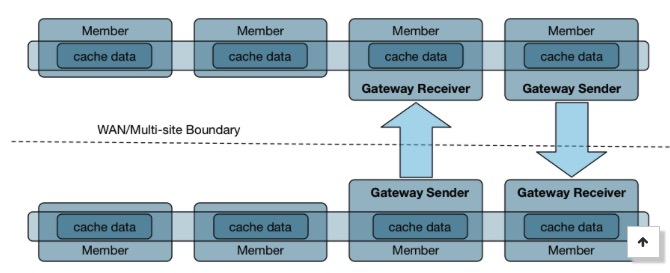
客户端/服务器拓扑是垂直拓展的模型，其中客户端通常在应用程序进程空间中托管一部分数据，并将其余部分委托给服务器系统。与点对点本身相比，客户端/服务器架构提供了更好的数据隔离、高获取性能和更高的可扩展性。如果数据分发会给网络带来高负载，那么客户端/服务器架构通常会提供更好的性能。

客户端/服务器系统中，服务器系统本身是对等系统，数据在服务器之间分配，客户端系统具有连接池，用于与服务器和其他Geode成员通信，客户端还可以包含本地缓存，拓扑架构如下图所示：



3）多点配置

对于水平扩展，可以使用松散耦合的多站点拓扑，即多个Geode系统松散耦合，通常跨域地理距离，连接速度较慢（WAN）。此拓扑提供了比单个系统的紧密耦合更好的性能，以及位置之间更大的独立性，因此如果连接或远超端点不可用，每个站点都可以自行运行。在多站点的安装中，每个站点都是对等或客户端/服务器系统，拓扑结构如下图：



## **3.2内存管理**

Caches是描述Geode集群中节点的抽象，应用程序架构师可以在对等网络或客户端/服务器拓扑中安排这些节点。在Cache节点中定义数据区域（Region），类似于关系数据库中的表，但是以键值对的形式管理数据，分区区域在缓存成员之间传播数据。在集群中数据区域存储多个副本。系统配置后，客户端应用程序可以在不了解底层系统体系结构情况下访问区域中分布式数据，用户可以创建监听器的方式在数据发生变动时自动创建通知，并且可以定义到期条件以删除区域中的过期数据。对于大型生产系统，Geode提供定位器（Locator）来进行事先服务的发现和负载均衡，客户端配置定位器服务列表，默认情况下客户端和服务器使用端口40404相互发现。

Geode中将缓存中的数据组织到数据区域中(Region)，核心实现类为GemFileCacheImpl，主要成员：

*List<InternalCacheServer> allCacheServers //region与cacheServer构成了内存网格*

*Set<PartitionedRegion> partitionedRegions*

*ConcurrentMap<String, DiskStoreImpl> diskStores*

*Map<String, InternalRegion> rootRegions*

*ConcurrentMap<String, InternalRegion> pathToRegion*

对于ClientCache，用于管理客户端的缓存。

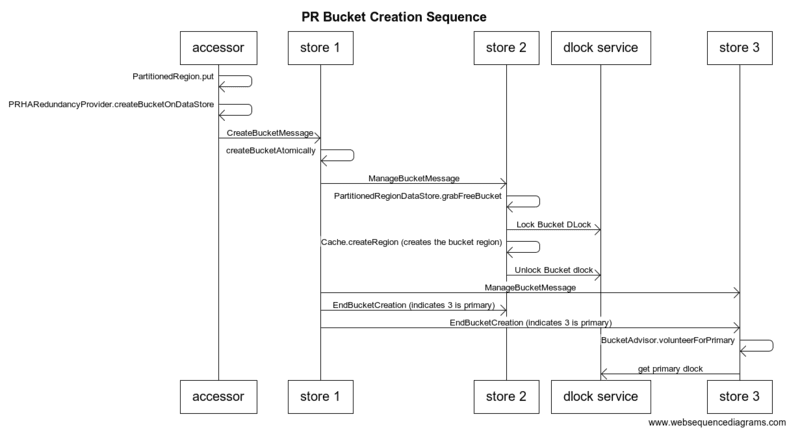
1. 数据区域（Region）管理，接口如下：

*public interface Region<K, V> extends ConcurrentMap<K, V>*

实现分布式KV的HashMap，并提供分布式事务、持久化及分区等高级功能，实现如下：

* LocalRegion，实现公共区域接口数据
* DistributedRegion，用于复制区域的类型，扩展了LocalRegion，但是具有用于消息传递的Advisor
* PartitionedRegion，分区区域，根据key的哈希码进行分桶，然后将桶分给各个成员

下图是分区桶中数据存储过程：



PartitionedRegion与DistributedRegion不同点在于，其选择一个成员作为桶的主要成员，当用户进行put操作是将其放置到主服务器上。

1. 数据Entry

Data Entry是存储在Region中的键值对，其入口是Region.put(K,V)，使用示例：

*String name = ...*

*String value = ...*

*this.currRegion.put(name,value);*

在DistributedRegion及PartitonedRegion中，数据以Event为方式分发到各CacheServer的存储桶中

*RegionEntry basicPutEntry(EntryEventImpl event, long lastModified)*

*throws TimeoutException, CacheWriterException {*

*final boolean isTraceEnabled = logger.isTraceEnabled();*

*if (requiresOneHopForMissingEntry(event)) {*

*RegionEntry re = getRegionEntry(event.getKey());*

*if (re == null /\* || re.isTombstone() \*/ || !generateVersionTag) {*

*final boolean ifNew = false;*

*final boolean ifOld = false;*

*boolean didDistribute = RemotePutMessage.distribute(event, lastModified, ifNew, ifOld, null,*

*false, !generateVersionTag);*

*......*

*}*

*}*

*return super.basicPutEntry(event, lastModified);*

*}*

在Region之上支持索引机制来确定向哪些节点发送查询、索引管理器IndexManager负责索引的创建、删除和更新等，不再赘述。

## **3.3处理器管理**

Geode的处理单元管理是基于消息通信系统来实现，通过ClusterDistributionManager进行Member的管理，在DistributedMemeber中封装了Member信息(端口、vmPid等)。底层的通信时通过DistributionAdvisor实现，例如当发现新Server时，将其添加到Listener中

*private void notifyListenersMemberAdded(InternalDistributedMember member) {*

*for (MembershipListener membershipListener : membershipListeners.keySet()) {*

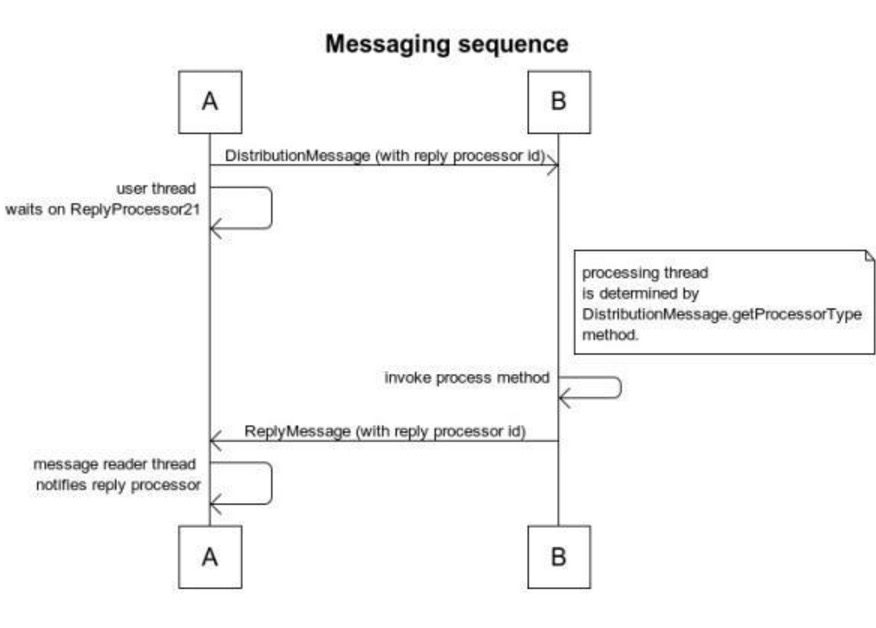
*try {*

*membershipListener.memberJoined(getDistributionManagerWithNoCheck(), member);*

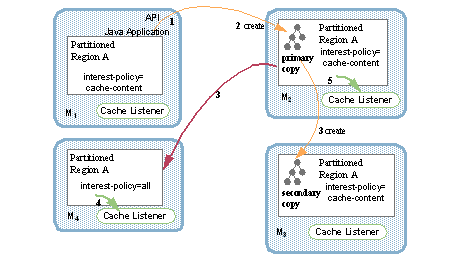
*} ....*

*}}*

成员之间的消息通信过程如下所示：

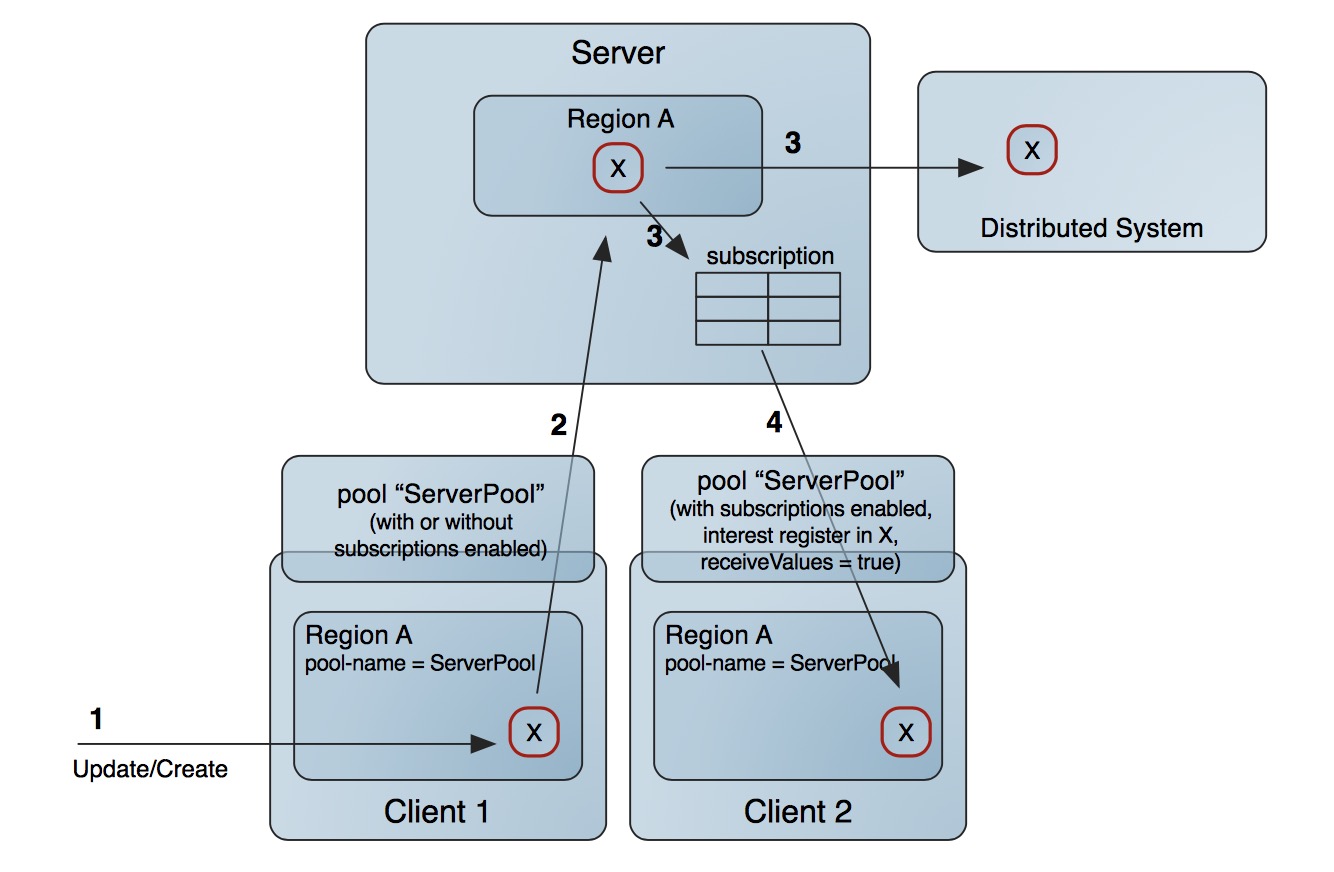


创建DistributedMessage，之后调用DistributedManage#putOutGoing来发送消息。执行Region或者Entry操作时，根据系统和缓存配置在集群中分配关联的事件，如下图：



成员M1中的api调用创建entry，则发送消息到副本成员中创建新条目，分区区域由M2来分发时间，其他成员M3在通过注册缓存监听器来实时更新。

客户端可以订阅感兴趣或者连续查询的事件，当服务端的事件通过异步队列进行更改时，将接收到更新，事件的分发去下图：



## **3.4 Function的运行**

### 3.4.1 使用

1. 启动Server及创建Region信息

*$ gfsh run --file=scripts/start.gfsh*

*1. Executing - start locator --name=locator --bind-address=127.0.0.1*

*.............*

*Locator in /Users/fys/workspace/geode-dev/geode-examples/functions/locator on 127.0.0.1[10334] as locator is currently online.*

*Successfully connected to: JMX Manager [host=promote.cache-dns.local, port=1099]*

*Cluster configuration service is up and running.*

*2. Executing - start server --name=server1 --locators=127.0.0.1[10334] --server-port=0*

*3. Executing - start server --name=server2 --locators=127.0.0.1[10334] --server-port=0*

*4. Executing - create region --name=example-region --type=REPLICATE*

*Member | Status | Message*

*------- | ------ | ---------------------------------------------*

*server1 | OK | Region "/example-region" created on "server1"*

*server2 | OK | Region "/example-region" created on "server2"*

*Cluster configuration for group 'cluster' is updated.*

*5. Executing - list members*

*Member Count : 3*

*Name | Id*

*------- | -----------------------------------------------------------*

*server1 | 192.168.1.102(server1:1145)<v1>:41000*

*locator | 127.0.0.1(locator:1113:locator)<ec><v0>:41000 [Coordinator]*

*server2 | 192.168.1.102(server2:1172)<v2>:41001*

*6. Executing - describe region --name=example-region*

*Name : example-region*

*Data Policy : replicate*

*Hosting Members : server2*

*server1*

*Non-Default Attributes Shared By Hosting Members*

*Type | Name | Value*

*------ | ----------- | ---------------*

*Region | data-policy | REPLICATE*

*| size | 0*

*| scope | distributed-ack*

*7. Executing - deploy --jar=build/libs/functions.jar*

*Member | Deployed JAR | Deployed JAR Location*

*------- | ------------- | --------------------------------------------------------------------------------*

*server1 | functions.jar | /Users/fys/workspace/geode-dev/geode-examples/functions/server1/functions.v1.jar*

*server2 | functions.jar | /Users/fys/workspace/geode-dev/geode-examples/functions/server2/functions.v1.jar*

*8. Executing - list functions*

*Member | Function*

*------- | -----------*

*server1 | PrimeNumber*

*server2 | PrimeNumber*

1. 定义测试类

*public class Example {*

*private int maximum;*

*public static void main(String[] args) {*

*ClientCache cache = new ClientCacheFactory().addPoolLocator("127.0.0.1", 10334)*

*.set("log-level", "WARN").create();*

*Region<Integer, String> region =*

*cache.<Integer, String>*

*createClientRegionFactory(ClientRegionShortcut.CACHING\_PROXY)*

*.create("example-region");*

*Execution execution = FunctionService.onRegion(region);*

*new Example().getPrimes(region, execution);*

*cache.close();*

*}*

*public Example() {*

*this(100);*

*}*

*public Example(int maximum) {*

*this.maximum = maximum;*

*}*

*public Set<Integer> getPrimes(Region<Integer, String> region, Execution execution) {*

*Set<Integer> primes = new HashSet<>();*

*for (Integer key : (Iterable<Integer>) () -> IntStream.rangeClosed(1, maximum).iterator()) {*

*region.put(key, key.toString());*

*}*

*ResultCollector<Integer, List> results = execution.execute(PrimeNumber.ID);*

*primes.addAll(results.getResult());*

*System.out.println("The primes in the range from 1 to " + maximum + " are:\n" + primes);*

*return primes;*

*}*

*}*

1. 运行

*$ ../gradlew run*

*Starting a Gradle Daemon, 2 incompatible and 1 stopped Daemons could not be reused, use --status for details*

*> Task :functions:run*

*The primes in the range from 1 to 100 are:*

*[[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]]*

*BUILD SUCCESSFUL in 1m 51s*

*7 actionable tasks: 2 executed, 5 up-to-date*

### 3.4.2函数的注册

1）函数的定义如下

*import org.apache.geode.cache.Region;*

*import org.apache.geode.cache.execute.Function;*

*import org.apache.geode.cache.execute.FunctionContext;*

*import org.apache.geode.cache.execute.RegionFunctionContext;*

*public class PrimeNumber implements Function {*

*public static final String ID = PrimeNumber.class.getSimpleName();*

*private boolean isPrime(int number) {*

*int limit = (int) Math.floor(Math.sqrt(number));*

*for (int divisor = 2; divisor <= limit; ++divisor) {*

*if (number % divisor == 0) {*

*return false;*

*}*

*}*

*return true;*

*}*

*@Override*

*public String getId() {*

*return ID;*

*}*

*@Override*

*public void execute(FunctionContext context) {*

*RegionFunctionContext regionContext = (RegionFunctionContext) context;*

*Region<Integer, String> region = regionContext.getDataSet();*

*List<Integer> primes = new ArrayList<>();*

*Set<Integer> keys = region.keySet();*

*for (Integer key : keys) {*

*if (isPrime(key)) {*

*primes.add(key);*

*}*

*}*

*Collections.sort(primes);*

*context.getResultSender().lastResult(primes);*

*}}*

2）执行命令

*deploy --jar=build/libs/functions.jar*

3）函数注册过程，通过JarDeployer实现函数的注册

*public List<DeployedJar> deploy(final Set<File> stagedJarFiles)*

*throws IOException, ClassNotFoundException {*

*List<DeployedJar> deployedJars = new ArrayList<>(stagedJarFiles.size());*

*......*

*lock.lock();*

*try {*

*for (File stagedJarFile : stagedJarFiles) {*

*deployedJars.add(deployWithoutRegistering(stagedJarFile));*

*}*

*return registerNewVersions(deployedJars);*

*} finally {*

*lock.unlock();*

*}*

*}*

执行核心在registerNewVersion中，核心逻辑

*public synchronized void registerFunctions() throws ClassNotFoundException {*

*.....*

*JarInputStream jarInputStream = null;*

*try {*

*// 从Jar包获取Function及FunctionAdapter接口实现类*

*Collection<String> functionClasses = findFunctionsInThisJar();*

*......*

*while (jarEntry != null) {*

*if (jarEntry.getName().endsWith(".class")) {*

*if (functionClasses.contains(className)) {*

*try {*

*Class<?> clazz = ClassPathLoader.getLatest().forName(className);*

*Collection<Function> registerableFunctions = getRegisterableFunctionsFromClass(clazz);*

*for (Function function : registerableFunctions) {*

*FunctionService.registerFunction(function);*

*this.registeredFunctions.add(function);*

*}*

*}*

*.....*

在组件FunctionService中保存Function，并供调用者根据id进行查询

*private static final ConcurrentHashMap<String, Function> idToFunctionMap =  
 new ConcurrentHashMap<>();*

### 3.4.3 函数的执行

执行函数如下：

*ResultCollector<Integer, List> results = execution.execute(PrimeNumber.ID);*

核心逻辑为OpExecutorImpl#execute

*@Override*

*public ResultCollector execute(final String functionName, long timeout, TimeUnit unit) {*

*......*

*Function functionObject = FunctionService.getFunction(functionName);*

*......*

*return executeFunction(functionObject, timeout, unit);*

*}*

*ServerRegionFunctionExecutor#*

*@Override*

*protected ResultCollector executeFunction(final Function function, long timeout, TimeUnit unit) {*

*byte hasResult = 0;*

*try {*

*.....*

*if (function.hasResult()) { // have Results*

*final int timeoutMs = TimeoutHelper.toMillis(timeout, unit);*

*hasResult = 1;*

*if (rc == null) {*

*ResultCollector defaultCollector = new DefaultResultCollector();*

*return executeOnServer(function, defaultCollector, hasResult, timeoutMs);*

*} else { // Custome Result COllector*

*return executeOnServer(function, rc, hasResult, timeoutMs);*

*}*

*} .....*

*}*

# **参考链接**

Apache Geode简介：

https://wjw465150.github.io/GeodeUserGuide/Geode\_2\_Configuring\_and\_Running\_a\_Cluster.html

Apache Geode使用：

https://www.tutorialandexample.com/apache-geode-tutorial/

https://geode.apache.org/docs/guide/12/developing/function\_exec/function\_execution.html

https://github.com/apache/geode-examples

Apache Geode源码解析：

https://www.infoq.cn/article/internal-structure-of-apache-geode-distributed-system

https://blog.csdn.net/karamos/article/details/80127929