**SUBWAYv1.2.0详细设计**

**说明书**

（设计文件）

**二零一八年一月**

**修订历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 日期 | 状态 | 修订人 | 摘要 |
| 1.0.0 | 2017-12 | C | 吴志明 | 起草 |
| 1.1.0 | 2018-1 | A | 吴志明 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

状态标识：C – Created A - Added M - Modified D - Deleted

# 业务概述

## 业务背景

## 目的和范围

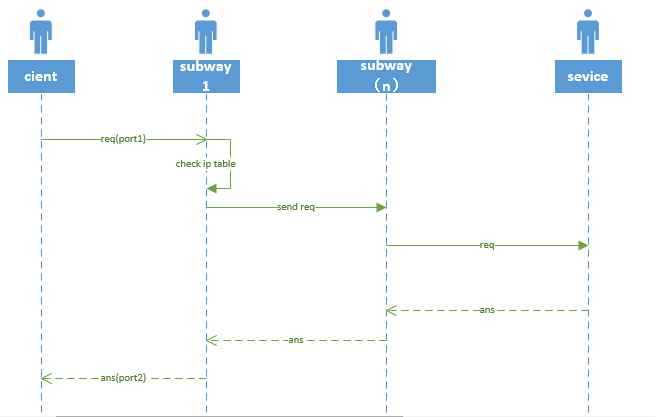
为后续开发及维护提供依据

## 术语和定义

## 参考资料

# 设计概述

## 系统架构



## 2.2、模块架构

**2.2.1、V1.0.0**

**一、接收模块**

1、一个连接以4K为一个处理单位。每接收4K开一个协程处理。

**二、处理模块**

1、根据消息头定义将消息发送到哪个队列和从哪个队列接收消息。

2、每个队列都存储在内存中。

**三、订阅模块**

1、为每个消息寻找订阅者，找到订阅者后。将消息推送给订阅者。

2、消息存储在固定的消息队列，消息插入和发布都需要上锁。

**四、当前架构思考**

1、经过测试、当数据量大的时候，且如果只有接受端的时候，subway的内存会爆涨。

2、当前代码中，每读取4K开一个协程处理。当数据交互比较大的时候，内存利用率不高。

3、由于消息入队和发布在同一个队列上，在操作队列的时候都需要加锁。使接收模块和订阅模块其实是加密耦合的。

4、消息丢包无法排查。

五、当前架构适用性

当前架构只适合在并发量较低且数据量较低，实时的场景

**2.2.1、V1.1.0**

**一、接收模块设计**

1.1、协程数量计算：

1、结果测试、一个协程的内存开销约为4K。

2、底层tcp协议每个socket连接也要消耗一定的内存空间。约4K（一个tcp网络链接内存消耗）

3、单节点以100W用户量计算，在无传输数据的情况下的内存使用量为1024\*1024 \* 8 \* 1024 = 8G。但是实际上很少场景会达到。以10W用户同时在线的情况下，一个连接对应2个协程使用1.6G。

1.2、多协程的副作用。

在golang中，需要特别注意的一点是所有的调度是平均的，如果要保证核心处理协程高效，需要尽量减少协程数量。如果100W个用户同时在线，以一个用户链接使用两个协程计算，那么就会有200W个协程产生，读写压力不说，光golang的调度，已经是一件很消耗cpu的了。需要注意，可能又时候读取完成到处理完成会不连续。

1.3、为每个链接配置两个两个协程处理。一个处理接收消息，一个处理发送消息。

**二、处理模块设计**

2.1、主题消息进行文件存储。一个是文件索引消息位置，一个是存储的文件。文件可以通过配置按需删除。

2.2、处理模块依然以4K为单位接收数据，但是处理的时候需要判断数据大小，4K的数据头肯定还是消息头，如果是短消息，支持在4K数据里边包含多个消息体。写文件以4K的整数倍大小写入。采用append的方式写入文件。

2.3、读取消息体的时候，通过消息序号来获取消息。每个主题对应一个文件存储，还有一个状态文件用来定位消息在文件中的偏移。由于状态文件会比较小，所以每次处理的时候会读取状态文件前后4K大小的内容到内存中。

**三、订阅模块**

3.1、消息订阅者发送消息到subway的时候需要发送订阅的主题以及编号，消息体为空即可。

3.2、subway通过调用者的订阅主题和编号，从状态文件中获取消息体位置，查看预留内存中是否有消息内容，然后获取消息返回给订阅者。并将消息后4M（可配置）文件内容存入内存。

**四、监控模块**

**3.1、内存监控**

使用golang内存监控方式监控内存，如果使用的内存超过健康值，采用的应对措施：

1. 请求拒绝、客户端做优化，如果服务端拒绝，则隔一段时间后重传。
2. 减少缓存长度。
3. 调用GC

**2.2、磁盘监控**

如果磁盘读写太大，发出告警。

## 设计策略

**V1.0.0 消息头内容**

//乘客头信息

**type** PassengerHead **struct** {

    IsGiz int64 //是否需要压缩

    iRuntime int64 //执行开始时间

    StrListenList [32]byte //从某个队列获取返回结果

    StrPushList [32]byte //入参发送到对应的队列

    Flag int64 //0 入参， 1出参

    Id int64 //发送或者接收者的Id号

    IputNumber int8 //参数个数

    BodyLen int64 //消息体长度

    IsLast int64 //是否为同一个调用的最后一次

    ListenTime int64 //监听时长

    IsBroken int //主动断开

}

**V1.1.0 消息头内容**

//乘客头信息

**type** PassengerHead **struct** {

    IsGiz int64 //是否需要压缩

    iRuntime int64 //执行开始时间

    StrListenList [32]byte //从某个队列获取返回结果

    StrPushList [32]byte //入参发送到对应的队列

    Id int64 //消息编号。

    BodyLen int64 //消息体长度

    Offset     int64 //消息体偏移，如果消息超过4K，那么需要多次发送，服务端用偏移拼接和校验消息体

    ListenTime int64 //监听时长,过了监听时长后连接会在服务端或者客户端会主动断开

    IsBroken int //主动断开

}

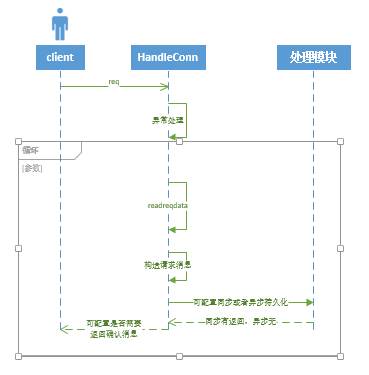
**2.2.1、接收模块**

|  |  |
| --- | --- |
| 方法名 | 方法描述 |
| HandleConn(Conn \*net.Conn) error | 处理客户端请求 |

要点：

1. 客户端需要给客户端发送两次消息，第一次是消息头，第二次是消息体。保存在subway中的消息包含消息头和消息体。
2. 大消息需要独占一个消息队列。

流程：



接收模块处理流程

1. 例行每个主要处理入口需要做异常处理。
2. 读取客户端发送到服务端的消息体，以4K为单位读取，客户端也是以4K为单位请求。
3. 拼接消息请求
4. 消息体小于4K，可能是4K中包含一条或者多条消息。那边直接把bug推给处理模块处理。
5. 消息体大于4K,小于40M（需要配置），消息头压缩部分必须为压缩。接收模块开辟一片BUG存储。多次传输，每次都需要加上消息头。
6. 消息体大于40M（需要配置），不处理。
7. 把消息体推到处理模块处理。

5、返回给客户端处理结果（可配，也可以不返回。）

**2.2.2、处理模块**

主要功能：

负责持久化请求数据。

要点分析：

1. 每次写为定时、定量写。定时用定时器实现。定量为当buf满的时候执行写操作。由于如果消息体为大消息体例外。
2. 拷贝次数方面，有两个方案：
3. 实现0拷贝，数据接收模块直接使用处理模块的buf。这样可以实现0拷贝

限制：客户端和服务端需要做好约定，如果传的内容只有长度，那么要有个回退操作，还有就是多个连接写入同一个buf的时候需要加锁。

缺点：如果需要分离处理模块和接收模块到不同进程，就完全不适用。且只适合高并发场景，可能会造成死锁。

1. 每个客户端发送的消息，由接入模块缓存到一定数量后再拷贝给处理模块。然后处理模块在进行处理。

缺点：接收模块需要加上组装数据的功能，数据推到处理模块，处理模块处理数据，都需要加锁。会产生很多零散的内存块。

综合来看：采用方案2，数据结构上，采用golang的list。减少锁操作时间。

1. 多一次拷贝和分多次IO操作，需要测试验证。实际上哪一种会比较快。这边默认做法是，判断接收模块推的数据的大小，足够大的话直接做IO操作，太小的话就做合并。
2. 一次写入的大小。需要测试多少的效率是最高的。

主要处理流程：

1. 从队列中取数据。如果队列为空则不处理。
2. 如果队列中的数据超过1M，直接写IO。将数据合并到合并队列，然后再一起写IO。
3. 如果队列只有一条数据，直接写IO。

二、增加的结构

1、IndexFile –索引文件

属性：

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 描述 |
| FileName | String | 索引文件名 |

Open() error方法

|  |  |
| --- | --- |
| 方法名 | Open() error |
| 描述 | 打开索引文件，以追加方式打开。 |
| 输入参数 | 无 |
| 输出参数 | 成功：nil，失败：具体失败的原因 |
| 异常 | 无 |

Close()方法

|  |  |
| --- | --- |
| 方法名 | Close() |
| 描述 | 关闭索引文件 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 异常 | 无 |

Append([]byte) (len int, err error)方法

|  |  |
| --- | --- |
| 方法名 | Append([]byte) (len int, err error) |
| 描述 | 消息索引追加写入文件。 |
| 输入参数 | 无 |
| 输出参数 | 无 |
| 异常 | 无 |

2、MsgFile ---消息文件结构

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 描述 |
| FileName | String | 文件名 |
| IndexfileName | String | 索引文件名 |

**2.2.3、订阅模块**

主要功能：

消息订阅者可以从subway中获取其需要类型的消息。

要点：

1. 如果订阅的ID为0，服务端返回消息的最大值和最小值。
2. 服务端通过订阅主题和消息号获取消息。
3. 消息返回形式：
4. 小于4K，以4K发送。消息头和消息体一起发送
5. 大于4K，消息头后再发消息体。
6. 消息体特大的情况下支持多点传输功能。每次传输消息块大小，需要测试验证。
7. 读取一个文件，每次读取的大小为多少效率最高，需要测试。
8. 控制一个连接在获取消息或者发送消息时候的内存消耗。

流程：

1. 客户端发起消息获取请求。带有消息号和消息主题
2. 服务端查看消息主题和消息号是否存在，如果不存在，返回直接返回失败。并记录客户端订阅失败次数，如果短时间内有多次请求都失败。断开客户端连接。
3. 如果消息号存在，获取消息体。判断消息体大小，如果小于4K，一次发送，包括消息头和消息体。如果消息体太大，这返回给客户端消息体大小，让客户端判断需要获取消息体的哪一段。如果客户端带上消息体段大小及其实位置，每次4K发送给客户端。每次都要带上消息头。
4. 客户接受到应答消息后，再传第二次。
5. 消息的拼接由客户端去完成。

**2.2.4、监控模块**

要点：

使用golang的pprof接口获取内存信息。每分钟获取一次并记录。

用脚本获取各种系统信息，包括cpu使用率，磁盘读写大小，磁盘空间大小等。

* 1. **特殊情况**

一、每个消息都是超大消息。

问题：100W用户，每个用户一条发100M大小的消息。需要的存储空间为1M\*100M = 1T。如果都存在内存做缓存后再写入文件，肯定是不行的。

处理方法：超过64K大小的消息，直接先存文件，状态文件中增加文件序列号。其中序列号大于1000的文件为临时文件。临时文件会在系统比较空闲的时候再合并到主文件，合并过程中会修改状态文件。

# 系统主要功能



## 报价参数设置