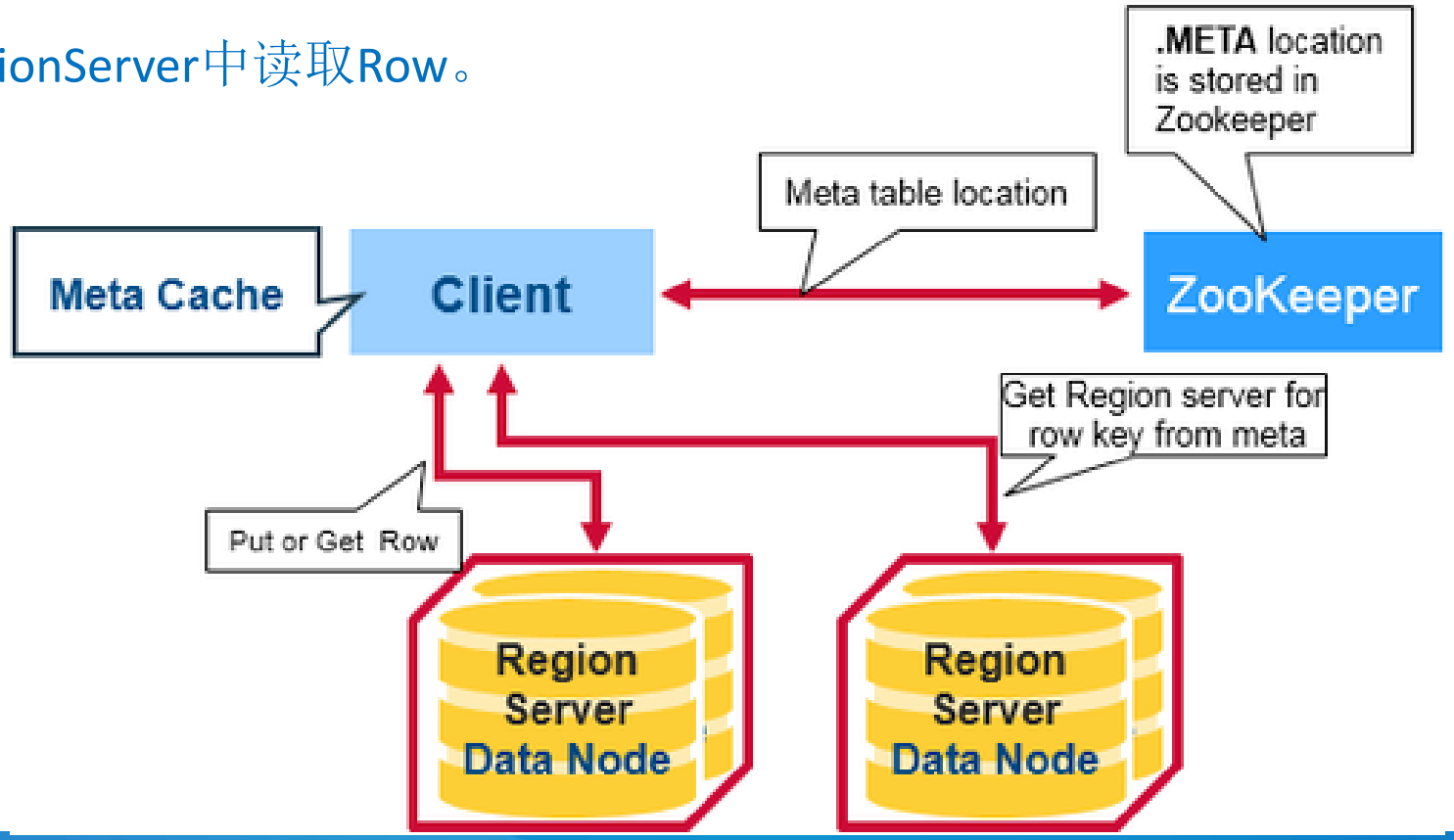
分布式面向列的数据库HBase

Northeastern University
HORIZON 昊宸科技

目录

HBase读写数据流程-第一次读数据

1. 从ZooKeeper(/hbase/meta-region-server)中获取hbase:meta的位置（HRegionServer的位置），缓存该位置信息。
2. 从hbase:meta中查询用户Table对应请求的RowKey所在的HRegionServer，缓存该位置信息。
3. 从查询到HRegionServer中读取Row。



HBase读写数据流程-hbase:meta表

Rowkey:tableName,regionStartKey,regionId,replicaId等

info:regioninfo列是RegionInfo的proto格式： regionId,tableName,startKey,endKey,offline,split,replicaId；

info:server格式： HRegionServer对应的server:port；

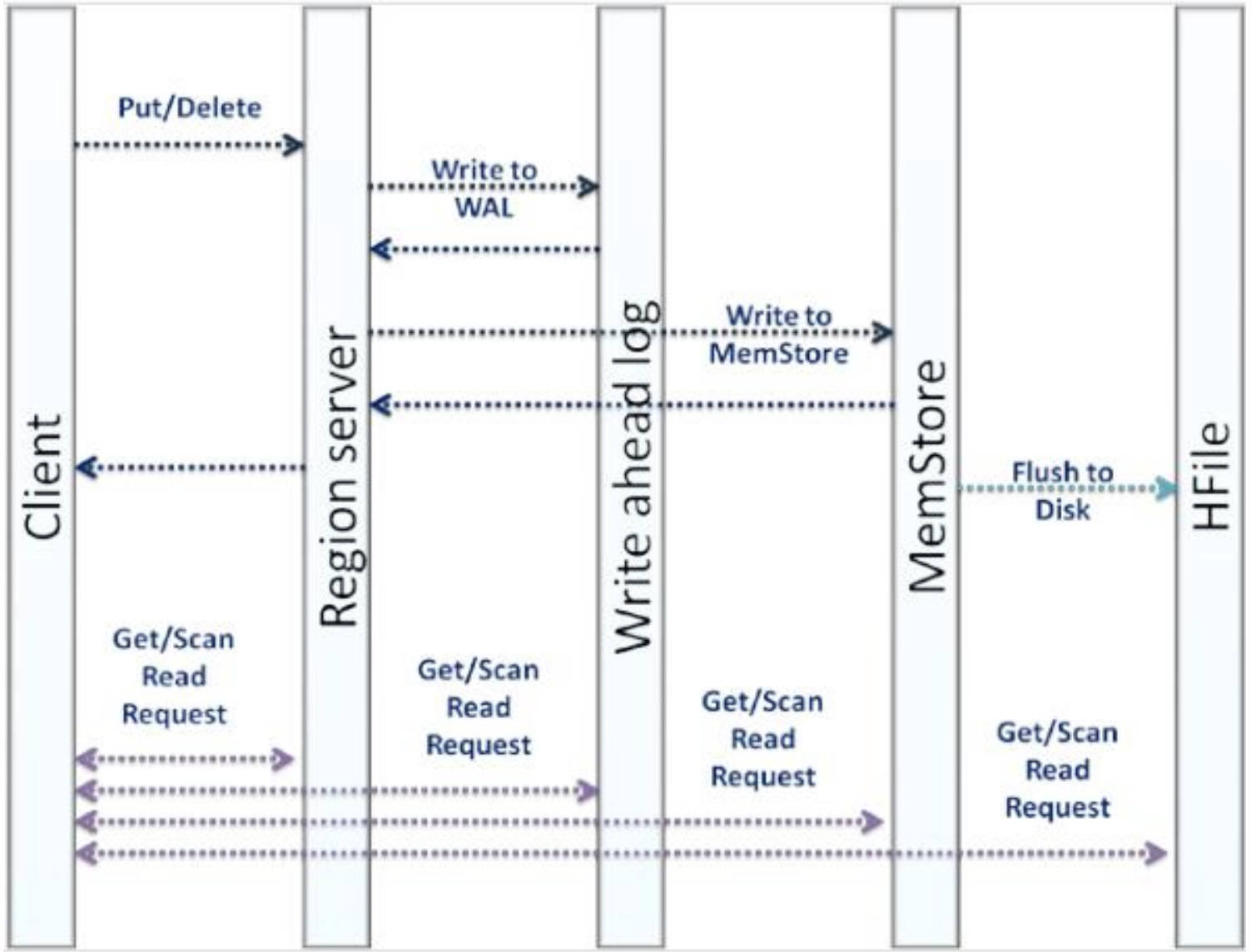
info:serverstartcode格式是HRegionServer的启动时间戳。

例如： scan 'hbase:meta',{FILTER=>"(PrefixFilter('tsdb'))"}

```
tsdb,\x10,1487582469433.5ceebc196 column=info:regioninfo, timestamp=1487582470358, value={ENCODED => 5ceebc1963fa88f262e2cc80d89b88b
3fa88f262e2cc80d89b88ba.          a, NAME => 'tsdb,\x10,1487582469433.5ceebc1963fa88f262e2cc80d89b88ba.', STARTKEY => '\x10', ENDKEY
                                   => '\x11'}
tsdb,\x10,1487582469433.5ceebc196 column=info:seqnumDuringOpen, timestamp=1493774852528, value=\x00\x00\x00\x00\x00\x00\x1Da
3fa88f262e2cc80d89b88ba.
tsdb,\x10,1487582469433.5ceebc196 column=info:server, timestamp=1493774852528, value=idh103:16020
3fa88f262e2cc80d89b88ba.
tsdb,\x10,1487582469433.5ceebc196 column=info:serverstartcode, timestamp=1493774852528, value=1493774819191
```

HBase读写数据流程-写数据流程

1. 先写到hlog
2. 向memstore写
3. memstore到达一定阈值后，在向hfile写。



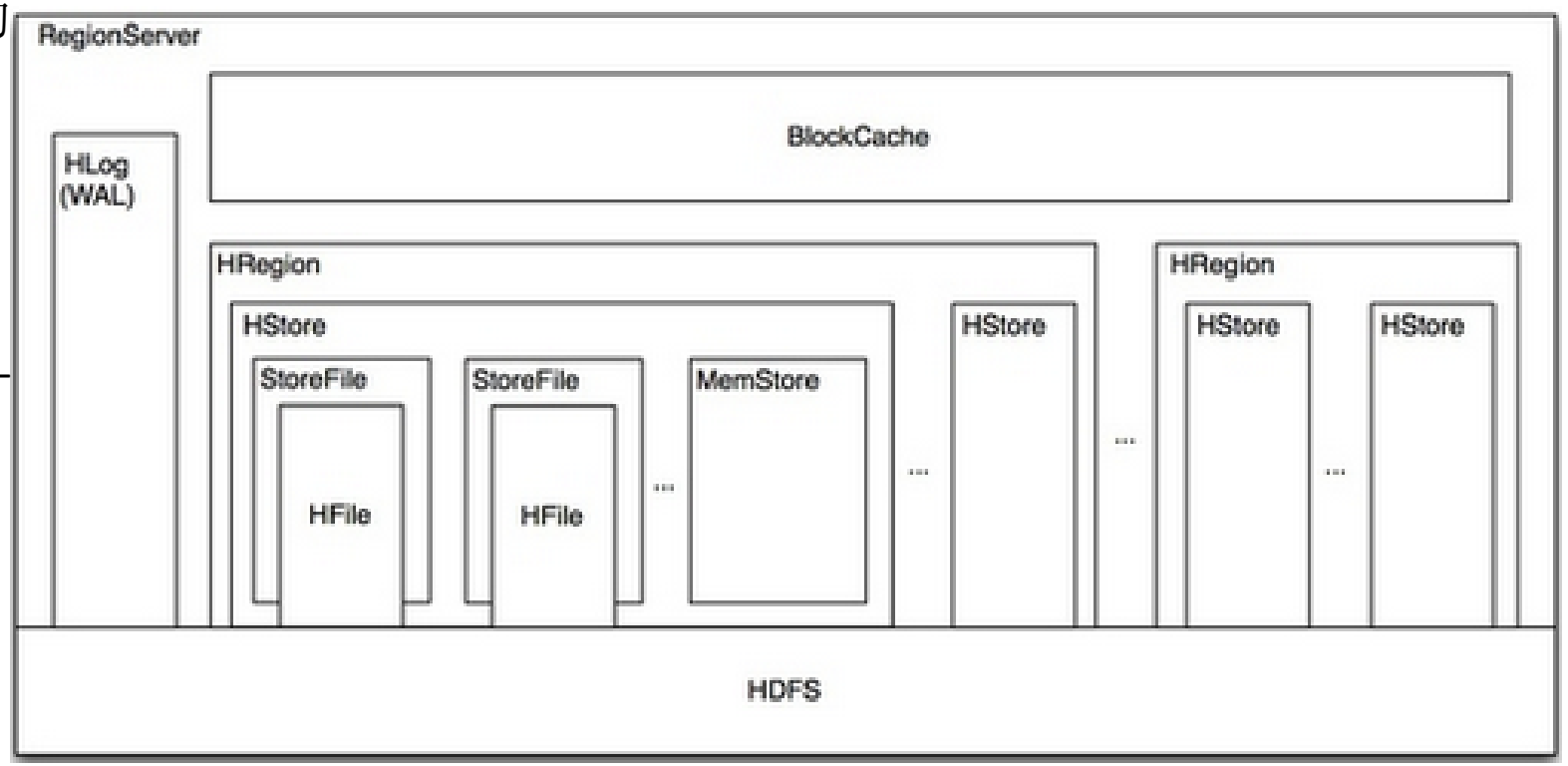
HBase读写数据流程-写数据流程

回顾一下HRegion。

Region是表按照RowKey范围划分的不同的部分，相当于DBMS中的分区。

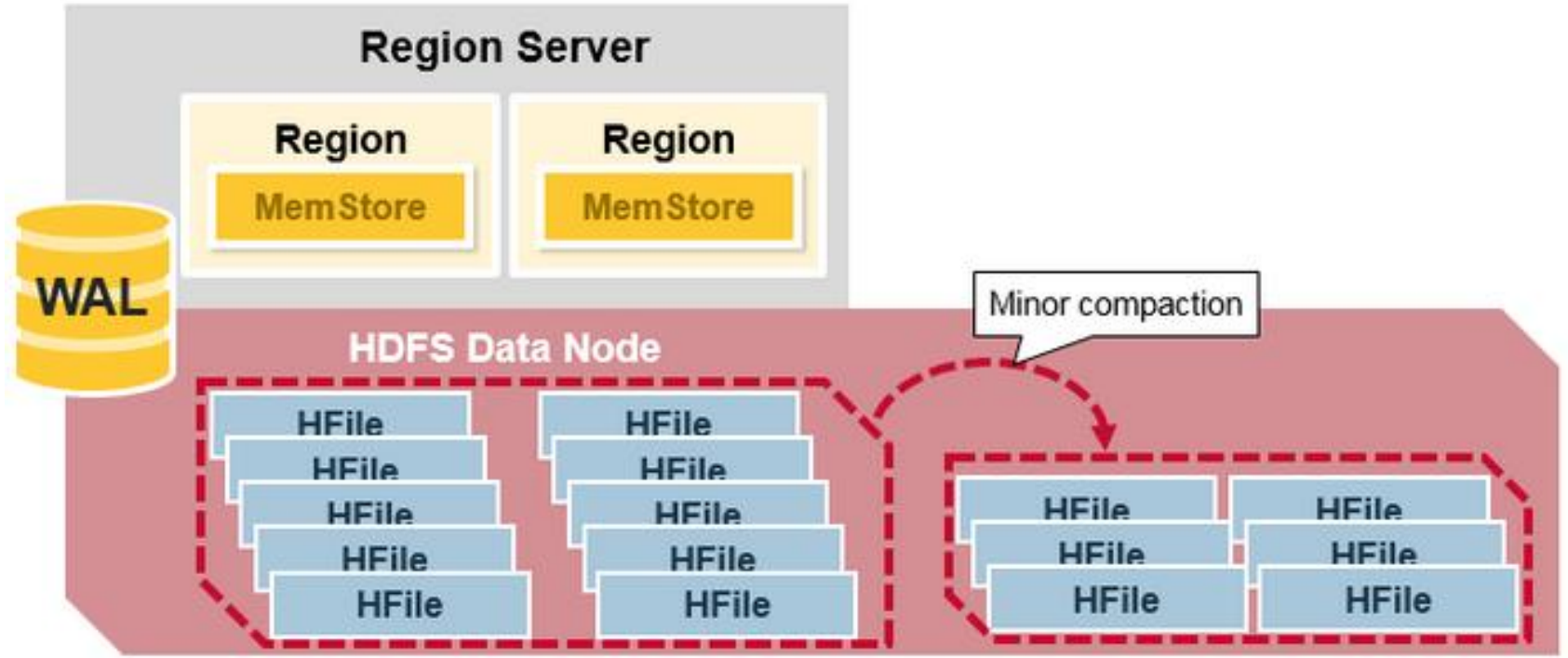
同时Region也是表在集群中分布的最小单位，可以被分配到某一个Region Server上。

- Region中又按照column Family分为不同的Hstore。每个Hstore又由memStore和StoreFile组成。
- 一个HStore------> 对应一个memstore----> 数据不断从memstore写入硬盘，造成了慢慢生成多个storeFile------>一个storefile ----> 对应一个hfile。



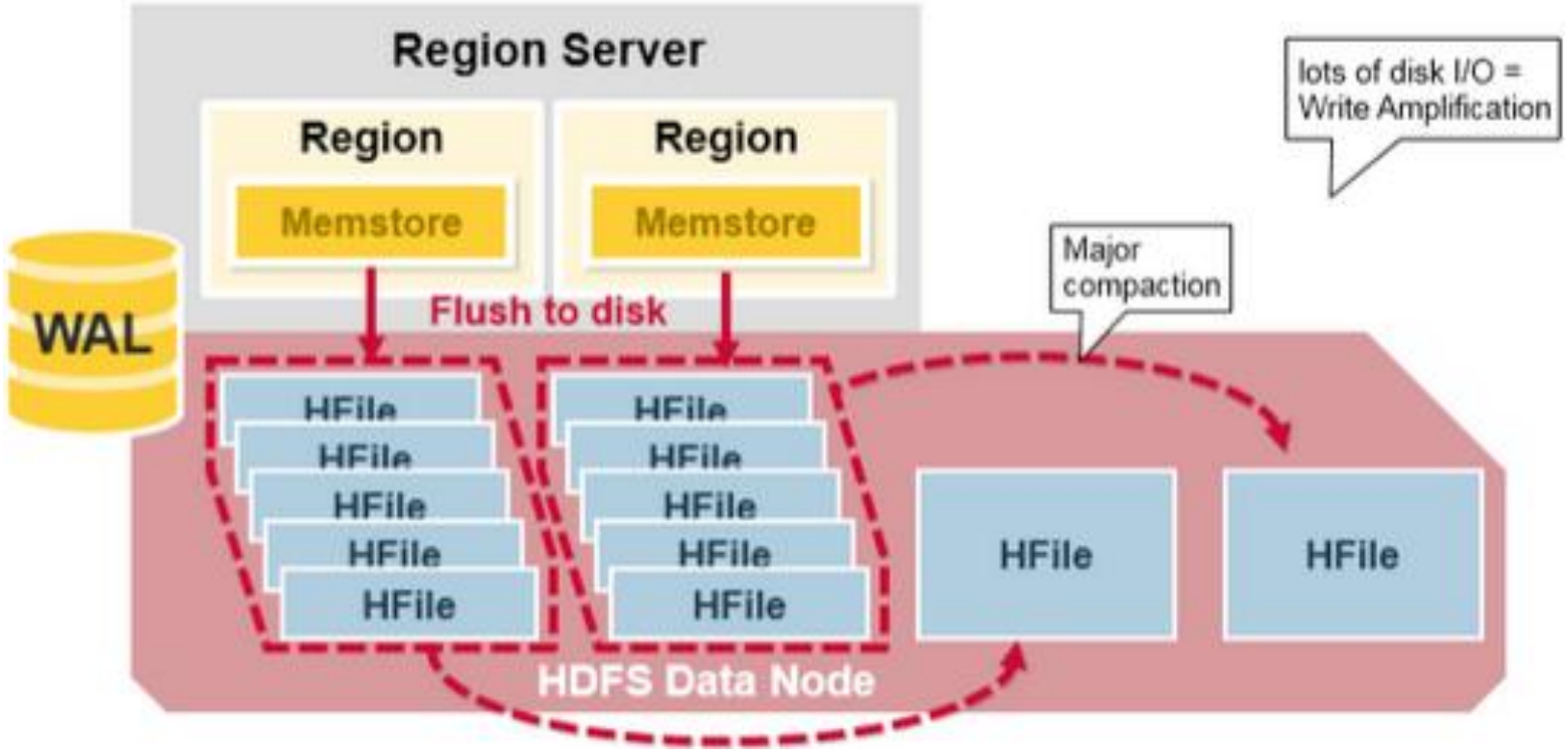
HBase MINOR-COMPACTION

HBase会自动拾取一些较小的HFiles，并将它们重新写入一些较大的HFiles中。
Minor compaction不会处理已经Deleted或Expired的Cell。



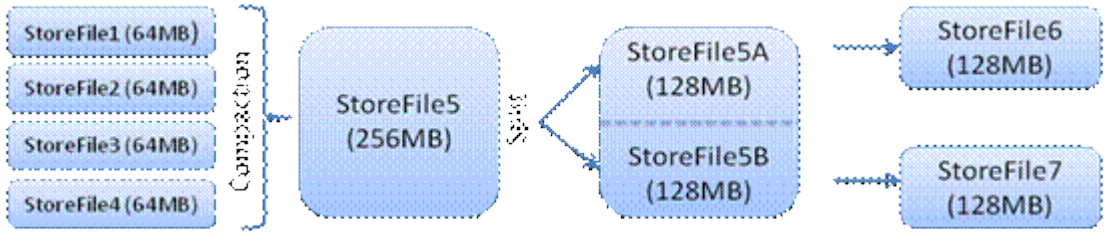
HBase MAJOR-COMPACTION

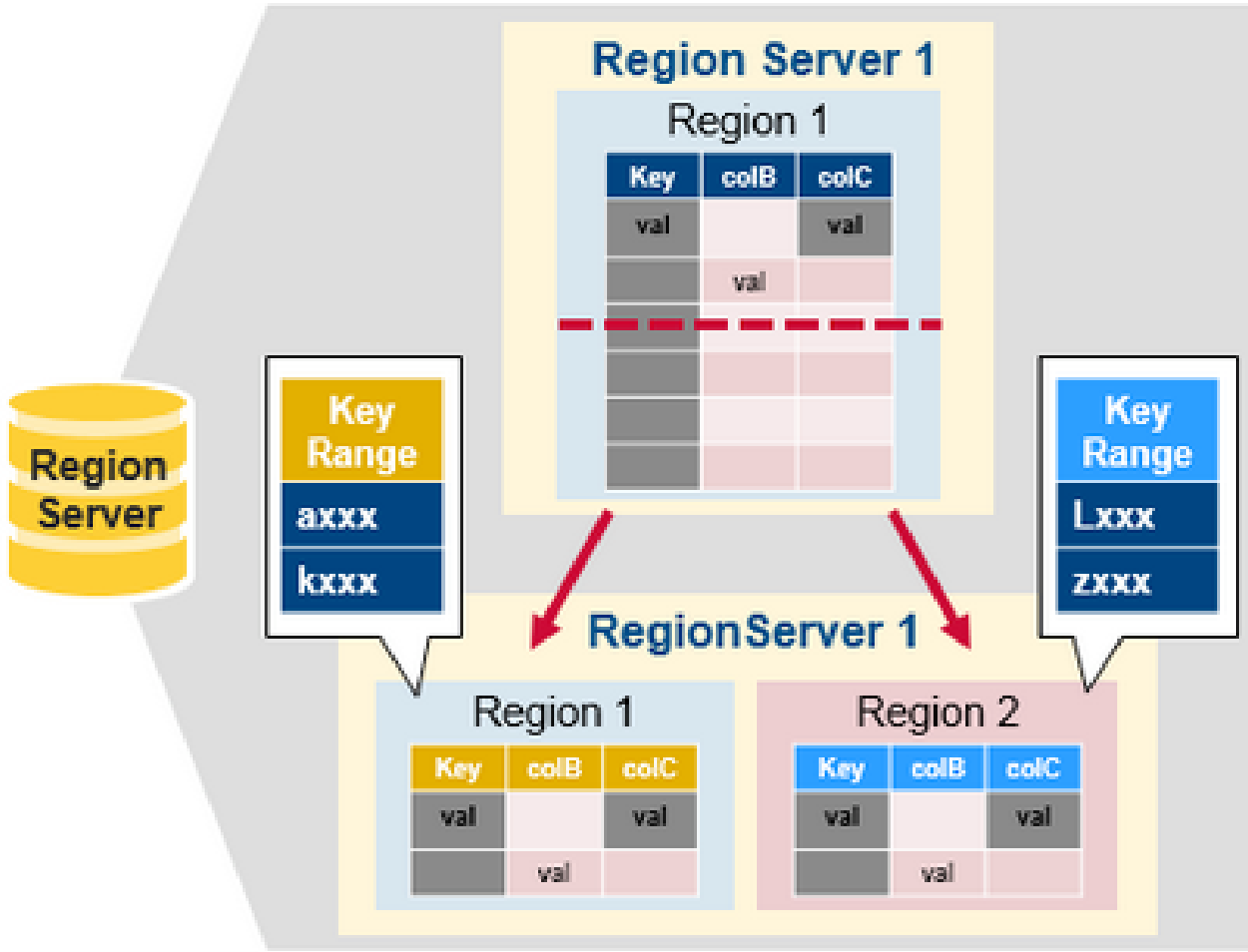
- 将Region上一个列族对应的多个Hfile合并为一个大的Hfile文件。
- 删除过期数据，被删除数据。
- 改善Hbase读取效率。
- 会引起磁盘IO和网络资源的紧缺。
- 可以被设置为周期执行。



HBase Region split

- Region的大小超过了hbase.hregion.max设置的值之后，region将被拆分。
- Memstore flush时会触发region的拆分。
- 可以强制拆分。
- Region split过程。
  - ✓ Region下线。
  - ✓ Region拆分。
  - ✓ 更新meta表。
  - ✓ 汇报给master。
  - ✓ 子region分配到regionServer上。



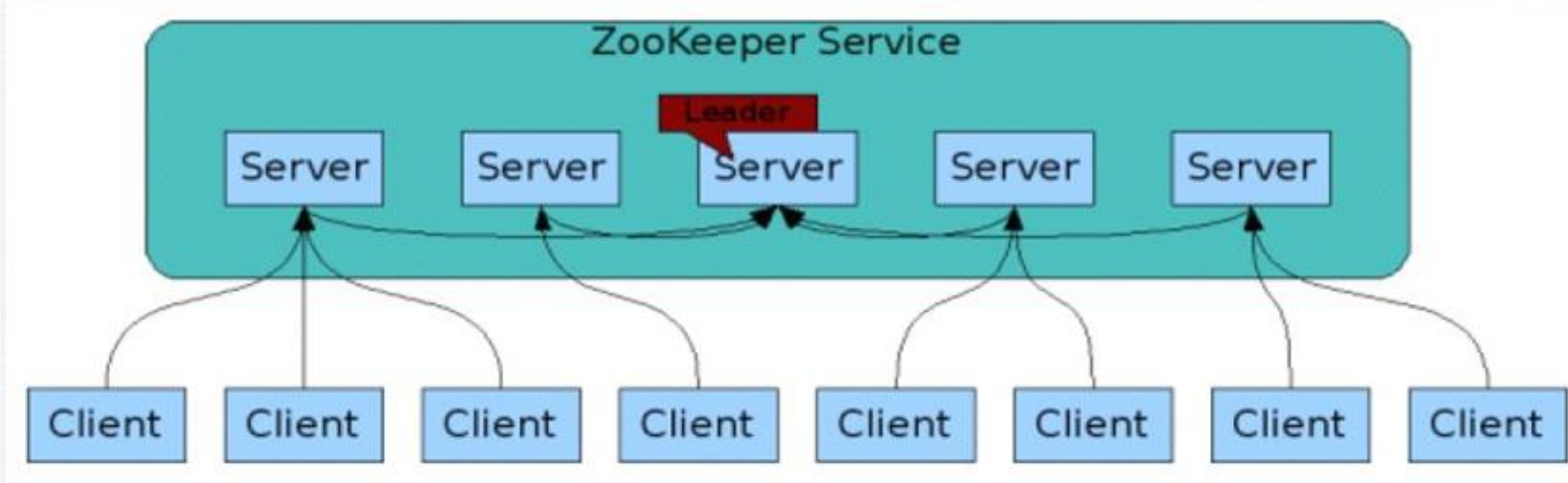


HBase java API

Zookeeper简介

- Zookeeper是一个开源的、分布式的应用程序协调服务。它提供了一套原语集，通过这套原语集，可以实现更高层次的同步服务、配置管理、集群管理以及命名管理。
- **一句话：**Zookeeper就是保证数据在集群中的事务一致性。
  - ✓ zk是集群部署的。
  - ✓ 集群之间是数据传递的。
  - ✓ 集群之间传递数据必须要保证事务的一致性。

Zookeeper架构

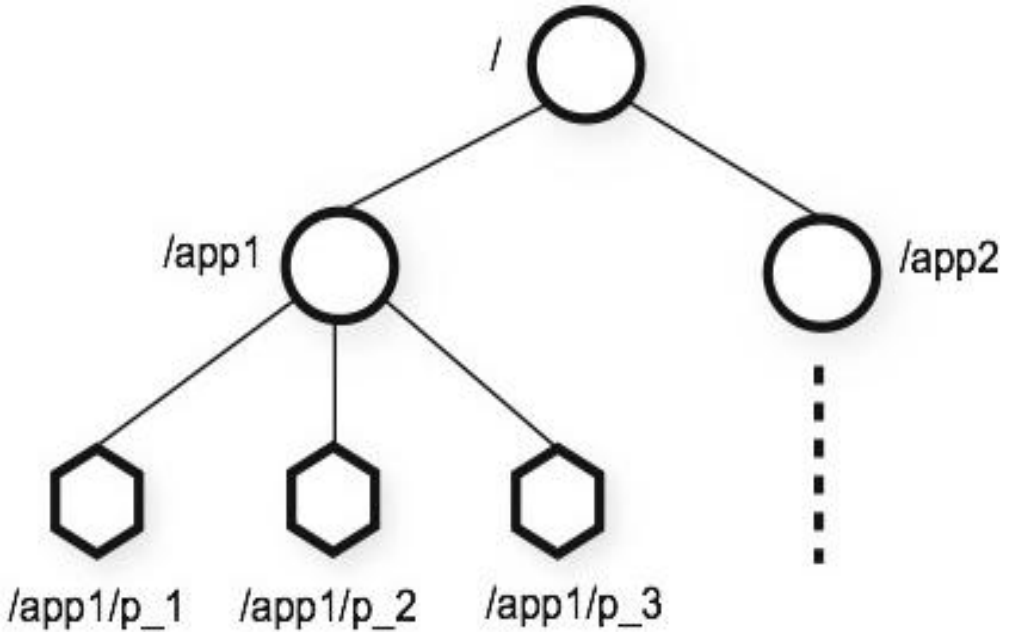


Zookeeper架构

角色		描述
领导者（Leader）		领导者负责进行投票的发起和决议，更新系统状态
学习者（Learner）	跟随者（Follower）	Follower 用于接收客户请求并向客户端返回结果，在选主过程中参与投票
	观察者（ObServer）	ObServer 可以接收客户端连接，将写请求转发给 leader 节点。但 ObServer 不参加投票过程，只同步 leader 的状态。ObServer 的目的是为了扩展系统，提高读取速度
客户端（Client）		请求发起方

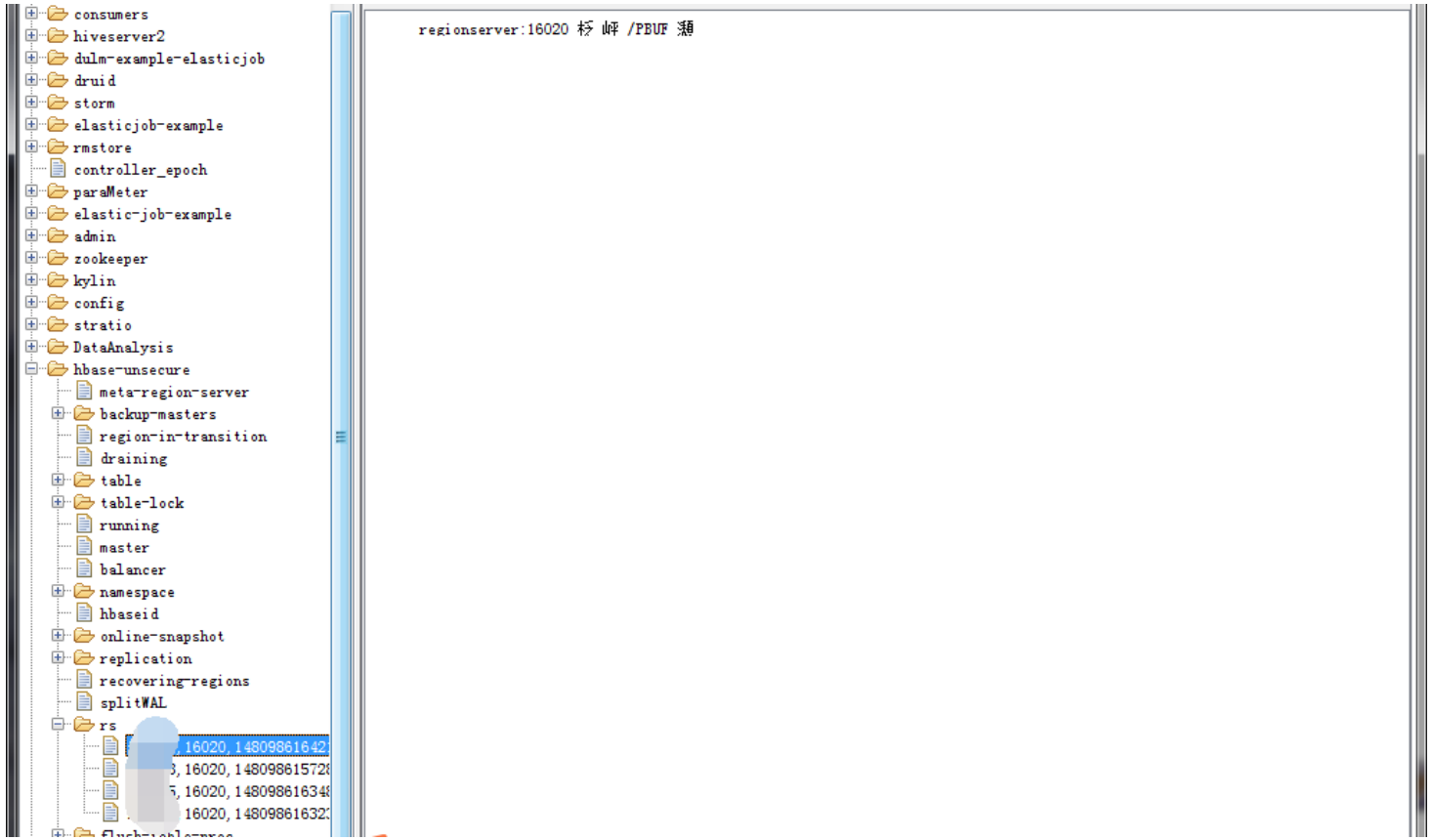
Zookeeper的数据模型

- 每个子目录项如p_1都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 p_1 这个 znode 的标识为 /app1/p_1。
- znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录,例如：p_1
- znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据
- znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了。
- znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2
- znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的，后面在典型的应用场景中会有实例介绍



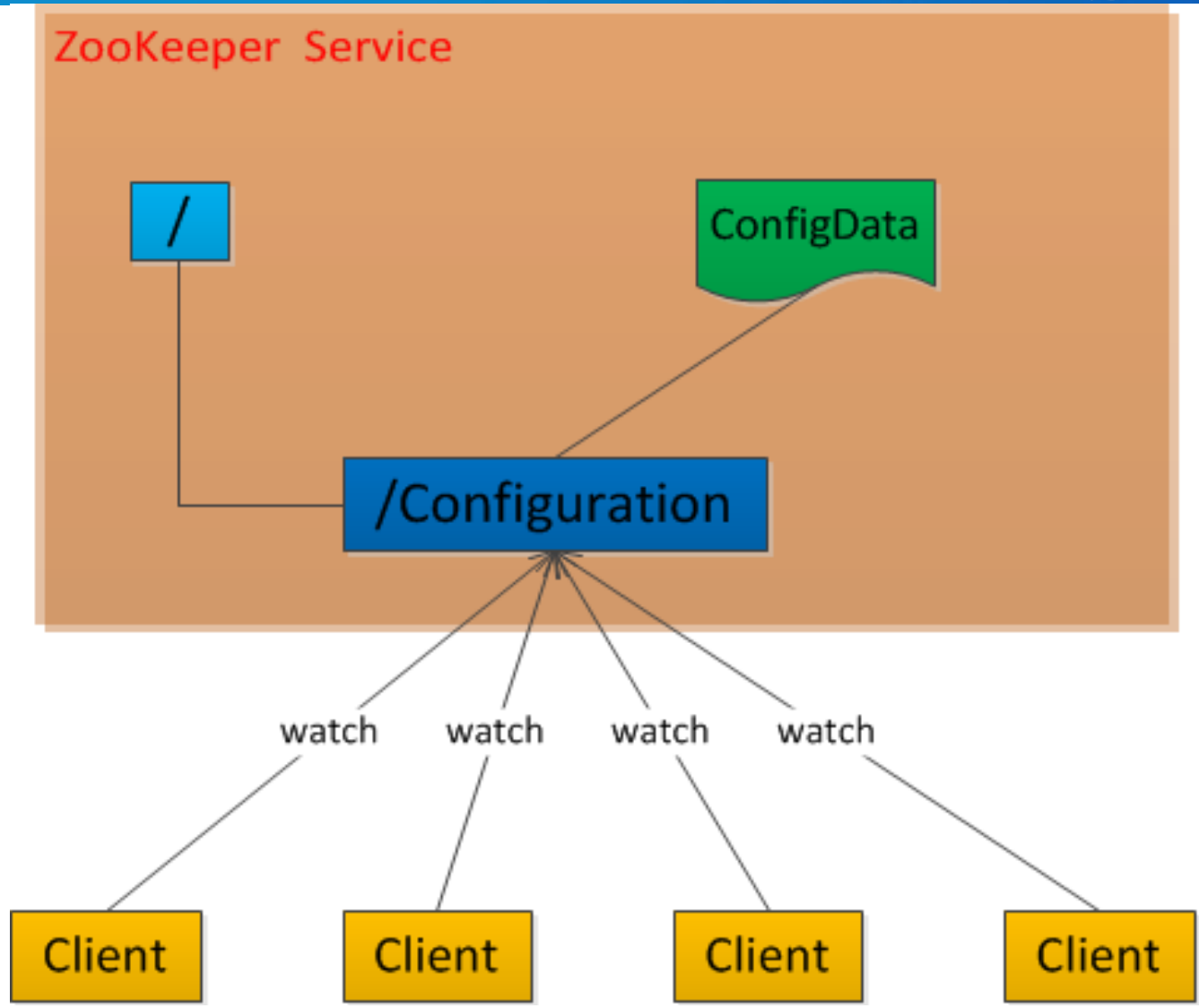
Zookeeper应用举例

- Hbase regionServer感知
  - ✓ /hbase/rs
  - ✓ 如果有一个RegionServer宕掉或者连接超时，那么regionServer在zk上的子目录便会自动删除。
  - ✓ HMaster监听到有一个zk子目录删除，便知道有一个regionServer宕了。



Zookeeper应用举例-数据发布和订阅

- 发布与订阅即所谓的配置管理，顾名思义就是将数据发布到ZK节点上，供订阅者动态获取数据，实现配置信息的集中式管理和动态更新。例如全局的配置信息，地址列表等就非常适合使用。集中式的配置管理在应用集群中是非常常见的，一般商业公司内部都会实现一套集中的配置管理中心，应对不同的应用集群对于共享各自配置的需求，并且在配置变更时能够通知到集群中的每一个机器。



Zookeeper应用举例-统一命名服务

分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

例如：公司有很多业务系统，这些业务系统都需要某种单据号，那么这个单据号肯定在所有系统中都是唯一的，在不指定哪个系统产生该单据号的情况下，可以使用ZK做这件事情。ZK可以保证原子性，不会因为多个系统的争抢造成不一致。

Zookeeper应用举例-分布通知/协调

ZooKeeper中特有watcher注册与异步通知机制，能够很好的实现分布式环境下不同系统之间的通知与协调，实现对数据变更的实时处理。使用方法通常是不同系统都对ZK上同一个znode进行注册，监听znode的变化（包括znode本身内容及子节点的），其中一个系统update了znode，那么另一个系统能够收到通知，并作出相应处理。

①另一种**心跳检测机制**：检测系统和被检测系统之间并不直接关联起来，而是通过ZK上某个节点关联，大大减少系统耦合。(HBase，HDFS，YARN)

②另一种**系统调度模式**：某系统由控制台和推送系统两部分组成，控制台的职责是控制推送系统进行相应的推送工作。管理人员在控制台作的一些操作，实际上是修改了ZK上某些节点的状态，而ZK就把这些变化通知给他们注册Watcher的客户端，即推送系统，于是，作出相应的推送任务。

③另一种**工作汇报模式**：一些类似于任务分发系统，子任务启动后，到ZK来注册一个临时节点，并且定时将自己的进度进行汇报（将进度写回这个临时节点），这样任务管理者就能够实时知道任务进度。

总之，使用zookeeper来进行分布式通知和协调能够大大降低系统之间的耦合。

Zookeeper应用举例-分布式锁

分布式锁，这个主要得益于ZooKeeper为我们保证了**数据的强一致性**，即用户只要完全相信每时每刻，zk集群中任意节点（一个zk server）上的相同znode的数据是一定是相同的。锁服务可以分为两类，一个是保持独占，另一个是控制时序。

①**保持独占**，就是所有试图来获取这个锁的客户端，最终只有一个可以成功获得这把锁。通常的做法是把ZK上的一个znode看作是一把锁，通过create znode的方式来实现。所有客户端都去创建 /distribute_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。

②**控制时序**，就是所有试图来获取这个锁的客户端，最终都是会被安排执行，只是有个全局时序了。做法和上面基本类似，只是这里 /distribute_lock 已经预先存在，客户端在它下面创建临时有序节点。Zk的父节点（/distribute_lock）维持一份sequence,保证子节点创建的时序性，从而也形成了每个客户端的全局时序。

Zookeeper应用举例-集群管理

集群机器监控:

这通常用于那种对集群中机器状态，机器在线率有较高要求的场景，能够快速对集群中机器变化作出响应。这样的场景中，往往有一个监控系统，实时检测集群机器是否存活。

Master选举:

Master选举则是zookeeper中最为经典的使用场景了，在分布式环境中，相同的业务应用分布在不同的机器上，有些业务逻辑，**例如**一些耗时的计算，网络I/O处，往往只需要让整个集群中的某一台机器进行执行，其余机器可以共享这个结果，这样可以大大减少重复劳动，提高性能，于是这个master选举便是这种场景下的碰到的主要问题。

利用ZooKeeper中两个特性，就可以实施另一种集群中Master选举：

① 利用ZooKeeper的强一致性，能够保证在分布式高并发情况下节点创建的全局唯一性，即：同时有多个客户端请求创建 /Master 节点，最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很轻易的在分布式环境中进行集群选举了。

②另外，这种场景演化一下，就是动态Master选举。这就要用到 EPHEMERAL_SEQUENTIAL类型节点的特性了，这样每个节点会自动被编号。允许所有请求都能够创建成功，但是得有个创建顺序，每次选取序列号最小的那个机器作为Master 。

Zookeeper shell命令

http://zookeeper.apache.org/doc/r3.4.6/zookeeperStarted.html#sc_ConnectingToZooKeeper

- ✓ 显示根目录下、文件： ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容 。
- ✓ 显示根目录下、文件： ls2 / 查看当前节点数据并能看到更新次数等数据
- ✓ 创建文件，并设置初始内容： create /zk "test" 创建一个新的 znode节点 “ zk ”以及与它关联的字符串
- ✓ 获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串
- ✓ 修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置
- ✓ 删除文件： delete /zk 将刚才创建的 znode 删除
- ✓ 退出客户端： quit
- ✓ 帮助命令： help

Zookeeper shell命令

➢ Zookeeper节点的状态信息

- ✓ czxid. 节点创建时的zxid.
- ✓ mzxid. 节点最新一次更新发生时的zxid.
- ✓ ctime. 节点创建时的时间戳
- ✓ mtime. 节点最新一次更新发生时的时间戳
- ✓ dataVersion. 节点数据的更新次数
- ✓ cversion. 其子节点的更新次数
- ✓ aclVersion. 节点ACL(授权信息)的更新次数
- ✓ ephemeralOwner. 如果该节点为ephemeral节点, ephemeralOwner值表示与该节点绑定的session id. 如果该节点不是ephemeral节点, ephemeralOwner值为0
- ✓ dataLength. 节点数据的字节数
- ✓ numChildren. 子节点个数

Zookeeper 常用四字命令

ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令 。

echo stat|nc 127.0.0.1 2181 来查看哪个节点被选择作为follower或者leader

echo ruok|nc 127.0.0.1 2181 测试是否启动了该Server，若回复imok表示已经启动。

echo dump| nc 127.0.0.1 2181 ,列出未经处理的会话和临时节点。

echo kill | nc 127.0.0.1 2181 ,关掉server

echo conf | nc 127.0.0.1 2181 ,输出相关服务配置的详细信息。

echo cons | nc 127.0.0.1 2181 ,列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。

echo envi |nc 127.0.0.1 2181 ,输出关于服务环境的详细信息（区别于 conf 命令）。

echo reqs | nc 127.0.0.1 2181 ,列出未经处理的请求。

echo wchs | nc 127.0.0.1 2181 ,列出服务器 watch 的详细信息。

echo wchc | nc 127.0.0.1 2181 ,通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表。

Zookeeper Java API

http://zookeeper.apache.org/doc/r3.4.6/javaExample.html
Zookeeper高级API工具curator：
http://curator.apache.org/getting-started.html

Elastic Job

作业

1.请说明zookeeper的节点个数为什么是奇数个？
2.HBase创建表如何预分区，举例说明。Shell命令，JAVA
3.了解一下phoenix，并简单介绍一下phoenix。
4.了解一下openTSDB,并简单介绍openTSDB的设计思想。

THANKS