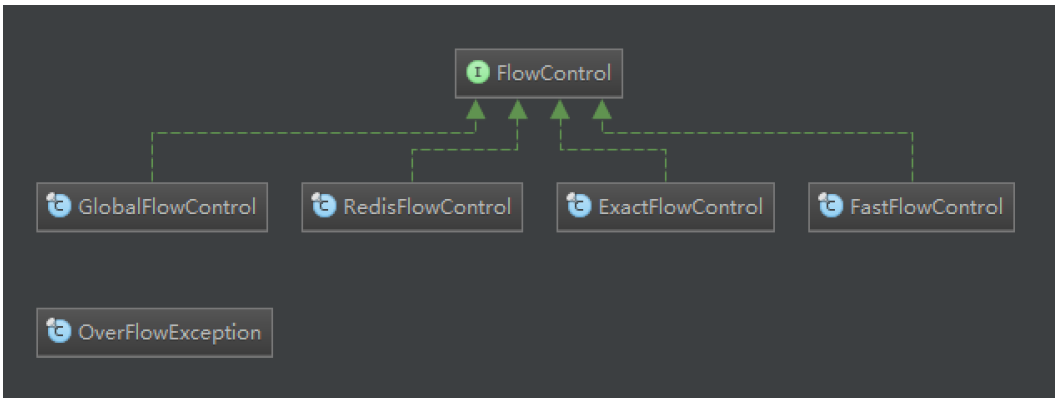


GlobalFlowControl　#单进程全局流控，对MPNS发送给网关的流控(单进程)

RedisFlowControl　#redis 流控(多进程)，用于限制广播消息到客户端的数量（这里代码实现不安全）

FastFlowControl　#快速流控(单进程)，用于限制广播消息到客户端的数量（这里代码实现不安全）

ExactFlowControl　#分段流控，分段统计，把1s 分成 100份，10ms一份，限制10ms内允许的最大数量

GlobalFlowControl

源码说明

```
1  public final class GlobalFlowControl implements FlowControl {
2   private final int limit;
3   private final int maxLimit;
4   private final long duration;
5   private final AtomicInteger count = new AtomicInteger();
6   private final AtomicInteger total = new AtomicInteger();
7   private final long start0 = System.nanoTime();
8   private volatile long start;
9
10   public GlobalFlowControl(int qps) {
11   this(qps, Integer.MAX_VALUE, 1000);//默认最大限制为Integer.MAX_VALUE, 限制时间1S
12   }
13
14   public GlobalFlowControl(int limit, int maxLimit, int duration) {
15   this.limit = limit;
16   this.maxLimit = maxLimit;
17   this.duration = TimeUnit.MILLISECONDS.toNanos(duration); //转换为纳秒
18   }
```

```
19
20  @Override
21  public void reset() {
22  count.set(0); //重新计数
23  start = System.nanoTime(); //重置开始时间
24  }
25
26  @Override
27  public int total() {
28  return total.get();//总共通过数
29  }
30
31  @Override
32  public boolean checkQps() {
33  // 先+1 然后判断是否小于Limit, 小于则通过
34  if (count.incrementAndGet() < limit) {
35  total.incrementAndGet(); //总数+1
36  return true;
37  }
38  //超过最大限制，抛异常
39  if (maxLimit > 0 && total.get() > maxLimit) throw new OverFlowException(t
rue);
40  //超过限制数，且过了限制时间，重新计数
41  if (System.nanoTime() - start > duration) {
42  reset();
43  total.incrementAndGet(); //总数+1
44  return true;
45  }
46  return false; //超过限制数，且没有过限制时间，进行限流(不阻塞)
47  }
48
49  @Override
50  public long getDelay() {
51  return duration - (System.nanoTime() - start); //剩下多少时间
52  }
53
54  @Override
55  public int qps() {
56  // QPS=总数/总时间
57  return (int) (TimeUnit.SECONDS.toNanos(total.get()) / (System.nanoTime()
- start0));
58  }
```

```
59
60  @Override
61  public String report() {
62  return String.format("total:%d, count:%d, qps:%d", total.get(),
count.get(), qps());
63  }
64 }
```

测试

```
1  public class PushClientTestMain2 {
2  public static void main(String[] args) throws Exception {
3  new PushClientTestMain2().testPush();
4  }
5  @Test
6  public void testPush() throws Exception {
7  Logs.init();
8  PushSender sender = PushSender.create();
9  sender.start().join();
10  Thread.sleep(1000);
11  //统计
12  Statistics statistics = new Statistics();
13  FlowControl flowControl = new GlobalFlowControl(1000);// qps=1000
14  //任务执行线程池
15  ScheduledThreadPoolExecutor service = new ScheduledThreadPoolExecutor(4);
16  //每隔1S时间打印流控、统计信息
17  Executors.newSingleThreadScheduledExecutor().scheduleAtFixedRate(() -> {
18  System.out.println("time=" + LocalTime.now()
19  + ", flowControl=" + flowControl.report()
20  + ", statistics=" + statistics
21  );
22  }, 1, 1, TimeUnit.SECONDS);
23
24  for (int k = 0; k < 1000; k++) {// 1000条信息
25  for (int i = 0; i < 1; i++) { //1个用户
26  //任务线程池超过1000 就开始休眠1S
27  while (service.getQueue().size() > 1000) Thread.sleep(1); // 防止内存溢出
28  //消息内容
29  PushMsg msg = PushMsg.build(MsgType.MESSAGE, "this a first push.");
30  msg.setMsgId("msgId_" + i);
31  //消息包装
32  PushContext context = PushContext.build(msg)
33  .setAckModel(AckModel.NO_ACK)
```

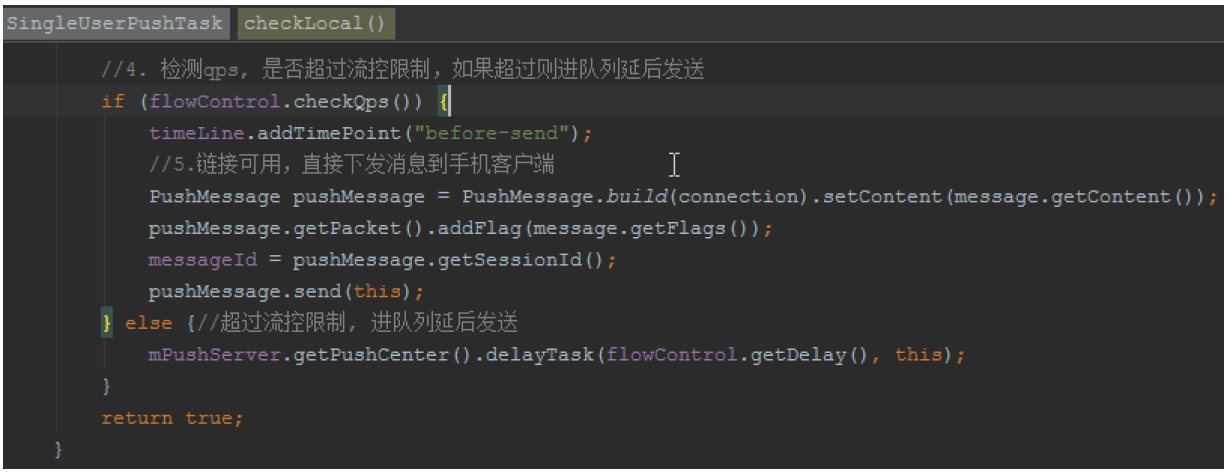
```
34   .setUserId("user-" + i)
35   .setBroadcast(false)
36   .setTimeout(60000)
37   .setCallback(new PushCallback() {
38   @Override
39   public void onResult(PushResult result) {
40   //统计成功、失败、离线、超时次数
41   statistics.add(result.resultCode);
42   }
43   });
44   service.execute(new PushTask(sender, context, service, flowControl, stati
stics));
45   }
46   }
47
48   LockSupport.parkNanos(TimeUnit.SECONDS.toNanos(30000));
49   }
50
51   private static class PushTask implements Runnable {
52   PushSender sender;
53   FlowControl flowControl;
54   Statistics statistics;
55   ScheduledExecutorService executor;
56   PushContext context;
57
58   public PushTask(PushSender sender,
59   PushContext context,
60   ScheduledExecutorService executor,
61   FlowControl flowControl,
62   Statistics statistics) {
63   this.sender = sender;
64   this.context = context;
65   this.flowControl = flowControl;
66   this.executor = executor;
67   this.statistics = statistics;
68   }
69
70   @Override
71   public void run() {
72   if (flowControl.checkQps()) {
73   FutureTask<PushResult> future = sender.send(context);
```

```
74   } else {
75   executor.schedule(this, flowControl.getDelay(), TimeUnit.NANOSECONDS);
76   }
77   }
78   }
79
80   private static class Statistics {
81   final AtomicInteger successNum = new AtomicInteger();
82   final AtomicInteger failureNum = new AtomicInteger();
83   final AtomicInteger offlineNum = new AtomicInteger();
84   final AtomicInteger timeoutNum = new AtomicInteger();
85   AtomicInteger[] counters = new AtomicInteger[]{successNum, failureNum, of
flineNum, timeoutNum};
86
87   private void add(int code) {
88   counters[code - 1].incrementAndGet();
89   }
90
91   @Override
92   public String toString() {
93   return "{" +
94   "successNum=" + successNum +
95   ", offlineNum=" + offlineNum +
96   ", timeoutNum=" + timeoutNum +
97   ", failureNum=" + failureNum +
98   '}';
99   }
100   }
101  }
```

使用说明

```
public final class PushCenter extends BaseService implements MessagePusher {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    private final GlobalFlowControl globalFlowControl = new GlobalFlowControl(
            CC.mp.push.flow_control.global.limit, CC.mp.push.flow_control.global.max, CC.mp.push.flow_control.global.duration
    );
```



RedisFlowControl

源码说明

```
1  public final class RedisFlowControl implements FlowControl {
2   private final BroadcastController controller;
3   private final long start0 = System.nanoTime();
4   private final long duration = TimeUnit.SECONDS.toNanos(1);
5   private final int maxLimit;
6   private int limit;
7   private int count;
8   private int total;
9   private long start;
10   public RedisFlowControl(String taskId) {
11   this(taskId, Integer.MAX_VALUE);//默认最大限制为Integer.MAX_VALUE
12   }
13   public RedisFlowControl(String taskId, int maxLimit) {
14   //操作redis
15   this.controller = new RedisBroadcastController(taskId);
16   this.limit = controller.qps(); //从redis中获取已经设置好的限流数量，默认1000
```

```
this.maxLimit = maxLimit;//最大限制数
}
@Override
public void reset() {
count = 0;//重新计数
start = System.nanoTime();//重置开始时间
}
@Override
public int total() {
return total; //总共通过数
}
@Override
public boolean checkQps() throws OverFlowException {
// 小于Limit限制数，则通过
if (count < limit) {
count++; //当前数+1
total++; //总数+1
return true;
}
//总数超过设定的最大限流数，则抛出异常
if (total() > maxLimit) {
throw new OverFlowException(true);
}
//超过限制数，且过了限制时间，重新计数
if (System.nanoTime() - start > duration) {
reset();
total++; //总数+1
return true;
}
//如果redis流控被取消，抛出异常
if (controller.isCancelled()) {// 这里大概是为了远程控制是否启用redis流控
throw new OverFlowException(true);
} else {
// //从redis中获取限流数量，默认1000
limit = controller.qps();//这里再赋值一遍，是因为远程的limit会实时修改
}
return false; //超过限制数，且没有过限制时间，进行限流(不阻塞)
}
//这里多线程情况下，会有原子性问题
@Override
```

```
57  public void end(Object result) { //每条消息广播成功后，将该消息推送的total归零
58  int t = total;
59  if (total > 0) {
60  total = 0;
61  controller.incSendCount(t); //将"总共通过数"累加到redis中(因为服务是多进程的)
62  }
63  // userId数组
64  if (result != null && (result instanceof String[])) {
65  // 将数组里的userId全部lpush到redis队列中
66  controller.success((String[]) result);
67  }
68  }
69
70  @Override
71  public long getDelay() {
72  return duration - (System.nanoTime() - start); //剩下多少时间
73  }
74
75  @Override
76  public String report() {
77  return String.format("total:%d, count:%d, qps:%d", total, count, qps());
78  }
79  //每次广播消息的QPS(这里计算好像不准确，上面total=0了，star0应该也需重置)
80  @Override
81  public int qps() {
82  // QPS=总数/总时间
83  return (int) (TimeUnit.SECONDS.toNanos(total) / (System.nanoTime() - start0));
84  }
85
86  public BroadcastController getController() {
87  return controller;
88  }
89 }
```

测试

略.....

使用说明

FastFlowControl

```
1  public final class FastFlowControl implements FlowControl {
2   private final int limit;
3   private final int maxLimit;
4   private final long duration;
5   private final long start0 = System.nanoTime();
6   private int count;
7   private int total;
8   private long start;
9
10
11   public FastFlowControl(int limit, int maxLimit, int duration) {
12   this.limit = limit;
13   this.maxLimit = maxLimit;
14   this.duration = TimeUnit.MILLISECONDS.toNanos(duration);
15   }
16
17   public FastFlowControl(int qps) {
18   this(qps, Integer.MAX_VALUE, 1000);
19   }
20
21   @Override
22   public void reset() {
23   count = 0;
24   start = System.nanoTime();
25   }
26
27   @Override
28   public int total() {
```

```
29  return total;
30  }
31
32  @Override
33  public boolean checkQps() {
34  if (count < limit) {
35  count++;
36  total++;
37  return true;
38  }
39
40  if (total > maxLimit) throw new OverFlowException(true);
41
42  if (System.nanoTime() - start > duration) {
43  reset();
44  total++;
45  return true;
46  }
47  return false;
48  }
49
50  @Override
51  public long getDelay() {
52  return duration - (System.nanoTime() - start);
53  }
54
55  @Override
56  public String report() {
57  return String.format("total:%d, count:%d, qps:%d", total, count, qps());
58  }
59
60  @Override
61  public int qps() {
62  return (int) (TimeUnit.SECONDS.toNanos(total) / (System.nanoTime() - star
t0));
63  }
64 }
```

ExactFlowControl

源码说明

```java
//参考 HystrixRollingNumber
public final class ExactFlowControl implements FlowControl {
 private static final long DELAY_100_MS = TimeUnit.MILLISECONDS.toNanos(1);
 private final RollingNumber rollingNumber;
 private final int qps_pre_10_mills;
 private final long start0 = System.nanoTime();

 public ExactFlowControl(int qps) {
 int timeInMilliseconds = 1000;// 1s
  int numberOfBuckets = 100;//把1s 分成 100份，10ms一份， 要计算处 每10ms内允许的最大数量qps_pre_10_mills

  int _10_mills = timeInMilliseconds / numberOfBuckets;//=10

  double real_qps_pre_10_mills = (qps / 1000f) * _10_mills;

  if (real_qps_pre_10_mills < 1) {//qps < 100;
  numberOfBuckets = 1;
  real_qps_pre_10_mills = qps;
  }

  this.qps_pre_10_mills = (int) real_qps_pre_10_mills;
  this.rollingNumber = new RollingNumber(timeInMilliseconds, numberOfBuckets);
  }

  @Override
  public void reset() {

  }

  @Override
  public int total() { //所以bucket的总和
  return (int) rollingNumber.getCumulativeSum(SUCCESS);
  }

  @Override
  public boolean checkQps() throws OverFlowException {
  //
  if (rollingNumber.getValueOfLatestBucket(SUCCESS) < qps_pre_10_mills) {
  rollingNumber.increment(SUCCESS);
  return true;
```

```
41  }
42  return false;
43  }
44
45  @Override
46  public long getDelay() {
47  return DELAY_100_MS;
48  }
49
50  @Override
51  public int qps() {
52  // QPS=总数/总时间
53  return (int) rollingNumber.getRollingSum(SUCCESS);
54  }
55
56  @Override
57  public String report() {
58  return String.format("total:%d, count:%d, qps:%d, avg_qps:%d",
59  total(), rollingNumber.getValueOfLatestBucket(SUCCESS), qps(),
60  TimeUnit.SECONDS.toNanos(total()) / (System.nanoTime() - start0));
61  }
62 }
```