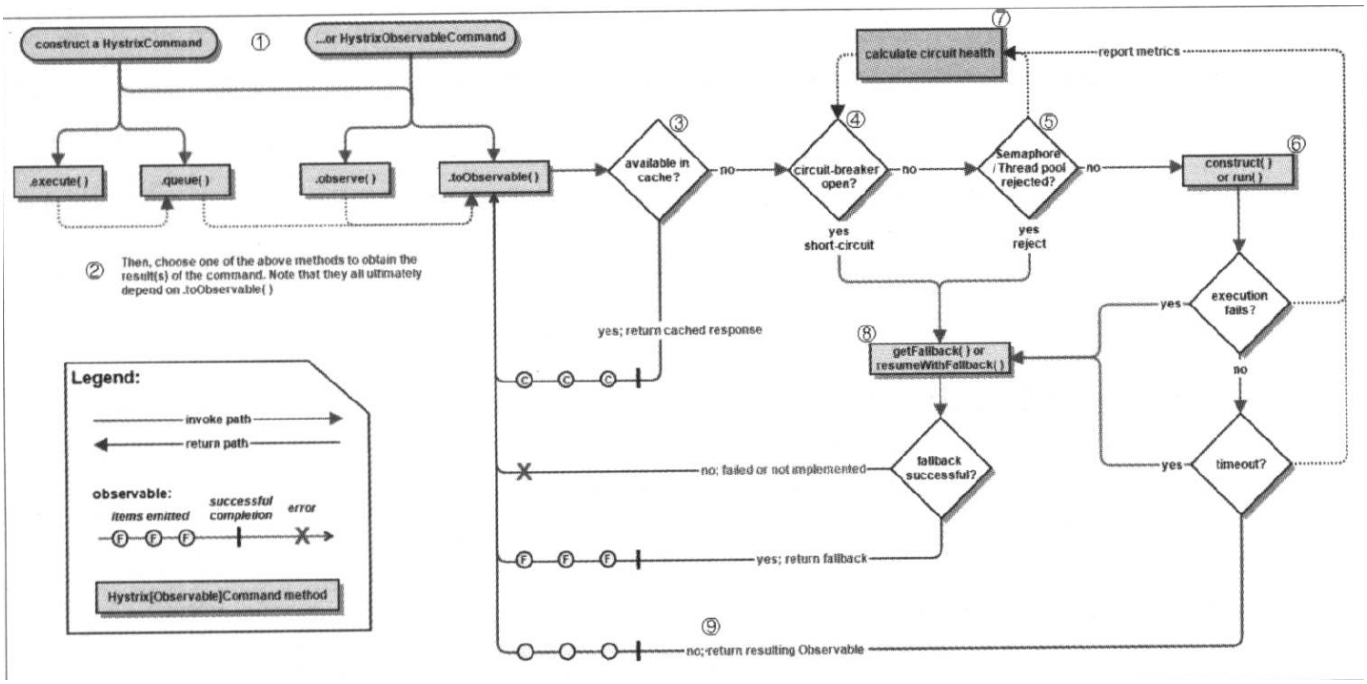
Hystrix

工作流程



上图描述了Hystrix的工作流程，下面我们对其进行进一步的解释：

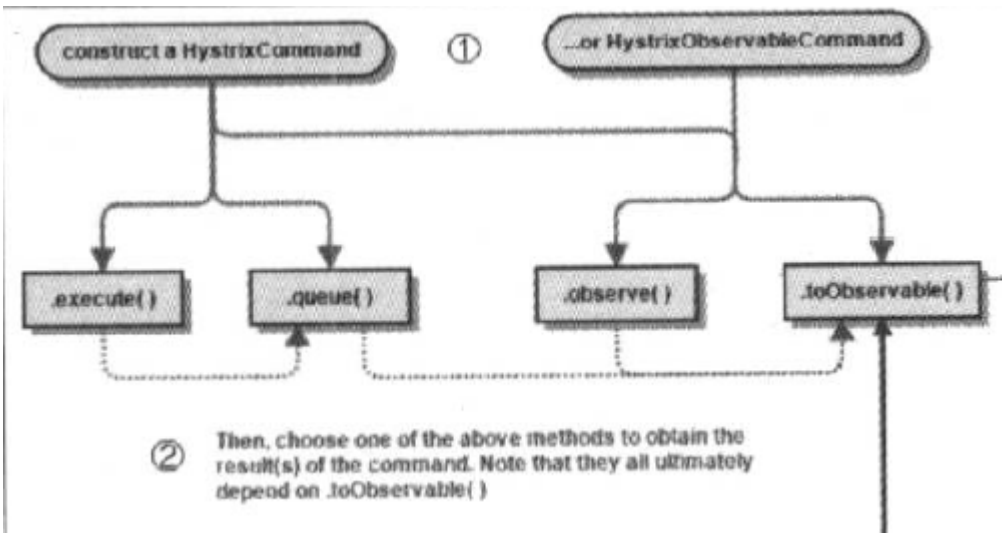
1. 创建HystrixCommand或者HystrixObservableCommand对象

构建HystrixCommand和HystrixObservableCommand对象，用于表示对依赖服务的请求操作，同时传递所有需要的参数。而两者的区别如下：

- HystrixCommand:用在依赖的服务返回单个操作结果的时候。
- HystrixObservableCommand:用在依赖的服务返回多个结果的时候。

2. 命令执行

根据图中我们看到，总共具有4中执行方式，大图如下：



我们可以看到HystrixCommand具有execute()和queue()方法，HystrixObservableCommand具有observe()和toObservable()方法。

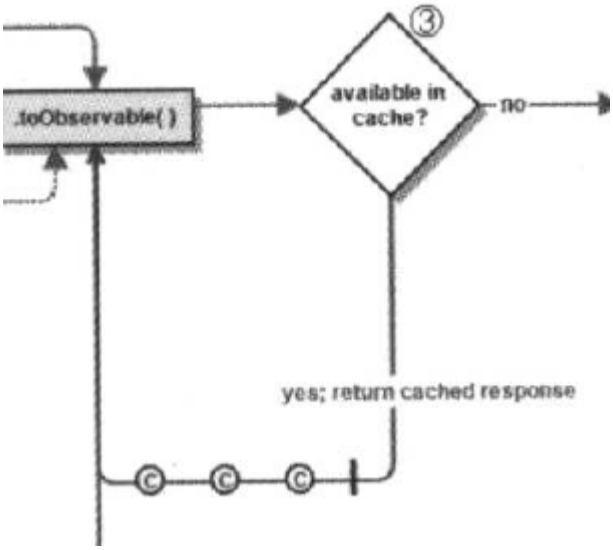
下面我们对其进行详细的描述：

- execute():同步执行，从依赖的服务返回单一的结果对象，或是在发生错误时抛出异常。
- queue():异步执行，直接返回一个Future对象，其中包含了服务执行结束时要返回的单一对象。
- observe():返回Observable对象，它代表了操作的多个结果，它是一个Hot Observable。
- toObservable():同样返回Observable对象，代表了操作的多个结果，返回到是一个Cold Observable。

事实上所有的的执行逻辑都依赖于toObservable()方法。

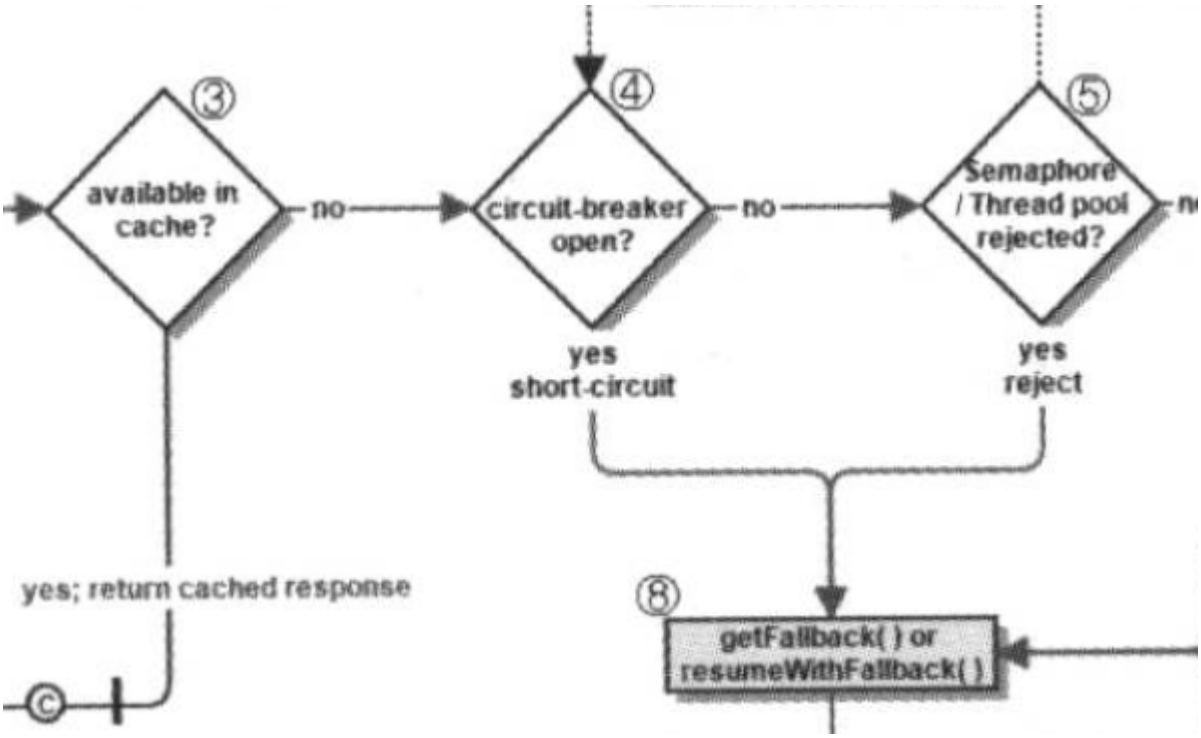
3. 结果是否被缓存

注意，从这里开始是execute()、queue()和observe()、toObservable()的执行逻辑。



若当前命令的请求缓存功能是启用的，并且该命令缓存命中，缓存的结果会立刻以Observable对象的形式返回。

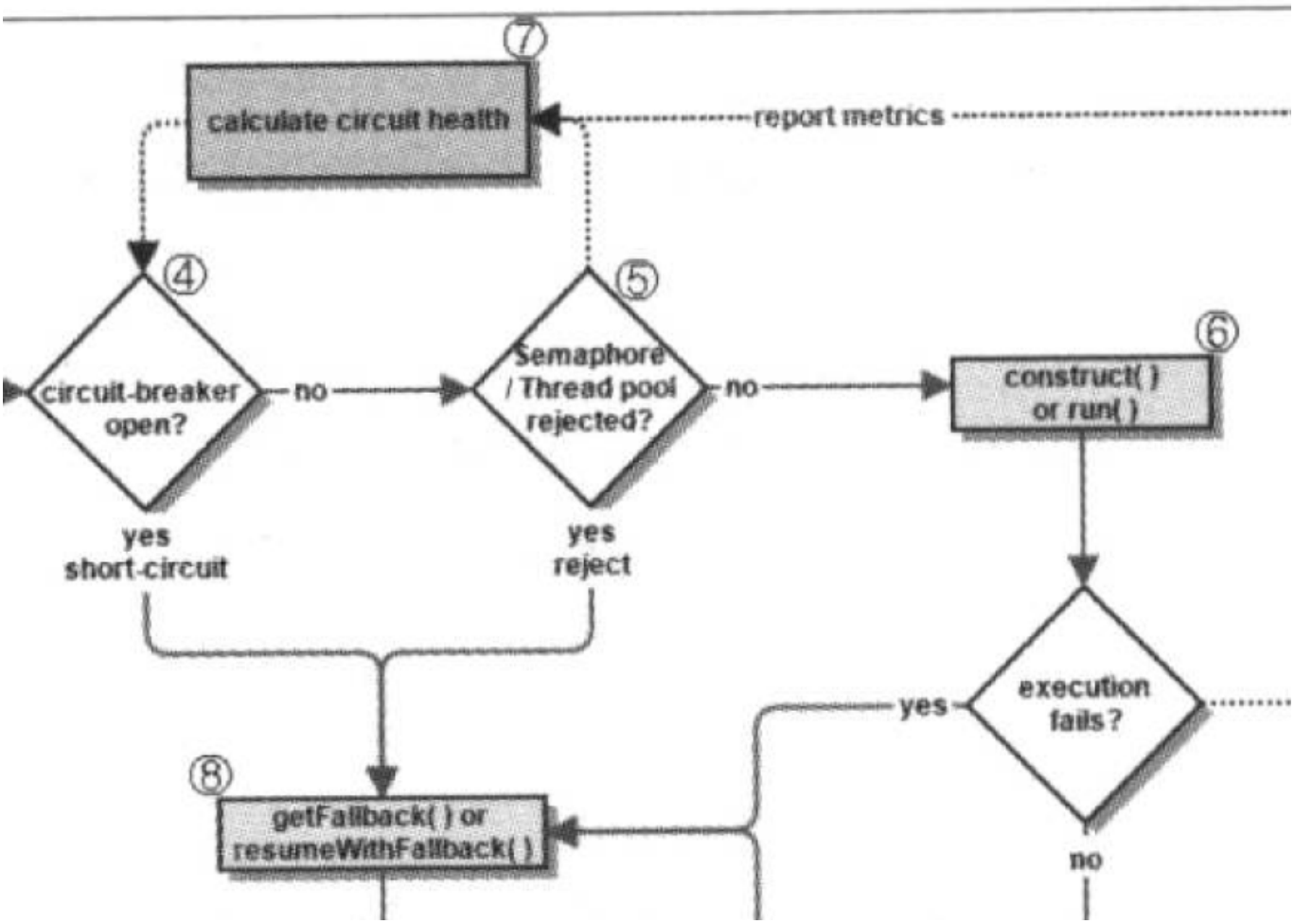
4. 断路器是否打开



在命令结果没有缓存命中的时候，Hystrix在执行命令前就要检查断路器是否是打开状态。

可以看到，如果断路器是打开的就跳到第八步，否则跳到第五步。

5. 线程池/信号量/请求队列是否占满



如果与命令相关的线程池和请求队列，或者信号量已经被占满，那么会执行第八步。

需要注意的是，这里Hystrix所判断的线程并非容器的线程池，而是每个依赖服务的专有线程池。Hystrix为了保证不会因为某个依赖服务的问题影响到其他依赖服务而采用“舱壁模式”来隔离每个依赖服务。

6. HystrixObservableCommand.construct()或HystrixCommand.run()

Hystrix会根据我们编写的方法来决定采用什么样的方式去请求依赖服务。

- HystrixCommand.run(): 返回单一结果，或抛出异常。
- HystrixObservableCommand.construct():返回一个Observable对象来发射多个结果，或通过onError来发送错误通知。