Tair

Tair介绍

Tair简介

Tair(Taobao Pair)是淘宝开发的分布式Key-Value存储引擎

服务器端自动负载均衡

分为持久化和非持久化两种方式存储

非持久化：分布式缓存使用 Memcached（mdb）、Redis（rdb）

持久化：SQL-DB使用FireBird（fdb）

NoSQL-DB：使用Kyoto Cabinet（kdb）、LevelDB（ldb）

Tair采用可插拔存储引擎设计，以上这些存储引擎可以很方便的替换，还可以引入新的存储引擎比如：MySQL

使用场景

分布式缓存

大多数使用场景

大访问少量临时数据的存储（kb左右）

用于缓存，降低对后端数据库的访问压力

session场景

高速访问某些数据结构的应用和计算（rdb）

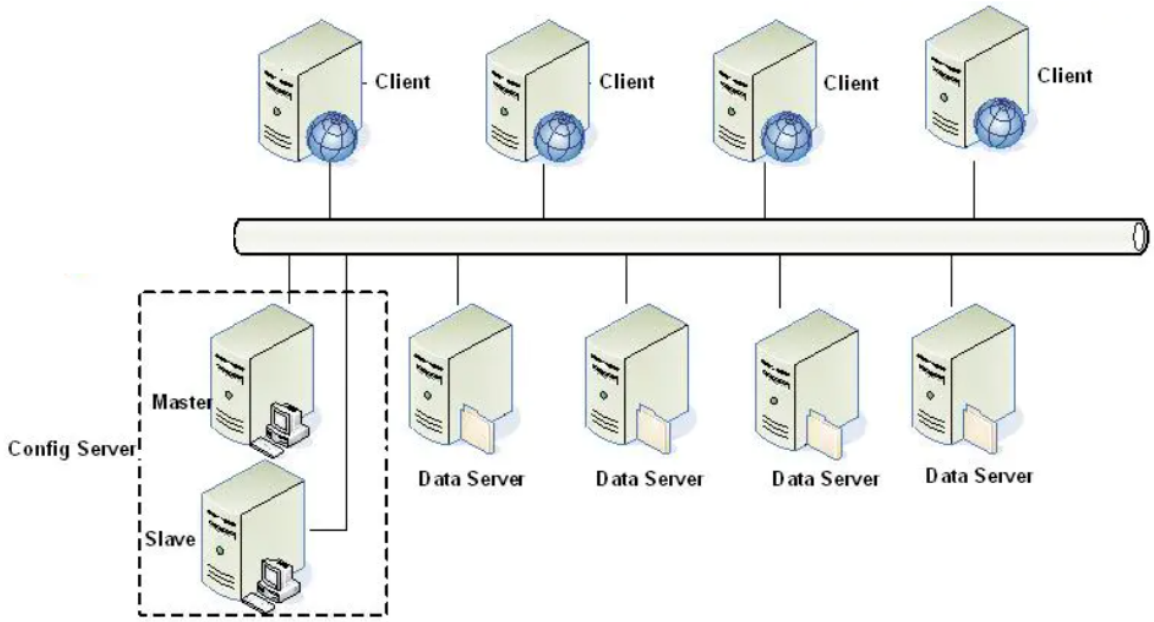
数据源存储

快速读取数据（fdb）

持续大数据量的存入读取（ldb），交易快照

高频度的更新读取（ldb），库存

Tair整体架构分析



一个Tair集群主要包括client、Config server和Dataserver 三个不同的应用。

Client在初始化时，从Config server处获取数据的分布信息，根据分布信息和相应的Data server交互完成用户的请求。

Config Server通过和Data Server的心跳（HeartBeat）维护集群中可用的节点，并根据可用的节点，构建数据的在集群中的分布信息

Data server负责数据的存储，并按照Config server的指示完成数据的复制和迁移工作。

Config Server

Config Server是单点，采用一主一备的方式保证可靠性。

管理所有的data server, 维护data server的状态信息

用户配置的桶数量、副本数、机房信息

数据分布的对照表

协调数据迁移、管理进度，将数据迁移到负载较小的节点上

Client和ConfigServer的交互主要是为了获取数据分布的对照表，当client获取到对照表后，会cache这张表，然后通过查这张表决定数据存储的节点，所以不需要和configserver交互，这使得Tair对外的服务不依赖configserver，所以它不是传统意义上的中心节点

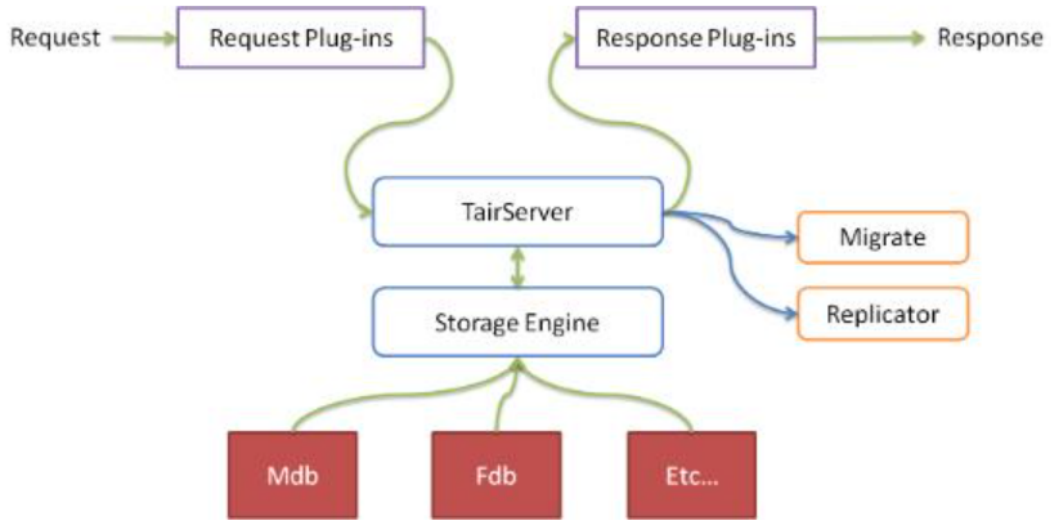
Config server维护的对照表有版本概念，由于集群变动或管理触发，构建新的对照表后，对照表的版本号递增，并通过Data server的心跳，将新表同步给数据节点。

客户端和Data server交互时，Dataserver每次都把自己缓存的对照表版本号放入response结构中，返回给客户端，客户端将Data server的对照表版本号和自己缓存的对照表版本号比较，如果不相同，会主动和Config server通信，请求新的对照表。

Tair的Configserver使客户端使用时候，不需要配置数据节点列表，也不需要处理节点的的状态变化，这使得Tair对最终用户来说使用和配置都很简单。

Data Server

Data server负责数据的物理存储，并根据Configserver构建的对照表完成数据的复制和迁移工作。Data server具备抽象的存储引擎层，可以很方便地添加新存储引擎。Data server还有一个插件容器，可以动态地加载/卸载插件。



Tair的存储引擎有一个抽象层，只要满足存储引擎需要的接口，便可以很方便地替换Tair底层的存储引擎。比如你可以很方便地将bdb、tc甚至MySQL作为Tair的存储引擎，而同时使用Tair的分布方式、同步等特性。

Tair默认包含两个存储引擎：mdb和fdb。

mdb是一个高效的缓存存储引擎，它有着和memcached类似的内存管理方式。mdb支持使用share memory（tmpfs），这使得我们在重启Tair数据节点的进程时不会导致数据的丢失，从而使升级对应用来说更平滑，不会导致命中率的较大波动。

fdb是一个简单高效的持久化存储引擎，使用树的方式根据数据key的hash值索引数据，加快查找速度。索引文件和数据文件分离，尽量保持索引文件在内存中，以便减小IO开销。使用空闲空间池管理被删除的空间。

Tair的安装与使用

环境准备

GIt

```
yum install git
yum update -y nss curl libcurl
```

svn

```
yum install subversion
```

Libtool

```
yum install libtool
```

boost-devel

```
yum install boost-devel
```

zlib

```
yum install zlib-devel
```

C++

```
yum install gcc-c++
```

项目源码下载和编译

tbsys库和tbnet库下载(tair底层依赖tbsys库和tbnet库**)**

```
#git源码下载
git clone https://github.com/kayaklee/tb-common-util.git
#设置环境变量
mkdir /var/tblib
export TBLIB_ROOT="/var/tblib"
#因为tbnet和tbsys在两个不同的目录，但它们的源码文件里头文件的互相引用却没有加绝对或相对路径，将两个目录的源码加入到C++环境变量中即可。否则编译时会出现：“fatal error:tysys.h: No such file or directory”的错误。
CPLUS_INCLUDE_PATH=/home/tair/tb-common-util/tbsys/src:/home/tair/tb-common-util/tbnet/src
export CPLUS_INCLUDE_PATH
#修改tbsys代码
cd ~/tb-common-util/trunk/tbsys/src
下载的代码有个错误：具体是tbsys/src/tblog.cpp中323行代码:需要将CLogger::CLogger& CLogger::getLogger()改为CLogger& CLogger::getLogger()
#编译tbsys和tbnet
cd tb-common-utils
./build.sh
注：安装成功后，TBLIB_ROOT所指示的目录下会有include和lib两个目录。
```

tair下载

```
#git源码下载
git clone https://github.com/alibaba/tair.git
#安装依赖
yum install -y openssl-devel libcurl-devel
#编译
./bootstrap.sh
#检测和生成 Makefile
./configure
#编译和安装到目标目录
make -j
make install
注：默认安装位置是 ~/tair_bin
```

Tair配置和启动

下面以MDB引擎为例配置一个最小化的Tair集群（1 * ConfigServer + 1 * DataServer）

```
```

```
#查看和设置系统tmpfs
#MDB 引擎默认使用共享内存，所以需要查看并设置系统的tmpfs的大小
# 这里根据实际机器内存情况配置，必须大于Tair使用内存的配置
vim /etc/fstab
tmpfs /dev/shm tmpfs defaults,size=1024M 0 0
#生效
mount -o remount /dev/shm
#查看tmpfs
cat /etc/fstab | grep /dev/shm
显示tmpfs       /dev/shm       tmpfs    rw,size=1G       0 0
# 定义配置文件 复制配置文件
cp etc/configserver.conf.default etc/configserver.conf
cp etc/group.conf.default etc/group.conf
etc/dataserver.conf.default etc/dataserver.conf
# configserver.conf
[public]
config_server=192.168.127.133:5198
#config_server=192.168.1.2:5198
#config_server=10.211.55.9:5198
#config_server=192.168.1.2:5198
dev_name=eno16777736
#group.conf
# data center A
_server_list=192.168.127.133:5191
#_server_list=192.168.1.2:5191
#_server_list=192.168.1.3:5191
#_server_list=192.168.1.4:5191

# data center B
#_server_list=192.168.2.1:5191
#_server_list=192.168.2.2:5191
#_server_list=192.168.2.3:5191
#_server_list=192.168.2.4:5191

#dataserver.conf
[public]
config_server=192.168.127.133:5198
#config_server=192.168.1.2:5198

dev_name=eno16777736
mdb_inst_shift=0
process_thread_num=4
io_thread_num=4
#这里 slab_mem_size控制MDB内存池的总大小，mdb_inst_shift 控制实例的个数，注意这里一个实例
必须大于512MB且小于64GB。
slab_mem_size=512
#修改 tair.sh启动脚本
#在CentOS 7下，安装目录下的 tair.sh 启动脚本有一行代码需要修改
tmpfs_size=`df -m |grep /dev/shm | awk '{print $2}'`
#启动Tair实例
#启动dataserver
tair_bin $ ./tair.sh start_ds
#启动configserver
tair_bin $  ./tair.sh start_cs
#查看进程
ps -ef |grep tair
注：执行后没有两行记录显示，说明启动失败，可查看logs中的log查看错误信息。
```

Tair测试

```
#客户端读写测试
tair_bin $ ./sbin/tairclient -c 192.168.127.133:5198 -g group_1
TAIR> health
TAIR> put key value
TAIR> get key
TAIR> remove key
TAIR> get key
```

Tair停止

```
#停止dataserver
tair_bin $ ./tair.sh stop_ds
#停止configserver
tair_bin $ ./tair.sh stop_cs
```

Tair高可用和负载均衡

Tair的高可用和负载均衡，主要通过对照表和数据迁移两大功能进行支撑。

对照表将数据分为若干个桶，并根据机器数量、机器位置进行负载均衡和副本放置，确保数据分布均匀，并且在多机房有数据副本。

在集群发生变化时，会重新计算对照表，并进行数据迁移。

对照表

在Tair系统中，采用对照表将数据均衡的分布在DataServer上

还能动态适应节点的扩容和缩容

Tair基于一致性Hash算法存储数据，根据配置建立固定数量的桶（bucket）

桶为Hash环节点，hash(key) 顺时针 设置桶

桶是负载均衡和数据迁移的基本单位

config server 根据一定的策略把每个桶指派到不同的data server上，

因为数据按照key做hash算法，所以可以认为每个桶中的数据基本是平衡的，

保证了桶分布的均衡性, 就保证了数据分布的均衡性。

比如：

```
bucket   data server
0       192.168.127.132   192.168.127.134
1       192.168.127.133   192.168.127.135
2       192.168.127.132   192.168.127.134
3       192.168.127.133   192.168.127.135
4       192.168.127.132   192.168.127.134
5       192.168.127.133   192.168.127.135
```

Tair支持自定义的备份数，比如你可以设置数据备份为2，以提高数据的可靠性。对照表可以很方便地支持这个特性。

第二列为主节点的信息，第三列为辅节点信息。在Tair中，客户端的读写请求都是和主节点交互。

当有节点不可用时，如果是辅节点，那么configserver会重新为其指定一个辅节点，如果是持久化存储，还将复制数据到新的辅节点上。如果是主节点，那么configserver首先将辅节点提升为主节点，对外提供服务，并指定一个新的辅节点，确保数据的备份数。

对照表的初始化

第一次启动ConfigServer会根据在线的DataServer生成对照表

ConfigServer会启动一个线程（table_builder_thread）,该线程会每秒检查一次是否需要重新构造对照表

Tair 提供了两种生成对照表的策略：

负载均衡优先

config server会尽量的把桶均匀的分布到各个data server上

一个桶的备份数据不能在同一台主机上

位置安全优先

一般我们通过控制 _pos_mask（Tair的一个配置项） 来使得不同的机房具有不同的位置信息

一个桶的备份数据不能都位于相同的一个位置（不在同一个机房）

数据迁移

当DataServer增加时，config server负责重新计算一张新的桶在data server上的分布表

将桶较为平均的分配到各个DataServer上

当DataServer发生故障时， config server负责重新计算一张新的桶在data server上的分布表,

将原来由故障机器服务的桶的访问重新指派到其它的data server中

这时需要做数据迁移，具体流程如下：

1、设置当前正在迁移的桶ID

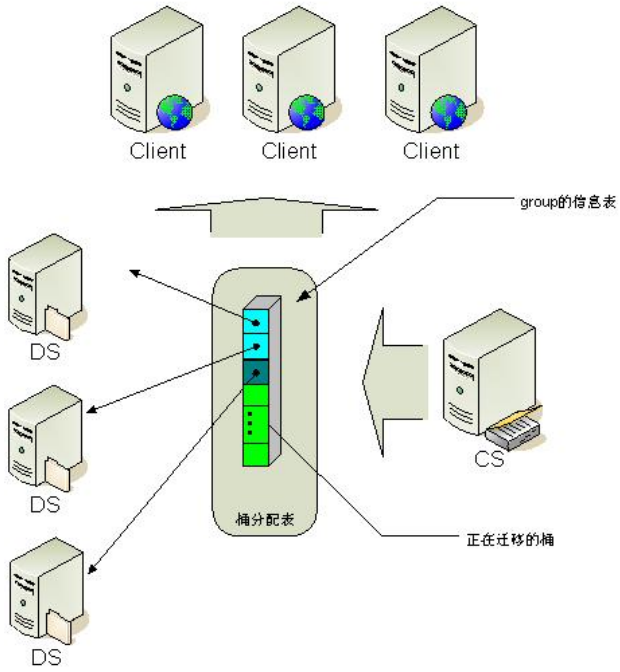
2、DataServer写入桶数据时，会写入redolog

3、migrate_manager迁移内存中的桶数据

4、migrate_manager迁移redolog数据

5、redolog数据迁移完成后，将桶标记为迁移完成

6、将信息发送给ConfigServer用于同步对照表



Tair存储引擎

Tair的存储引擎有一个抽象层（storage_manager）,只要实现存储引擎接口，便可以替换Tair的底层存储引擎。

可插拔存储引擎---类似MySQL

Tair默认包含四种存储引擎：mdb、fdb、kdb和ldb

mdb

数据存储：Memcached

高效缓存存储

使用share memory ， 重启不会数据丢失

支持K-V存储、prefix操作

适用于：String缓存使用 （json）、大访问少量的临时数据存储、Session分离

fdb

数据存储：FireBird

高效的持久化SQL存储

索引文件和数据文件分离------>mysql MyISam

使用Tree的方式根据key的hash值索引数据 B tree

索引文件在内存中

适用于：快速访问较小的数据

kdb

数据存储：Kyoto Cabinet

Cabinet开发的KV的持久化存储

简单的包含记录的数据文件

存储形式为hash表或B+Tree

适用于：简单临时存储

ldb

数据存储：LevelDB

是Google开发的高性能持久化KV存储

可内嵌mdb缓存　mdb+ldb

支持kv、prefix

支持批量操作

适用于：大数据量的存取（交易快照）、高频度的更新（库存）、离线大批量数据导入

rdb

数据存储：Redis

高效缓存存储

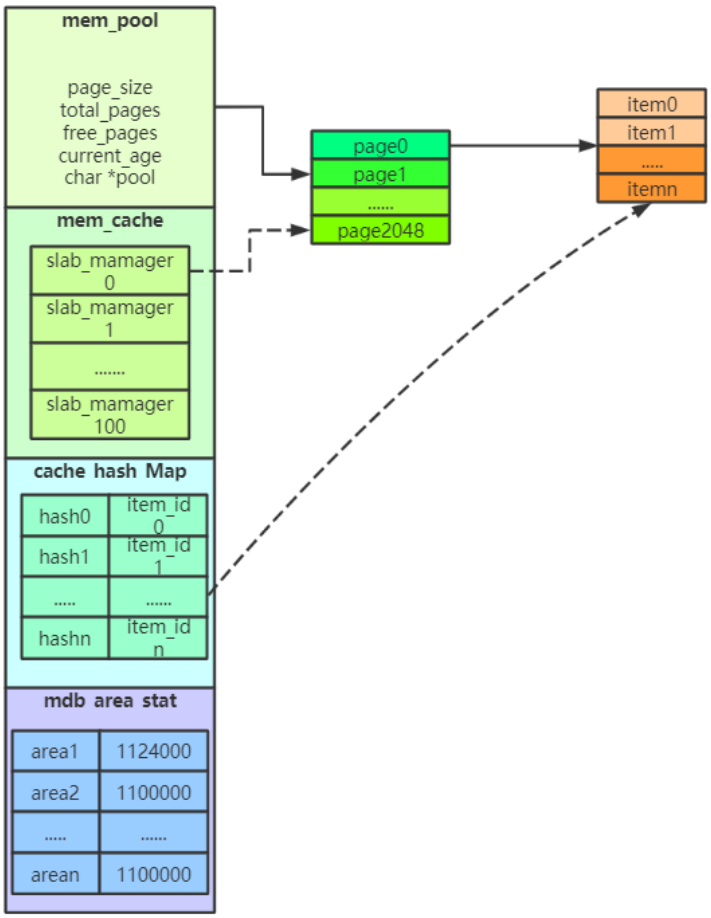
多种数据结构和计算

适用于：复杂数据结构存储、商品属性、粉丝列表、商品评论、消息队列

	数据存储	**持久化**	**数据格式**	**应用场景**
mdb	Memcached	非	K-V存储、prefix操作	缓存使用、Session分离
fdb	FireBird	是	SQL、BTree	快速访问较小的数据
kdb	Kyoto Cabinet	是	Hash、B+Tree	简单临时存储
ldb	LevelDB	是	kv、prefix、batch	大数据量高频度的存储
rdb	Redis	非	List、Set、Hash....	复杂数据结构缓存

阿里使用：mdb、rdb、ldb

mdb的存储结构（Memcached）



mem_pool

用于共享内存管理，将内存分为若干个page

page个数根据slab_mem_size设置，单位为MB

单个DataServer最多使用64G内存

mem_cache

用于管理slab，存放slab_manager列表

slab_manager管理item(数据块)

slab个数为100，一个slab可存储800kb

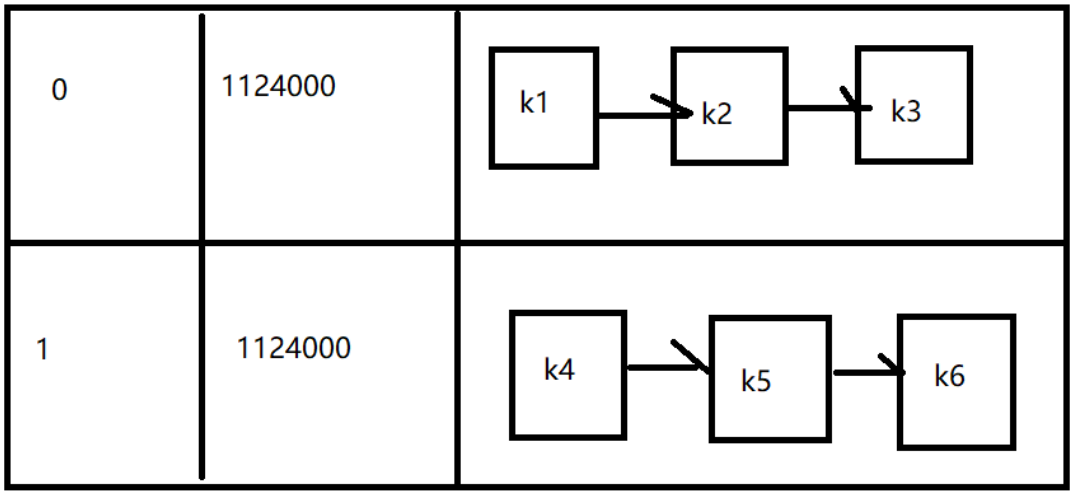
cache_hash_map

用于存储hash表，根据key进行hash对应item数据

hash冲突，产生链表

mdb_area_stat

用于维护area（namespace）状态，记录了该area的数据链表和数据量限制(tablespace)



Tair相应API

key/value常见操作API

	操作	说明
get	get key [area]	获得key的值
put	put key data [area] [expired]	设置key和value
incr	incr key [count] [initValue] [area]	对key的值自增
decr	decr key [count] [area]	对key的值自减
batcheGet	mget key1 ... keyn area	批量获得
expire	expire key time	设置key的过期时间
prefixPut	pput [area] pkey skey value	根据前缀设置，按照prefix计算hash，同一个prefix会存储在同一个hash表，形成链表

	操作	说明
prefixGet	pget [area] pkey skey	根据前缀读取

version

Tair中的每个数据都包含版本号，版本号在每次更新后都会递增。这个特性有助于防止由于数据的并发更新导致的问题，类似于乐观锁

比如，系统有一个value为“a”，A和B同时get到这个value。

A执行操作改为"b"

B执行操作改为"c"

如果不控制，无论A和B谁先更新成功，它的更新都会被后到的更新覆盖。

使用version解决

第一次put version为1

get version为1

A修改成功后 version增加1

B修改时本身version为1，小于服务器版本2

所以B修改不成功

在put时，version传0 可强制修改

```
struct mdb_item{
    uint64_t item_id;
    .....
    uint16_t version
}
```

Tair实现乐观锁

trylock（获取版本）–> transaction –> unlock(版本检查) — 正常—> version+1 commit
— 异常—> rollback

- 只要能获取版本，就认为trylock成功
- 版本检查则更新version，一般是加1；否则，异常处理，version不变，并开启回滚
- 多事务同时获得乐观锁时，unlock只能有一个成功，其余的都应该失败

version 初始化要大于1

Tair实现分布式锁

利用 Tair 的 version 特性可以实现分布式锁，由于 LDB 具有持久化功能，当服务有出现宕机的情况，也不会因此出现锁丢失或者锁不可释放的情况。

如果KEY不存在的话，传入一个固定的初始化VERSION（需要大于1），Tair会在保存这个缓存的同时设置这个缓存的VERSION为你传入的 VERSION+1

KEY如果已经存在，Tair会校验你传入的VERSION是否等于现在这个缓存的VERSION，如果相等则允许修改，否则将失败。

示例代码如下：

```
//获得锁
public boolean lock(String lockKey) {
        //10 :version  expiretime : 过期时间 秒
        ResultCode code = defaultTairManager.put(lockKey, defaultVlaue, 5,
expiretime);
        if (ResultCode.SUCCESS.equals(code))
            return true;
        else
            return false;
    }

//释放锁
public Boolean unlock(String lockKey){
        ResultCode code = defaultTairManager.delete(lockKey);
        return ResultCode.SUCCESS.equals(code);
  }
```

Java Client

pom.xml

```
<!-- https://mvnrepository.com/artifact/com.taobao.tair/tair-client -->
        <dependency>
            <groupId>com.taobao.tair</groupId>
            <artifactId>tair-client</artifactId>
            <version>2.3.5</version>
        </dependency>
```

demo

```
        DefaultTairManager defaultTairManager = new DefaultTairManager();
        List<String> cs = new ArrayList<String>();
        cs.add("192.168.127.132:5198");
        defaultTairManager.setConfigServerList(cs);
        defaultTairManager.setGroupName("group_1");
        defaultTairManager.init();
        defaultTairManager.put(0,"name:001","zhangfei");
        Result<DataEntry> value= defaultTairManager.get(0,"name:001");
        System.out.println(value);
        //设置版本0 强制更新  过期时间2秒
        defaultTairManager.put(0,"name:002","zhaoy",0,2);
        Result<DataEntry> value2= defaultTairManager.get(0,"name:002");
        System.out.println(value2);
```

Tair

机房、负载均衡、数据迁移、不停止服务

更好的可用性和负载均衡

Redis

多数据结构处理、计算

Tair+Redis