目录

[定时器Timer类 1](#_Toc90574927)

[功能 1](#_Toc90574928)

[部分成员 1](#_Toc90574929)

[实现方法 1](#_Toc90574930)

[Client设计 2](#_Toc90574931)

[本地缓存 2](#_Toc90574932)

[工作流程 2](#_Toc90574933)

[异常处理 3](#_Toc90574934)

[应用层超时重传 3](#_Toc90574935)

[收发包管理 4](#_Toc90574936)

[日志 4](#_Toc90574937)

[不足之处 4](#_Toc90574938)

# 定时器Timer类

## 功能

基于链表和信号实现了单进程下的多定时器，最小定时间隔1微秒，同时可指定单次定时或循环定时，通过void\*结构体为定时器回调函数传入参数。

## 部分成员

文本

描述已自动生成

## 实现方法

使用**sigaction**绑定**SIGALRM**信号处理函数，使用**setitimer**以一定的时间间隔发送**SIGALRM**信号，在信号处理函数中遍历定时器链表，更新并检查每个定时器的信息，判断是否达到定时时间并执行每个定时器单独的回调函数。

# Client设计

## 本地缓存

为了减轻Master-Server的压力，增加一层本地缓存存放Key-Addr数据，防止每次都需要拉取分布导致Master-Server压力过大。缓存方案选择了Key-Addr，在实现上采用LRU作为缓存淘汰方案。

## 工作流程

图示

描述已自动生成

图片包含 图示

描述已自动生成

## 异常处理

### 应用层超时重传

TCP虽然可以保证数据可靠传输，但传输成功不代表被对方逻辑层成功处理。基于内存容量方面的考虑，服务端可能使用了限长的消息队列，如果收到的瞬时消息过多，超过了消息队列的可处理个数，所有超出的消息会被它丢弃。因此，应用层超时重传可以解决没有被逻辑层成功处理的情况。

图形用户界面, 文本, 应用程序

描述已自动生成

为每个Cache-Server连接分配一个定时器。

文本

描述已自动生成

### 收发包管理

当Cache-Server缩容或宕机时，无论通过本地缓存的addr还是通过Master-Server再获取的addr，都已经失效，此时读写请求将会失败。为了保证Client的每一个请求都能够得到正确响应，在本地为每个Cache-Server维护一个Key-List用于进行收发包管理。Client向Cache-Server发送一次请求时，将Key存入本地的某个对应此Cache-Server的Key-List中。当Client收到此Cache-Server的回复后，消除掉此Cache-Server的Key-List中对应的请求key。如果某个Key-List的size超过一定的大小，则判定该Key-List所属的Cache-Server出现了缩容或宕机。在主动发现缩容或宕机后，重新连接Master并获取没有被响应的Key的最新分布，然后重新向Cache-Server发送请求。

手机屏幕截图

描述已自动生成

## 日志

使用[easylogging++](https://github.com/amrayn/easyloggingpp)作为日志库。Easylogging的特点是只需一个头文件，所有功能都是内部实现，无需依靠其他第三方库，使用方便。

## 不足之处

1. 定时器定时时间只能是遍历间隔的整数倍，可以使用基于升序链表的定时器容器、时间轮、时间堆来处理多个定时事件
2. 某个Cache-Server宕机后，Master-Server无法立刻发现其宕机，此时若Client访问宕机的Cache-Server，将无法连接，那么Client会去Master-Server拉取最的分布，但此时Master-Server并不知道Cache-Server宕机，因此其内部的分布信息依然是旧的。
3. 在Client上线初期，将多次访问Master-Server以建立本地缓存，若很多Client仅仅只查询少量几次数据，则Client本地的Key-Address缓存可能无法被使用到，即在此种情况下Key-Address缓存失效，造成Master-Server压力增大。
4. 无法实现透明传输，数据包采用#为控制符，Key或Value中出现'#'会产生错误