## **序**

本文主要研究一下hystrix的queueSizeRejectionThreshold参数

## **HystrixThreadPoolProperties**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/HystrixThreadPoolProperties.java

/\*\*

\* Queue size rejection threshold is an artificial "max" size at which rejections will occur even **if** {@link *#maxQueueSize} has not been reached. This is done because the {@link #maxQueueSize} of a*

\* {@link BlockingQueue} **can** **not** **be** **dynamically** **changed** **and** **we** **want** **to** **support** **dynamically** **changing** **the** **queue** **size** **that** **affects** **rejections**.

\* <p>

\* This is used **by** {@link HystrixCommand} **when** **queuing** **a** **thread** **for** **execution**.

\*

\* @**return** {@code HystrixProperty<Integer>}

\*/

public HystrixProperty<Integer> queueSizeRejectionThreshold() {

return queueSizeRejectionThreshold;

}

设计这个参数的原因在于BlockingQueue的大小不能动弹调整，因此使用这个参数来满足动弹调整的需求

## **HystrixThreadPool.HystrixThreadPoolDefault.isQueueSpaceAvailable**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/HystrixThreadPool.java

**static** **class** **HystrixThreadPoolDefault** **implements** **HystrixThreadPool** {

**private** **static** **final** Logger logger = LoggerFactory.getLogger(HystrixThreadPoolDefault.class);

**private** **final** HystrixThreadPoolProperties properties;

**private** **final** BlockingQueue<Runnable> queue;

**private** **final** ThreadPoolExecutor threadPool;

**private** **final** HystrixThreadPoolMetrics metrics;

**private** **final** **int** queueSize;

*//......*

*/\*\**

*\* Whether the threadpool queue has space available according to the <code>queueSizeRejectionThreshold</code> settings.*

*\**

*\* Note that the <code>queueSize</code> is an final instance variable on HystrixThreadPoolDefault, and not looked up dynamically.*

*\* The data structure is static, so this does not make sense as a dynamic lookup.*

*\* The <code>queueSizeRejectionThreshold</code> can be dynamic (up to <code>queueSize</code>), so that should*

*\* still get checked on each invocation.*

*\* <p>*

*\* If a SynchronousQueue implementation is used (<code>maxQueueSize</code> <= 0), it always returns 0 as the size so this would always return true.*

*\*/*

@Override

**public** **boolean** isQueueSpaceAvailable() {

**if** (queueSize <= 0) {

*// we don't have a queue so we won't look for space but instead*

*// let the thread-pool reject or not*

**return** **true**;

} **else** {

**return** threadPool.getQueue().size() < properties.queueSizeRejectionThreshold().get();

}

}

}

这里判断threadPool的queueSize是否小于queueSizeRejectionThreshold，来判断是否有空余空间

## **HystrixContextScheduler**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/strategy/concurrency/HystrixContextScheduler.java

**private** **class** **HystrixContextSchedulerWorker** **extends** **Worker** {

**private** **final** Worker worker;

**private** **HystrixContextSchedulerWorker**(Worker actualWorker) {

**this**.worker = actualWorker;

}

**@Override**

**public** **void** **unsubscribe**() {

worker.unsubscribe();

}

**@Override**

**public** **boolean** **isUnsubscribed**() {

**return** worker.**isUnsubscribed**();

}

**@Override**

**public** Subscription **schedule**(Action0 action, **long** delayTime, TimeUnit unit) {

**if** (threadPool != **null**) {

**if** (!threadPool.isQueueSpaceAvailable()) {

**throw** **new** RejectedExecutionException("Rejected command because thread-pool queueSize is at rejection threshold.");

}

}

**return** worker.schedule(**new** HystrixContexSchedulerAction(concurrencyStrategy, action), delayTime, unit);

}

**@Override**

**public** Subscription **schedule**(Action0 action) {

**if** (threadPool != **null**) {

**if** (!threadPool.isQueueSpaceAvailable()) {

**throw** **new** RejectedExecutionException("Rejected command because thread-pool queueSize is at rejection threshold.");

}

}

**return** worker.schedule(**new** HystrixContexSchedulerAction(concurrencyStrategy, action));

}

}

HystrixContextSchedulerWorker的schedule方法在堆action进行调度之前，会先判断threadPool.isQueueSpaceAvailable()，如果超出限制，则抛出RejectedExecutionException异常

## **HystrixContextScheduler**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/strategy/concurrency/HystrixContextScheduler.java

**public** **HystrixContextScheduler**(HystrixConcurrencyStrategy concurrencyStrategy, HystrixThreadPool threadPool, Func0<Boolean> shouldInterruptThread) {

**this**.concurrencyStrategy = concurrencyStrategy;

**this**.threadPool = threadPool;

**this**.actualScheduler = **new** ThreadPoolScheduler(threadPool, shouldInterruptThread);

}

**@Override**

**public** Worker **createWorker**() {

**return** **new** HystrixContextSchedulerWorker(actualScheduler.createWorker());

}

**private** **static** **class** **ThreadPoolScheduler** **extends** **Scheduler** {

**private** **final** HystrixThreadPool threadPool;

**private** **final** Func0<Boolean> shouldInterruptThread;

**public** **ThreadPoolScheduler**(HystrixThreadPool threadPool, Func0<Boolean> shouldInterruptThread) {

**this**.threadPool = threadPool;

**this**.shouldInterruptThread = shouldInterruptThread;

}

**@Override**

**public** Worker **createWorker**() {

**return** **new** ThreadPoolWorker(threadPool, shouldInterruptThread);

}

}

这里的worker为HystrixContextSchedulerWorker，它内部使用的是ThreadPoolScheduler创建的worker

## **ThreadPoolWorker**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/strategy/concurrency/HystrixContextScheduler.java

*/\*\**

*\* Purely for scheduling work on a thread-pool.*

*\* <p>*

*\* This is not natively supported by RxJava as of 0.18.0 because thread-pools*

*\* are contrary to sequential execution.*

*\* <p>*

*\* For the Hystrix case, each Command invocation has a single action so the concurrency*

*\* issue is not a problem.*

*\*/*

**private** **static** **class** **ThreadPoolWorker** **extends** **Worker** {

**private** **final** HystrixThreadPool threadPool;

**private** **final** CompositeSubscription subscription = **new** CompositeSubscription();

**private** **final** Func0<Boolean> shouldInterruptThread;

**public** **ThreadPoolWorker**(HystrixThreadPool threadPool, Func0<Boolean> shouldInterruptThread) {

**this**.threadPool = threadPool;

**this**.shouldInterruptThread = shouldInterruptThread;

}

**@Override**

**public** **void** **unsubscribe**() {

subscription.unsubscribe();

}

**@Override**

**public** **boolean** **isUnsubscribed**() {

**return** subscription.**isUnsubscribed**();

}

**@Override**

**public** Subscription **schedule**(**final** Action0 action) {

**if** (subscription.isUnsubscribed()) {

*// don't schedule, we are unsubscribed*

**return** Subscriptions.**unsubscribed**();

}

*// This is internal RxJava API but it is too useful.*

ScheduledAction sa = **new** ScheduledAction(action);

subscription.add(sa);

sa.addParent(subscription);

ThreadPoolExecutor executor = (ThreadPoolExecutor) threadPool.getExecutor();

FutureTask<?> f = (FutureTask<?>) executor.submit(sa);

sa.add(**new** FutureCompleterWithConfigurableInterrupt(f, shouldInterruptThread, executor));

**return** sa;

}

**@Override**

**public** Subscription **schedule**(Action0 action, **long** delayTime, TimeUnit unit) {

**throw** **new** IllegalStateException("Hystrix does not support delayed scheduling");

}

}

ThreadPoolWorker的schedule方法，就是将ScheduledAction提交到ThreadPoolExecutor去执行

## **HystrixConcurrencyStrategy.getThreadPool**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/strategy/concurrency/HystrixConcurrencyStrategy.java

**public** ThreadPoolExecutor getThreadPool(**final** HystrixThreadPoolKey threadPoolKey, HystrixThreadPoolProperties threadPoolProperties) {

**final** ThreadFactory threadFactory = getThreadFactory(threadPoolKey);

**final** boolean allowMaximumSizeToDivergeFromCoreSize = threadPoolProperties.getAllowMaximumSizeToDivergeFromCoreSize().get();

**final** int dynamicCoreSize = threadPoolProperties.coreSize().get();

**final** int keepAliveTime = threadPoolProperties.keepAliveTimeMinutes().get();

**final** int maxQueueSize = threadPoolProperties.maxQueueSize().get();

**final** BlockingQueue<Runnable> workQueue = getBlockingQueue(maxQueueSize);

**if** (allowMaximumSizeToDivergeFromCoreSize) {

**final** int dynamicMaximumSize = threadPoolProperties.maximumSize().get();

**if** (dynamicCoreSize > dynamicMaximumSize) {

logger.error("Hystrix ThreadPool configuration at startup for : " + threadPoolKey.name() + " is trying to set coreSize = " +

dynamicCoreSize + " and maximumSize = " + dynamicMaximumSize + ". Maximum size will be set to " +

dynamicCoreSize + ", the coreSize value, since it must be equal to or greater than the coreSize value");

**return** **new** ThreadPoolExecutor(dynamicCoreSize, dynamicCoreSize, keepAliveTime, TimeUnit.MINUTES, workQueue, threadFactory);

} **else** {

**return** **new** ThreadPoolExecutor(dynamicCoreSize, dynamicMaximumSize, keepAliveTime, TimeUnit.MINUTES, workQueue, threadFactory);

}

} **else** {

**return** **new** ThreadPoolExecutor(dynamicCoreSize, dynamicCoreSize, keepAliveTime, TimeUnit.MINUTES, workQueue, threadFactory);

}

}

threadPoolExecutor的workQueue的大小由参数threadPoolProperties.maxQueueSize()来设置，默认是-1。如果要修改default线程池队列的大小，则需要设置hystrix.threadpool.default.maxQueueSize属性。

## **HystrixConcurrencyStrategy.getBlockingQueue**

hystrix-core-1.5.12-sources.jar!/com/netflix/hystrix/strategy/concurrency/HystrixConcurrencyStrategy.java

/\*\*

\* **Factory** **method** to provide instance **of** {@code **BlockingQueue**<**Runnable**>} used **for** each {@link **ThreadPoolExecutor**} **as** constructed **in** {@link *#getThreadPool}.*

\* <p>

\* **Note**: **The** maxQueueSize value **is** provided so any **type** **of** queue can be used but typically an implementation such **as** {@link **SynchronousQueue**} **without** a queue (just a handoff) **is** preferred **as**

\* queueing **is** an anti-pattern to be purposefully avoided **for** latency tolerance reasons.

\* <p>

\* <b>**Default** **Implementation**</b>

\* <p>

\* **Implementation** returns {@link **SynchronousQueue**} **when** maxQueueSize <= 0 **or** {@link **LinkedBlockingQueue**} **when** maxQueueSize > 0.

\*

\* @param maxQueueSize

\* **The** max size **of** the queue requested via properties (**or** system default **if** no properties set).

\* @**return** instance **of** {@code **BlockingQueue**<**Runnable**>}

\*/

public **BlockingQueue**<**Runnable**> getBlockingQueue(int maxQueueSize) {

/\*

\* **We** are **using** **SynchronousQueue** **if** maxQueueSize <= 0 (meaning a queue **is** **not** wanted).

\* <p>

\* **SynchronousQueue** will **do** a handoff **from** calling thread to worker thread **and** **not** allow queuing which **is** what we want.

\* <p>

\* **Queuing** results **in** added latency **and** would only occur **when** the thread-pool **is** full at which point there are latency issues

\* **and** rejecting **is** the preferred solution.

\*/

**if** (maxQueueSize <= 0) {

**return** new **SynchronousQueue**<**Runnable**>();

} **else** {

**return** new **LinkedBlockingQueue**<**Runnable**>(maxQueueSize);

}

}

如果是-1的话，创建的是SynchronousQueue，大于0则根据其大小创建LinkedBlockingQueue

## **小结**

hystrix提供了queueSizeRejectionThreshold属性(hystrix.threadpool.default.queueSizeRejectionThreshold)来动态控制线程池队列的上限，而线程池本身队列的大小，则是由maxQueueSize属性(hystrix.threadpool.default.maxQueueSize)来决定，默认为-1，创建的队列是SynchronousQueue，如果设置大于0则根据其大小创建LinkedBlockingQueue