An Simple Cache System With Akka Cluster

For Scala Meetup in Shenzhen 2018.09.02

凤凰木 <weiwen@weiwen.org>

2018.09.02

Content

业务需求

业务场景 1

- 一堆复杂的配置信息，存在 DB 里
- 配置不定期更新
- 一组应用服务器，需要实时读取这些配置信息

业务场景 2

- 若干张基础数据表，存在 DB 里
- 数据不定期更新
- 一组应用服务器，需要实时读取这些基础数据

业务场景 3

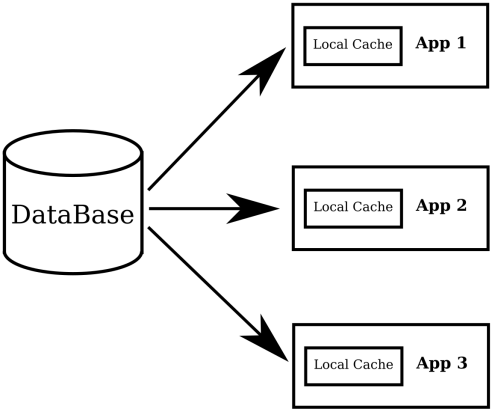
- 若干核心业务逻辑的报表，存在 DB 里
- 报表不断更新
- 一组应用服务器，需要实时读取这些报表
- 从 DB 读取报表的代价比较大，比如需要聚合

业务需求

1. 一组高性能的服务，$2 \leq instances \leq 100$
2. 高速读取来自各种数据源的数据
3. 读取压力可能大到不允许有网络 IO
4. 数据不断被更新
5. 读取的代价可能比较大，尤忌并发读取
6. 数据不要求强一致性
7. 数据量不是特别巨大，远小于 JVM Heap

架构设计

最早的做法

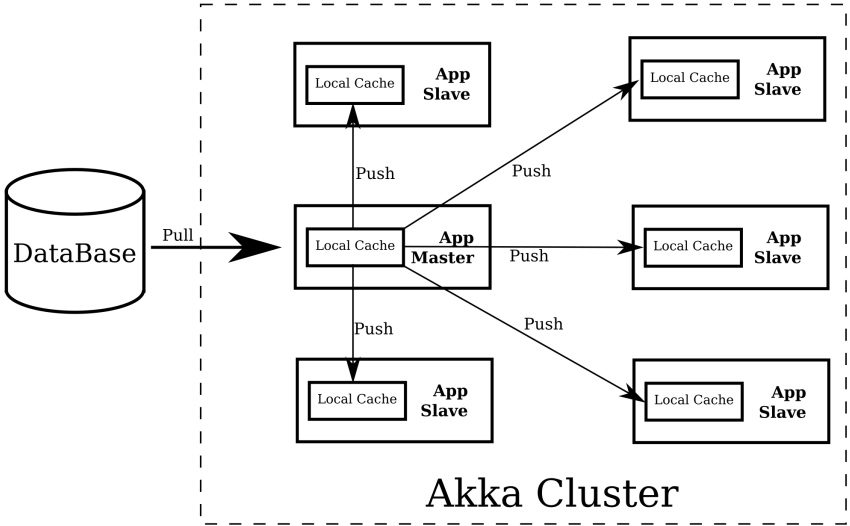

DataBase
Local Cache
App 1
Local Cache
App 2
Local Cache
App 3

最早的做法

弊端：

- 数据源压力线性增长

改进的方案


Local Cache
App
Slave
Local Cache
App
Slave
Push
Push
DataBase
Pull
Local Cache
App
Master
Push
Local Cache
App
Slave
Push
Push
Local Cache
App
Slave
Local Cache
App
Slave
Akka Cluster

各组件要点

- 缓存作为一个子系统，嵌入应用内
- 使用 Akka Cluster 来组成集群
- 使用 Akka Cluster Singleton Actor 作为 Master Node
- Akka Distributed Publish Subscribe 实现缓存的获取和分发逻辑
- Protobuf 作为数据传输格式

实现

Data Structure: Protobuf

file src/main/protobuf/cache_meta.proto:

```
1  syntax = "proto3";
2
3  package simple.idl.cache;
4
5  enum CacheCate {
6      EmptyT   = 0;
7      SimpleT  = 1;
8      ComplexT = 2;
9  }
10
11 message PackedCachePair {
12     bytes key   = 1;
13     bytes value = 2;
14 }
15
16 message PackedCache {
17     int64 version               = 1;
18     repeated PackedCachePair cc = 2;
19 }
```

Data Structure: Protobuf, Why?

Questions:

1. Why **`Protobuf`** ?
2. Why **`PackedCachePair`** and **`PackedCache`** ?

Data Structure: concurrent.TrieMap

```
1 scala.collection.concurrent.TrieMap
```

- A concurrent hash-trie or TrieMap is a concurrent thread-safe lock-free implementation of a hash array mapped trie.
- It has particularly scalable concurrent insert and remove operations and is memory-efficient.
- It supports O(1), atomic, lock-free snapshots which are used to implement linearizable lock-free size, iterator and clear operations.
- The cost of evaluating the (lazy) snapshot is distributed across subsequent updates, thus making snapshot evaluation horizontally scalable.

Data Structure: TrieMapCache

```scala
import scala.collection.concurrent.TrieMap
import simple.idl.cache.CacheCate

trait TrieMapCache[K, V] {

  def underlying: TrieMap[K, V]

  def cate: CacheCate

  var version: Long = 0L
  // var version: java.util.concurrent.atomic.AtomicLong = ???

}
```

Data Structure: TrieMapCacheOps, 1

```
1  trait TrieMapCacheOps[K, V] {
2
3    this: TrieMapCache[K, V] ⇒
4
5    def setVersion(v: Long): Unit = {
6      this.version = v
7    }
8
9    def incVersion(): Long = {
10     this.version += 1L
11     this.version
12   }
13
14 }
```

Data Structure: TrieMapCacheOps, 2

```scala
trait TrieMapCacheOps[K, V] {

  this: TrieMapCache[K, V] =>

  // 按 key 获取
  def get(key: K, default: K => V): V = underlying.getOrElse(key, default(key))

  // 按 key 获取
  def getIfPresent(key: K): Option[V] = underlying.get(key)

  // 按 key 更新
  def put(key: K, value: V): Option[V] = underlying.put(key, value)

  // 批量更新
  def putAll(m: Iterable[(K, V)]): TrieMap[K, V] = underlying ++= m

  // 批量逐出, 按 keys
  def invalidateAll(keys: Iterable[K]): TrieMap[K, V] = underlying --= keys

}
```

Data Structure: TrieMapCacheOps, 3

```scala
trait TrieMapCacheOps[K, V] {

  this: TrieMapCache[K, V] ⇒

  // 全量更新
  def updateAll(m: Iterable[(K, V)]): Unit = {
    val toDrops = underlying.keys.toVector.diff(m.map(_._1).toVector)
    invalidateAll(toDrops)
    putAll(m)
  }

  def updateAllWithVersion(v: Long, m: Iterable[(K, V)]): Unit = {
    updateAll(m)
    setVersion(v)
  }

  def updateAllAndIncVersion(m: Iterable[(K, V)]): Unit = {
    updateAll(m)
    this.incVersion()
  }

}
```

Data Structure: InMemoryCache

```
1  trait InMemoryCache[K, V] extends
2    TrieMapCache[K, V] with TrieMapCacheOps[K, V]
```

Data Structure: PackableInMemoryCache, 1

```scala
import com.google.protobuf.ByteString
import simple.idl.cache.{PackedCachePair, PackedCache}

trait PackableInMemoryCache[K, V] extends InMemoryCache[K, V] {

  val underlying: TrieMap[K, V] = TrieMap.empty[K, V]

  def packKey(key: K): ByteString

  def packValue(value: V): ByteString

  def unPackKey(bytes: ByteString): K

  def unPackValue(bytes: ByteString): V

}
```

Data Structure: PackableInMemoryCache, 2

```scala
import simple.idl.cache.{PackedCachePair, PackedCache}

trait PackableInMemoryCache[K, V] extends InMemoryCache[K, V] {

  def pack: PackedCache = {
    val ps = underlying.toList.map { case (k, v) ⇒
      PackedCachePair(packKey(k), packValue(v))
    }
    PackedCache(valid = 1, version = version, cc = ps)
  }

  def unPack(g: PackedCache): TrieMap[K, V] = {
    val xs = g.cc.map { case PackedCachePair(k, v) ⇒
      (unPackKey(k), unPackValue(v))
    }
    TrieMap(xs: _*)
  }

}
```

Data Structure: PbInMemoryCache

```scala
import com.google.protobuf.ByteString
import scalapb.GeneratedMessage

trait PbInMemoryCache[K <: GeneratedMessage, V <: GeneratedMessage]
  extends PackableInMemoryCache[K, V] {

  override def packKey(key: K): ByteString = key.toByteString

  override def packValue(value: V): ByteString = value.toByteString

}
```

SimpleCache: protobuf

file src/main/protobuf/simple_cache.proto:

```
1  syntax = "proto3";
2
3  package simple.idl.cache;
4
5  message SimpleCacheKey {
6      int64 id    = 1;
7      int32 subId = 2;
8  }
9
10 message SimpleCacheValue {
11     string name        = 1;
12     int64  impressions = 2;
13     int64  clicks      = 3;
14 }
```

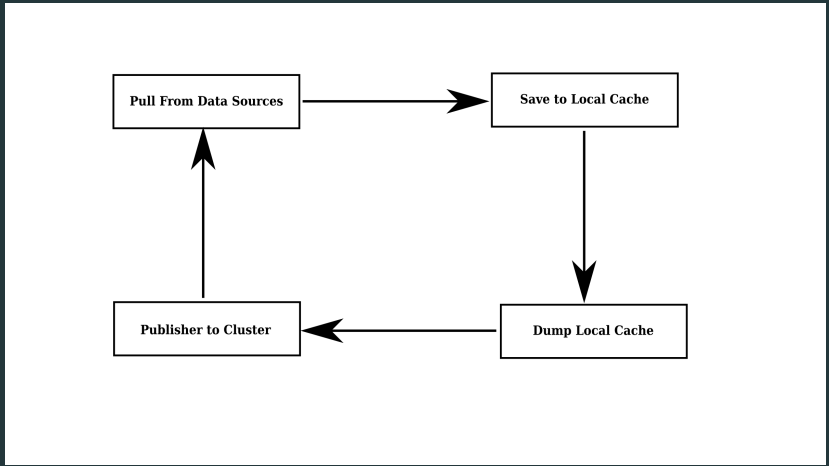
SimpleCache: scala, 1

```scala
import com.chinamobiad.adx.idl.cache.CacheCate
import com.google.protobuf.ByteString
import simple.idl.cache.{SimpleCacheKey, SimpleCacheValue}

object SimpleCache {

  type K = SimpleCacheKey

  type V = SimpleCacheValue

}
```

SimpleCache: scala, 2

```scala
import com.chinamobiad.adx.idl.cache.CacheCate
import com.google.protobuf.ByteString
import simple.idl.cache.{SimpleCacheKey, SimpleCacheValue}

trait SimpleCache {

  import SimpleCache._

  val simpleCache: PbInMemoryCache[K, V] =

    new PbInMemoryCache[SimpleCache.K, SimpleCache.V] {

    val cate = CacheCate.SimpleT

    def unPackKey(bytes: ByteString): K =
      SimpleCacheKey.parseFrom(bytes.toByteArray)

    def unPackValue(bytes: ByteString): V =
      SimpleCacheValue.parseFrom(bytes.toByteArray)
  }

}
```

Master Actor Flow



Configuration: build.sbt

file build.sbt:

```
libraryDependencies ++= Seq(
    "com.typesafe.akka"    %% "akka-stream"          % akkaV
  , "com.typesafe.akka"    %% "akka-cluster"         % akkaV
  , "com.typesafe.akka"    %% "akka-cluster-tools"   % akkaV
  , "com.twitter"          %% "chill-akka"           % "0.9.3"
)
```

Configuration: application.conf, 1

file src/main/resources/application.conf:

```
akka {
  actor {
    provider = cluster
    allow-java-serialization = off
    serializers {
      java = "akka.serialization.JavaSerializer"
      kryo = "com.twitter.chill.akka.AkkaSerializer"
      proto = "akka.remote.serialization.ProtobufSerializer"
    }
    serialization-bindings {
      "scalapb.GeneratedMessage" = proto
      "scalapb.GeneratedEnum" = proto
      "java.io.Serializable" = kryo
    }
  }
}
```

Configuration: application.conf, 2

file src/main/resources/application.conf:

```
1  akka {
2    cluster {
3      seed-nodes = [
4        "akka.tcp://simple@127.0.0.1:25510"
5        , "akka.tcp://simple@127.0.0.1:25520"
6        , "akka.tcp://simple@127.0.0.1:25530"
7      ]
8    }
9    remote {
10     enabled-transports = ["akka.remote.netty.tcp"]
11     log-remote-lifecycle-events = off
12     netty.tcp.port = 25510
13   }
14 }
```

Reference: akka cluster config and serialization

Akka Cluster:

- https://doc.akka.io/docs/akka/current/cluster-usage.html

Serialization:

- https://doc.akka.io/docs/akka/current/serialization.html?language=scala

PubSubCacheManagerT

```scala
import com.typesafe.scalalogging.Logger
import simple.idl.cache._

object PubSubCacheManager {

  def masterNameOf(cate: CacheCate): String =
    s"CacheMaster${cate.name}"

  def topicNameOf(cate: CacheCate): String =
    s"PubSubAcorTopic${cate.name}"

}

trait PubSubCacheManagerT[K, V] {

  def cache: InMemoryCache[K, V]

  protected val logger: Logger = Logger(getClass)

}
```

PubSubCacheMaster, 1

```scala
trait PubSubCacheMaster[K, V] {

  this: PubSubCacheManagerT[K, V] ⇒

  import PubSubCacheManager._

  // 每轮循环的间隔
  val tickInterval: FiniteDuration = 1000.milliseconds

  // 把本地缓存 dump 成 protobuf 消息
  def dumpCache: scalapb.GeneratedMessage

  // 从数据源获取数据，更新本地缓存
  def updateMasterCache()(implicit ec: ExecutionContext): Future[_]

}
```

PubSubCacheMaster, 2

```scala
import scala.concurrent.{ExecutionContext, Future}
import akka.actor._
import akka.stream._
import akka.stream.scaladsl._
import akka.cluster.pubsub.DistributedPubSub
import akka.cluster.pubsub.DistributedPubSubMediator._

trait PubSubCacheMaster[K, V] {

  this: PubSubCacheManagerT[K, V] ⇒

  import PubSubCacheManager._

  def flow()(implicit system: ActorSystem
             , ec: ExecutionContext): Flow[Unit, Any, NotUsed] = {
    val publisher: ActorRef = DistributedPubSub(system).mediator
    Flow[Unit]
      .mapAsync(1)(_ ⇒ updateMasterCache())
      .map(_ ⇒ cache.incVersion())
      .map { _ ⇒
        publisher ! Publish(topicNameOf(cache.cate), dumpCache)
      }
  }
```

Reference: Distributed Publish Subscribe in Cluster

Distributed Publish Subscribe in Cluster:

- https://doc.akka.io/docs/akka/current/distributed-pub-sub.html

PubSubCacheMaster, 3

```scala
trait PubSubCacheMaster[K, V] {
  this: PubSubCacheManagerT[K, V] ⇒
  def killSwitch()(implicit system: ActorSystem
                   , mat: Materializer
                   , ec: ExecutionContext): UniqueKillSwitch = {
    RestartSource.withBackoff(minBackoff = 1.seconds
      , maxBackoff = 60.seconds, randomFactor = 0.2) { () ⇒
      Source.fromFuture {
        Source
          .tick(0.seconds, tickInterval, ())
          .via(flow)
          .withAttributes(ActorAttributes.supervisionStrategy({
            case NonFatal(e) ⇒
              e.printStackTrace()
              Supervision.Resume
          }))
          .runWith(Sink.ignore)
      }
    }
      .viaMat(KillSwitches.single)(Keep.right)
      .toMat(Sink.ignore)(Keep.left)
      .run()
  }
```

Reference: RestartSource in Akka Stream

RestartSource:

- https://doc.akka.io/docs/akka/current/stream/stream-error.html

PubSubCacheMaster, 4

```
1  trait PubSubCacheMaster[K, V] {
2
3    this: PubSubCacheManagerT[K, V] ⇒
4
5    private[this] class MasterActor extends Actor {
6
7      implicit val mat: ActorMaterializer = ActorMaterializer()
8
9      // 立即启动 Master 工作流
10     val ks: UniqueKillSwitch =
11       killSwitch()(context.system, mat, context.dispatcher)
12
13     override def postStop(): Unit = {
14       ks.shutdown()
15       mat.shutdown()
16       super.postStop()
17     }
18
19     def receive: Receive = PartialFunction.empty[Any, Unit]
20
21   }
22 }
```

PubSubCacheMaster, 5

```
1  trait PubSubCacheMaster[K, V] {
2
3    this: PubSubCacheManagerT[K, V] ⇒
4
5    def clusterSingleton()(implicit system: ActorSystem): ActorRef =
6      system.actorOf(
7        ClusterSingletonManager.props(
8          singletonProps = Props(new MasterActor)
9          , terminationMessage = 0
10         , settings = ClusterSingletonManagerSettings(system)
11       )
12       , name = masterNameOf(cache.cate)
13     )
14
15 }
```

Reference: Cluster Singleton

Cluster Singleton:

- https://doc.akka.io/docs/akka/current/cluster-singleton.html

PubSubCacheClient

```scala
trait PubSubCacheClient[K, V] {

  this: PubSubCacheManagerT[K, V] ⇒

  import PubSubCacheManager._

  def clientReceive: PartialFunction[Any, Unit]

  def startClient()(implicit system: ActorSystem): ActorRef =
    system.actorOf(Props(new ClientActor))

  class ClientActor extends Actor {
    override def receive: Receive = clientReceive
    override def preStart(): Unit = {
      val mediator = DistributedPubSub(context.system).mediator
      // 订阅消息
      mediator ! Subscribe(topicNameOf(cache.cate), self)
    }
  }

}
```

SimplePbPubSubCacheManager

```scala
trait SimplePbPubSubCacheManager[K, V]
  extends PubSubCacheManagerT[K, V]
    with PubSubCacheMaster[K, V]
    with PubSubCacheClient[K, V] {

  override def cache: PackableInMemoryCache[K, V]

  override def dumpCache: GeneratedMessage = cache.pack

  def currentVersion: Long = cache.version

  override def clientReceive: PartialFunction[Any, Unit] = {
    // 需要检查版本号
    case x@PackedCache(1, v, _) if v > cache.version ⇒
      cache.updateAllWithVersion(v, cache.unPack(x))
  }

  def start()(implicit system: ActorSystem): Unit = {
    startClient()
    val _ = clusterSingleton()
  }

}
```

SimpleCacheManager

```scala
import scala.concurrent.{ExecutionContext, Future}
import simple.idl.cache.{SimpleCacheKey, SimpleCacheValue}

class SimpleCacheManager(
  val cache: PbInMemoryCache[SimpleCacheKey, SimpleCacheValue]
                          ) extends
  SimplePbPubSubCacheManager[SimpleCacheKey, SimpleCacheValue] {

  val dao: SimpleCacheDAO = ???

  override def updateMasterCache()(
    implicit ec: ExecutionContext
  ): Future[Unit] = {
    val mF: Future[List[(SimpleCacheKey, SimpleCacheValue)]] =
      dao.queryAll()
    mF.map(cache.updateAllAndIncVersion)
  }
}
```

Cache

```
object Cache extends SimpleCache
  with ComplexCache
  with WxxCache
  with XxxCache
  with YyyCache
  with ZzzCache
```

SimpleService

```scala
object SimpleService {

  // 如果找不到，就返回默认值 0L
  def findClicks(id: Long, subId: Int): Long =
    Cache.simpleCache
         .get(SimpleCacheKey(id, subId))
         .clicks

}
```

CacheManager

```
1  object CacheManager {
2
3    def start()(implicit system: ActorSystem
4                , ec: ExecutionContext) = {
5      new SimpleCacheManager(Cache.simpleCache).start()
6      new ComplexCacheManager(Cache.complexCache).start()
7       ...
8    }
9
10 }
```

Main

```scala
import akka.actor.ActorSystem
import akka.stream.ActorMaterializer
import scala.concurrent.ExecutionContext.Implicits.global

object Main extends App {

  implicit val system: ActorSystem = ActorSystem()

  CacheManager.start()

}
```

问题与总结

Conclusion

- 弹性好, 可以很容易拓展
- 可靠性高, 即使集群脑裂也暂时无事
- 缓存没有过期机制, 但可以加
- 容量有限
- 总是全量更新，有利有弊，可以改进

完

谢谢大家！