支持服务隔离的线程池

一、背景

1.1. 背景需求

通常通过一个线程池来执行多种类型的任务，但是不同类型的任务有不同的优先级，以及不同类型任务的服务隔离，都难以支持。比如 tip 中不同业务方的 API 肯定有不同的优先级，我们肯定不希望大量的淘宝客 API 调用消耗了系统大量的资源，而导致比如交易 API 不可用，或响应慢。这就需要将不同类型的业务隔离起来，对不同的业务配置不同的资源配置，并能控制他们的资源消耗在配置范围之内。

但是针对不同类型的任务，使用不同的线程池，极容易导致资源的浪费，某种类型的任务线程池繁忙处理不过来，而有些类型的任务线程池空闲，而导致资源的浪费，并且不易管理。

我们希望通过共享单个线程池，来支持不同类型的任务的服务隔离，资源配置（配置单个类型的任务的资源），资源限制，当让如果能做到动态资源配置同时兼顾任务类型的优先级，那就完美了。

1.2. 名词解释：

名词	解释
TaskDomain	任务领域，即 任务类型
ResourceConfiguration	资源配置，配置线程，队列大小
TaskDomainResource	任务领域的资源，线程，队列
TaskDomainResourceController	任务领域资源控制器，管理资源的申请，释放
Dispatcher	任务领域排队的任务分发器，负责在资源许可的情况执行处于排队的任务
ThreadPool	线程池，负责管理所有的线程
Watchdog	看门狗，负责清理超时执行的任务
TaskDomainRuntime	任务领域运行时，运行时控制

二、详细设计

2.1. 资源控制

在本系统中，主要考虑两种资源，线程(thread)，队列(queue)，每个一个 TaskDomain 都有自己的 ResourceConfiguration 和相对于配置的 TaskDomainResource。TaskDomainResourceController 控制 TaskDomain 的资源的申请，释放。执行一个任务的流程如下图：

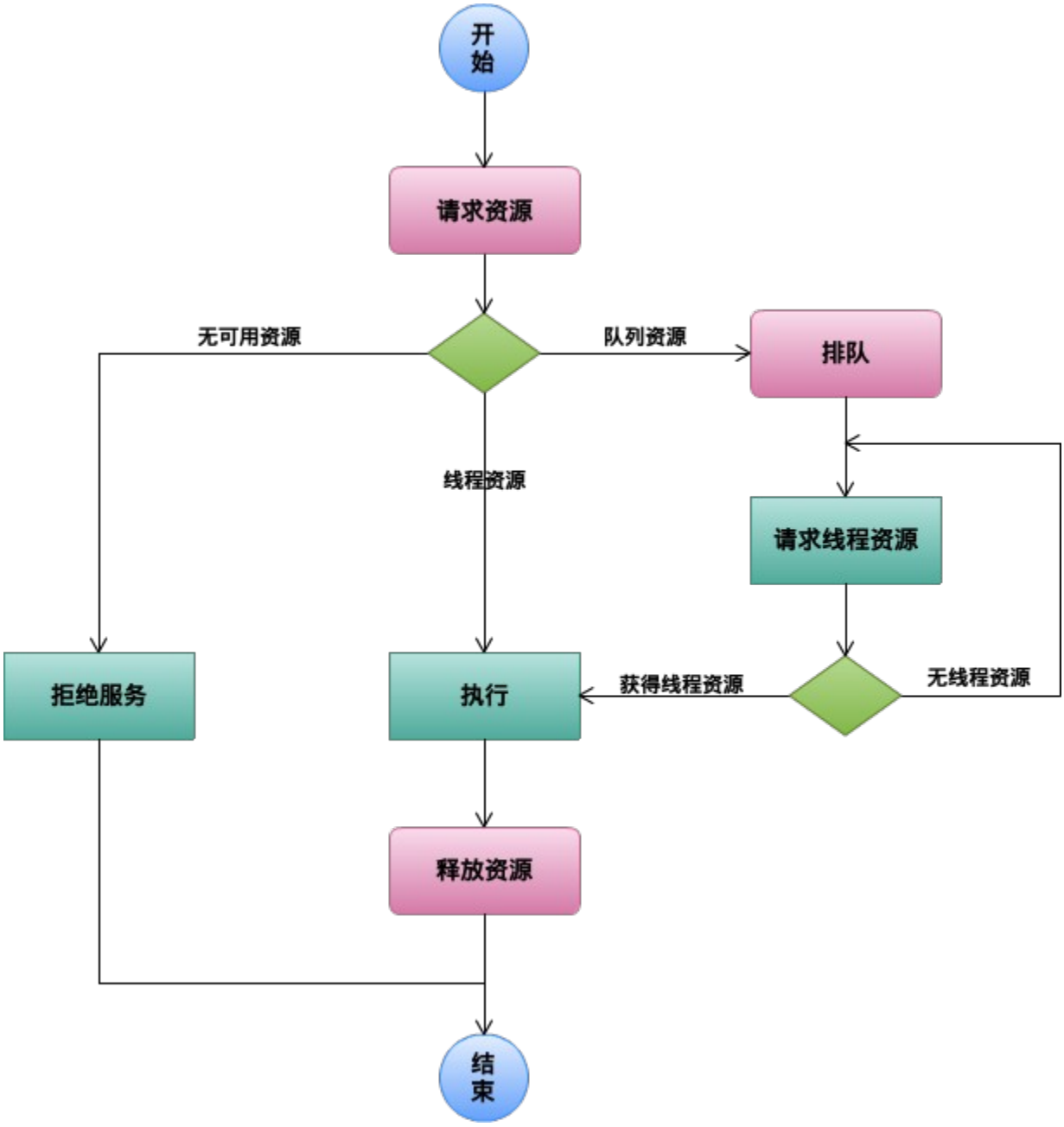

图 2-1 任务执行流程

资源控制模式(ResourceControlMode)

模式	**解释**
MAX	限制最多使用多少，Default Task Domain 的线程控制模式一定是 MAX 模式
RESERVED	保证最少使用多少，即使不需要这么多，也会被保留，但是在自己的 RESERVED 资源用完后，可以向 Default Task Domain 借用资源。

所有的 TaskDomain 共享一个线程池，线程池中线程根据 TaskDomain 的 ResourceConfiguration 划分给不同的 TaskDomain，剩于的资源(thread，queue)划分给 Default Task Domain。共享的线程池，不提供任务排队的功能，每个 TaskDomain 有自己的队列，这样考虑主要是为了避免不同 TaskDomain 的任务的优先级的问题[如果共享一个队列，就只能靠 FIFO 了，但是 FIFO 不能满足，TaskDomain 内部是必须 FIFO 的]。

TaskDomainRuntime 负责接受任务，申请资源，提交任务 threadpool 执行，或是提交任务到等待队列，并在有线程资源可用，提交排队的任务到 threadpool 执行。

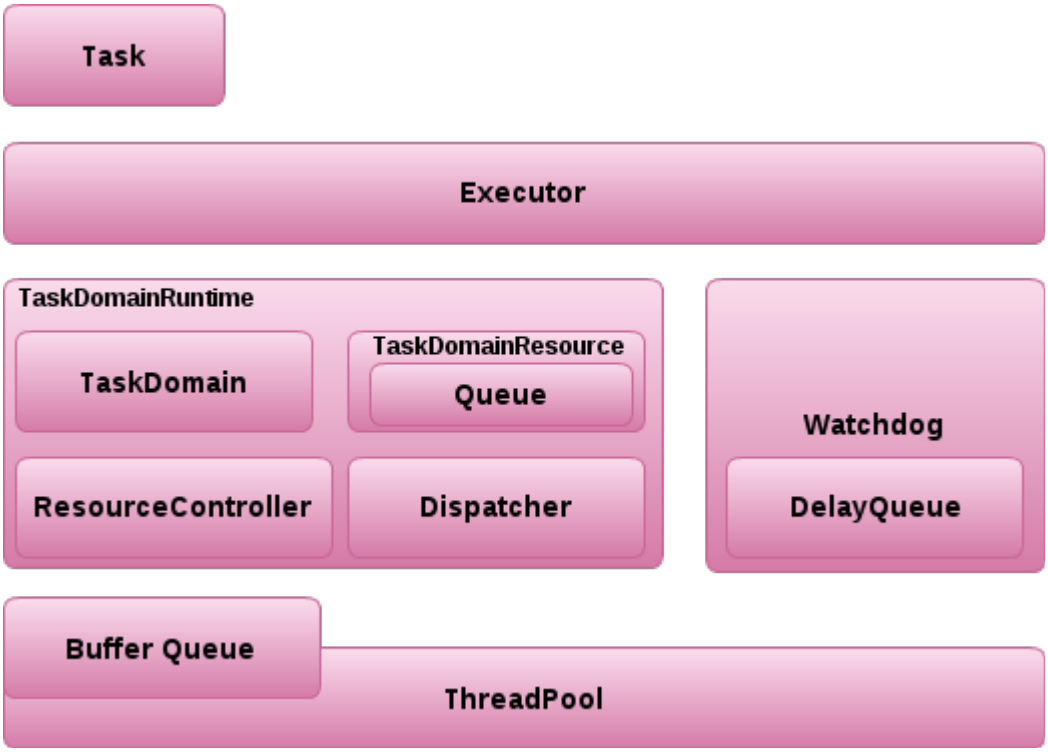

图 2-2 整体结构

MAX 模式控制

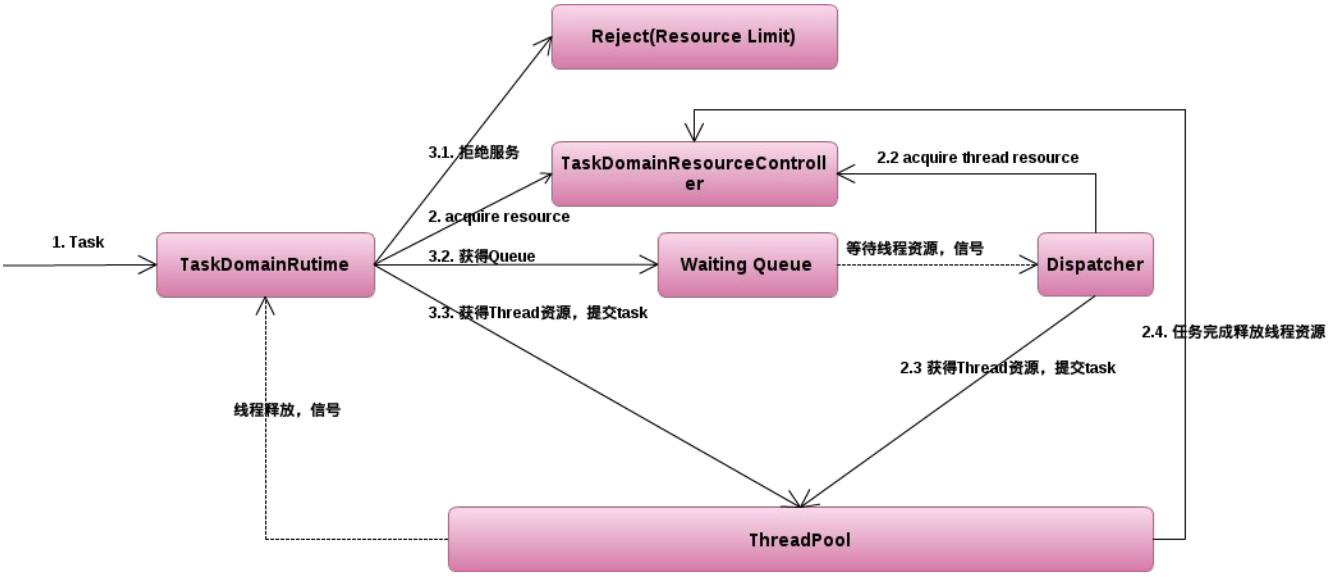

图 2-3 *MAX* 模式控制

注释：

MAX 模式的 TaskDomain 在自己的资源耗尽时，直接拒绝服务。

RESERVED 模式

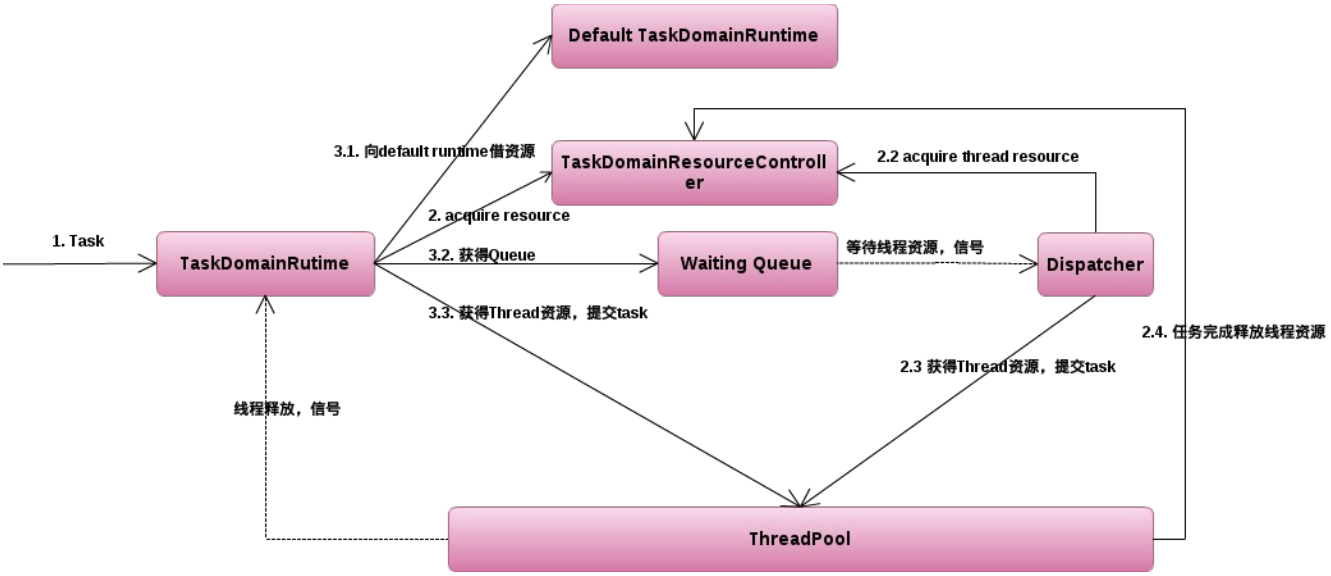

图 2-4 *RESERVED* 模式控制

注释：

RESERVED 模式的 TaskDomain 在自己的资源耗尽时，可以尝试向 Default TaskDomain 借用资源，如果与此同时 Default TaskDomain 资源也耗尽，就只有拒绝服务。

2. TaskDomain 任务排队处理

每个 TaskDomain 有自己的任务队列，和一个后台 Dispatcher 线程，不停的从任务队列却排队的任务，并在有可用线程资源时，执行排队的任务。

TaskDomain 在无法获得线程资源，退而求其次尝试获取队列资源，如果获得了队列资源，将任务排入队列，后台 Dispatcher 线程会被启动，尝试处理队列中的。

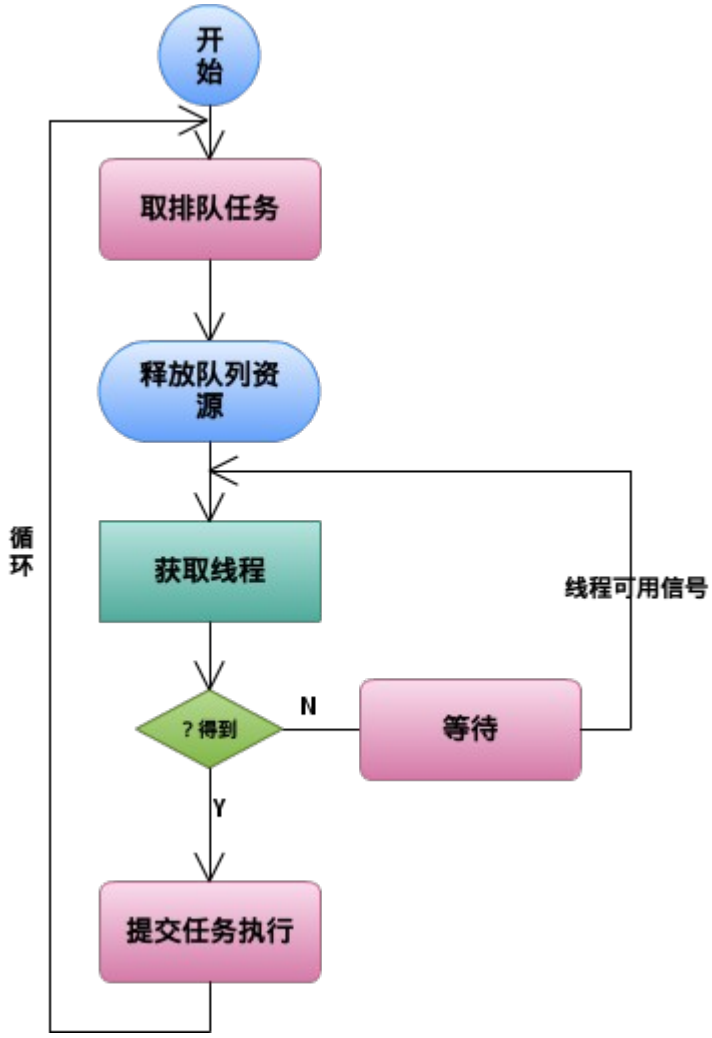

图 2-5 Dispatcher

Dispatcher 循环:

1. 从 queue 中取排队的任务，阻塞直到有排队的任务。
2. 尝试从本 Task Domain 获取线程资源，如果有可用资源 进入 3，无可用资源进入 4。
3. 提交任务到线程池执行，循环 1。
4. 等待有可用线程资源， 如果被唤醒回到 2 重新执行。

3. 任务超时检测及处理（ Watchdog 看门狗）

所有 TaskDomain 公有一个 Watchdog 看门狗，检测及处理超时任务。TaskDomainRuntime 在提交任务给线程池的同时，如果任务设置了超时时间，或是 TaskDomainRuntime 有默认的超时时间设置，或 Executor 强制的超时时间设置，就会构造一个 WatchedTask 提交给 Watchdog 的 DelayQueue 队列，Watchdog 的线程会阻塞直到 DelayQueue 中至少有一个 WatchedTask 超时，Watchdog 将超时的代码任务执行的 Future 取消。

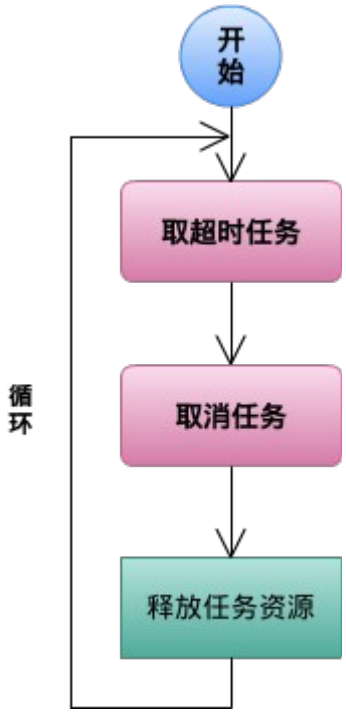

图 2-6 Watchdog 循环

三、高级特性

1. 支持资源动态在各个 TaskDomain 里拆借(资源配置动态调整)。想像如下的场景，系统中配置了两个 TaskDomain:

```
TaskDomain A:
            资源控制模型: MAX
            线程:          16  (MAX, 最多 16)
            队列:          32  (MAX, 最大 32)
TaskDomain B:
            资源控制模型: RESERVED
            线程:          32  (RESERVED, 最多 32)
            队列:          64  (MAX, 最大 64)

Default TaskDomain:
            资源控制模型: MAX
            线程:          16  (MAX, 最多 16)
            队列:          32  (MAX, 最大 32)
```

静态配置，任务 TaskDomain B 是重要的，需要重点保护的， TaskDomain A 非重要的。但是如果系统在某些时刻，如果 TaskDomain B 的请求并不大，而 TaskDomain A 的请求却非常繁忙，这样就会导致 系统为 TaskDomain B 保留的大量资源和 Default TaskDomain 的资源被浪费。此时系统有大量的空闲的资源，却不能为 TaskDomain A 的请求提供正常的服务。

通过监控每一个 TaskDomain 的资源使用率，线程， 队列，可以从资源使用率的 TaskDomain 释放一些资源到其他繁忙的 TaskDomain，或是 Default TaskDomain。

2. 动态强制释放一个 TaskDomain 的所有资源。想象一下这样的场景，某种 TaskDomain 的后端服务完全不可用，这时可以强制将该 TaskDomain 的所有资源释放给其他 TaskDomain 或 Default TaskDomain。(这样的场景发生在，如后某个 ISP 问题，有时会封 API，但是目前封 API，并不会释放给该 API 或业务保留的资源)。