

## 18 | 反应式编程框架设计：如何使程序调用不阻塞等待，立即响应？

2020-01-01 李智慧

后端技术面试38讲

[进入课程 >](#)

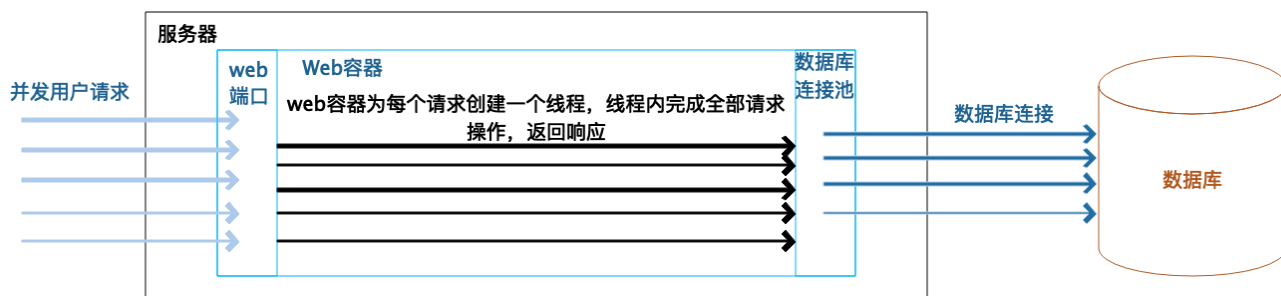


讲述：李智慧

时长 12:02 大小 9.65M



我们在专栏 [第 1 篇](#) 就讨论了为什么在高并发的情况下，程序会崩溃。主要原因是，在高并发的情况下，有大量用户请求需要程序计算处理，而目前的处理方式是，为每个用户请求分配一个线程，当程序内部因为访问数据库等原因造成线程阻塞时，线程无法释放去处理其他请求，这样就会造成请求堆积，不断消耗资源，最终导致程序崩溃。



这是传统的 Web 应用程序运行期的线程特性。对于一个高并发的应用系统来说，总是同时有很多个用户请求到达系统的 Web 容器。Web 容器为每个请求分配一个线程进行处理，线程在处理过程中，如果遇到访问数据库或者远程服务等操作，就会进入阻塞状态，这个时候，如果数据库或者远程服务响应延迟，就会出现程序内的线程无法释放的情况，而外部的请求不断进来，导致计算机资源被快速消耗，最终程序崩溃。

那么有没有不阻塞线程的编程方法呢？

## 反应式编程

答案就是反应式编程。反应式编程本质上是一种异步编程方案，在多线程（协程）、异步方法调用、异步 I/O 访问等技术基础之上，提供了一整套与异步调用相匹配的编程模型，从而实现程序调用非阻塞、即时响应等特性，即开发出一个反应式的系统，以应对编程领域越来越高的并发处理需求。

人们还提出了一个反应式宣言，认为反应式系统应该具备如下特质：

**即时响应**，应用的调用者可以即时得到响应，无需等到整个应用程序执行完毕。也就是说应用调用是非阻塞的。

**回弹性**，当应用程序部分功能失效的时候，应用系统本身能够进行自我修复，保证正常运行，保证响应，不会出现系统崩溃和宕机的情况。

**弹性**，系统能够对应用负载压力做出响应，能够自动伸缩以适应应用负载压力，根据压力自动调整自身的处理能力，或者根据自身的处理能力，调整进入系统中的访问请求数量。

**消息驱动**，功能模块之间，服务之间，通过消息进行驱动，完成服务的流程。

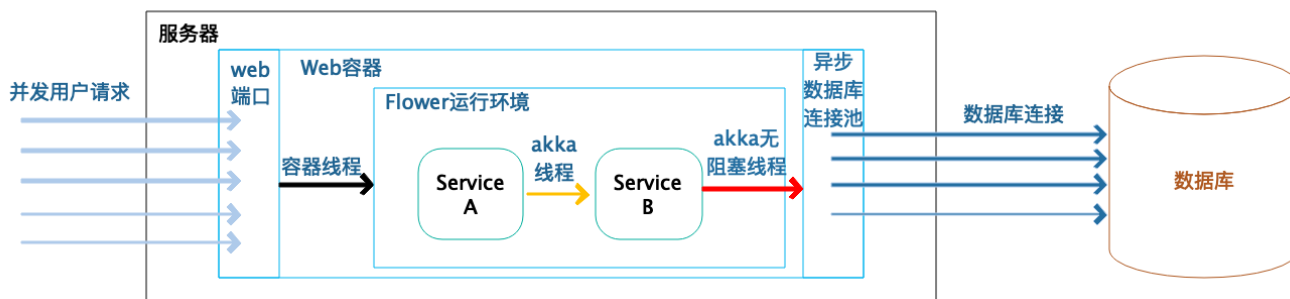
目前主流的反应式编程框架有 RxJava、Reactor 等，它们的主要特点是基于**观察者设计模式**的异步编程方案，编程模型采用函数式编程。

观察者模式和函数式编程有自己的优势，但是反应式编程并不是必须用观察者模式和函数式编程。Flower 就是一个纯消息驱动，完全异步，支持命令式编程的反应式编程框架。

下面我们就看看 Flower 如何实现异步无阻塞的调用，以及 Flower 这个框架设计使用了什么样的设计原则与模式。

## 反应式编程框架 Flower 的基本原理

一个使用 Flower 框架开发的典型 Web 应用的线程特性如下图所示：



当并发用户到达应用服务器的时候，Web 容器线程不需要执行应用程序代码，它只是将用户的 HTTP 请求变为请求对象，将请求对象异步交给 Flower 框架的 Service 去处理，自身立刻就返回。因为容器线程不做太多的工作，所以只需极少的容器线程就可以满足高并发的用户请求，用户的请求不会被阻塞，不会因为容器线程不够而无法处理。相比传统的阻塞式编程，Web 容器线程要完成全部的请求处理操作，直到返回响应结果才能释放线程；**使用 Flower 框架只需要极少的容器线程就可以处理较多的并发用户请求，而且容器线程不会阻塞。**

用户请求交给基于 Flower 框架开发的业务 Service 对象以后，Service 之间依然是使用异步消息通讯的方式进行调用，不会直接进行阻塞式的调用。一个 Service 完成业务逻辑处理计算以后，会返回一个处理结果，这个结果以消息的方式异步发送给它的下一个 Service。

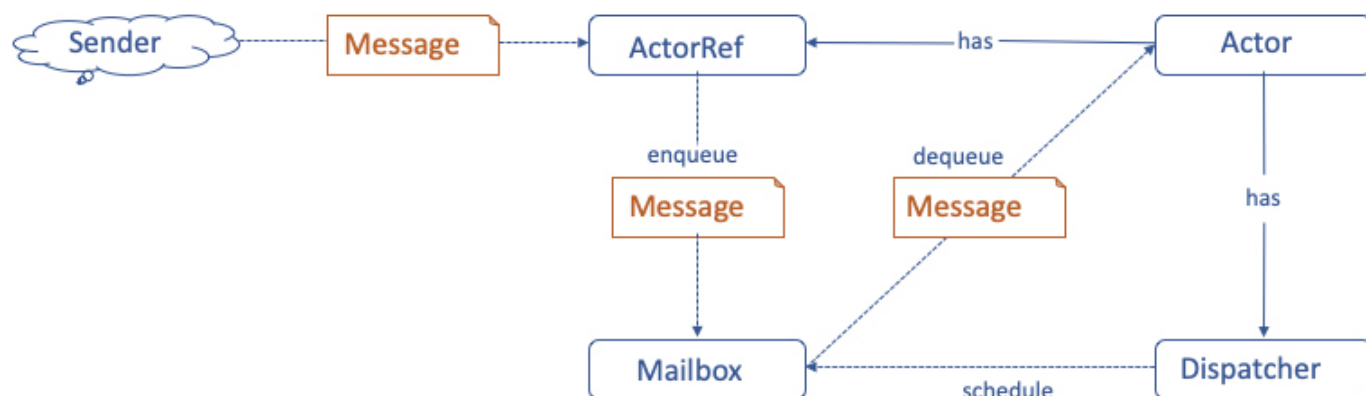
传统编程模型的 Service 之间如果进行调用，如我们在专栏第一篇讨论的那样，被调用的 Service 在返回之前，调用的 Service 方法只能阻塞等待。而 Flower 的 Service 之间使用了 AKKA Actor 进行消息通信，调用者的 Service 发送调用消息后，不需要等待被调用者返回结果，就可以处理自己的下一个消息了。事实上，这些 Service 可以复用同一个线程去处理自己的消息，也就是说，**只需要有限的几个线程就可以完成大量的 Service 处理和消息传输，这些线程不会阻塞等待。**

我们刚才提到，通常 Web 应用主要的线程阻塞，是因为数据库的访问导致的线程阻塞。Flower 支持异步数据库驱动，用户请求数据库的时候，将请求提交给异步数据库驱动，立刻就返回，不会阻塞当前线程，异步数据库访问连接远程的数据库，进行真正的数据库操作，得到结果以后，将结果以异步回调的方式发送给 Flower 的 Service 进行进一步的处理，**这个时候依然不会有线程被阻塞。**

也就是说，使用 Flower 开发的系统，在一个典型的 Web 应用中，几乎没有任何地方会被阻塞，所有的线程都可以被不断地复用，**有限的线程就可以完成大量的并发用户请求，从而大大地提高了系统的吞吐能力和响应时间**，同时，由于线程不会被阻塞，**应用就不会因为并发量太大或者数据库处理缓慢而宕机，从而提高了系统的可用性。**

Flower 框架实现异步无阻塞，一方面是利用了 Web 容器的异步特性，主要是 Servlet3.0 以后提供的 AsyncContext，快速释放容器线程；另一方面是利用了异步的数据库驱动以及异步的网络通信，主要是 HttpAsyncClient 等异步通信组件。而 Flower 框架内，核心的应用代码之间的异步无阻塞调用，则是利用了 Akka 的 Actor 模型实现。

Akka Actor 的异步消息驱动实现如下：



一个 Actor 向另一个 Actor 进行通讯的时候，当前 Actor 就是一个消息的发送者 sender，当它想要向另一个 Actor 进行通讯的时候，就需要获得另一个 Actor 的 ActorRef，也就是一个引用，通过引用进行消息通信。而 ActorRef 收到消息以后，会将这个消息放入到目标 Actor 的 Mailbox 里面去，然后就立即返回了。

也就是说一个 Actor 向另一个 Actor 发送消息的时候，不需要另一个 Actor 去真正地处理这个消息，只需要将消息发送到目标 Actor 的 Mailbox 里面就可以了。自己不会被阻塞，可以继续执行自己的操作，而目标 Actor 检查自己的 Mailbox 中是否有消息，如果有消


息，Actor 则会在从 Mailbox 里面去获取消息，对消息进行异步的处理，而所有的 Actor 会共享线程，这些线程不会有任何的阻塞。

## 反应式编程框架 Flower 的设计方法

但是直接使用 Actor 进行编程有很多不便，Flower 框架对 Actor 进行了封装，开发者只需要编写一些细粒度的 Service，这些 Service 会被包装在 Actor 里面，进行异步通信。

Flower Service 例子如下：


```
1 public class ServiceA implements Service<Message2> {
2     @Override
3     public Object process(Message2 message) {
4         return message.getAge() + 1;
5     }
6 }
```

 复制代码

每个 Service 都需要实现框架的 Service 接口的 process 方法，process 方法的输入参数就是前一个 Service process 方法的返回值，这样只需要将 Service 编排成一个流程，Service 的返回值就会变成 Actor 的一个消息，被发送给下一个 Service，从而实现 Service 的异步通信。

Service 的流程编排有两种方式，一种方式是编程实现，如下：

```
1 getServiceFlow().buildFlow("ServiceA", "ServiceB");
2
```

 复制代码

表示 ServiceA 的返回值将作为消息发送给 ServiceB，成为 ServiceB 的输入值，这样两个 Service 就可以合作完成一些更复杂的业务逻辑。

Flower 还支持可视化的 Service 流程编排，像下面这张图一样编辑流程定义文件，就可以开发一个异步业务处理流程。

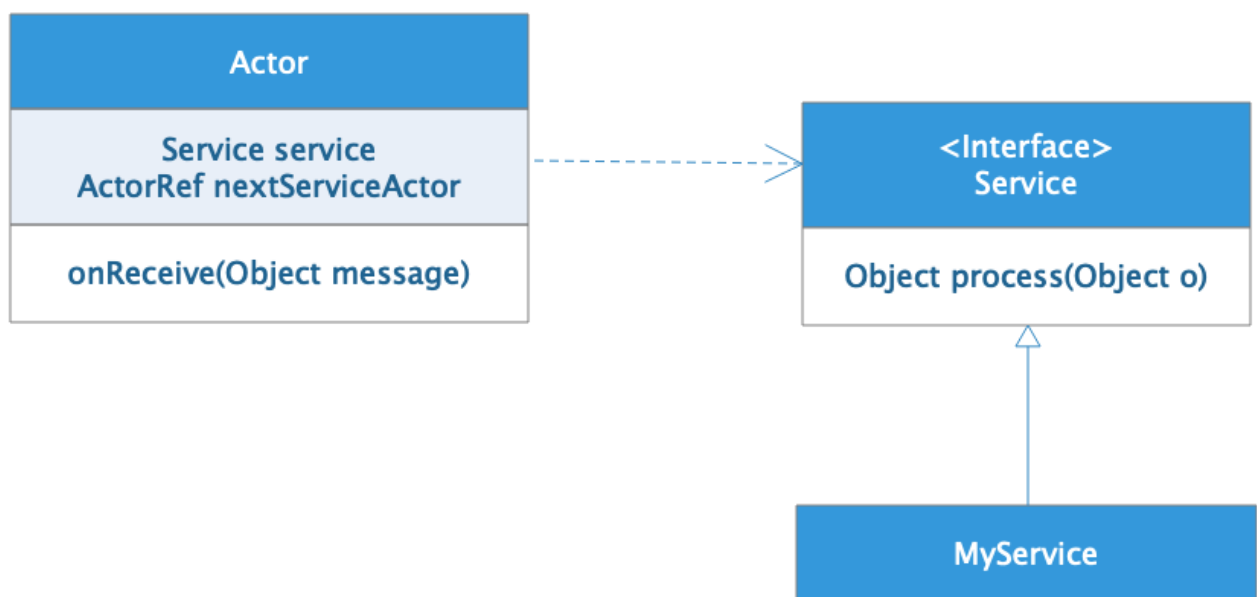
```
// -> service1 -> service2 -> service5 -> service4
//      ^       |           ^       |
//      |       -> service3 -|       |
//      |_____|
```

```
service1 -> service2
service1 -> service3
service2 -> service5
service3 -> service5
service5 -> service4
service4 -> service1
```

那么这个 Flower 框架是如何实现的呢？

Flower 框架的设计也是基于前面专栏讨论过的 [依赖倒置原则](#)。所有应用开发者实现的 Service 类都需要包装在 Actor 里面进行异步调用，但是 Actor 不会依赖开发者实现的 Service 类，开发者也不会依赖 Actor 类，他们共同依赖一个 Service 接口，这个接口是框架提供的，如上面例子所示。

Actor 与 Service 的依赖倒置关系如下图所示：



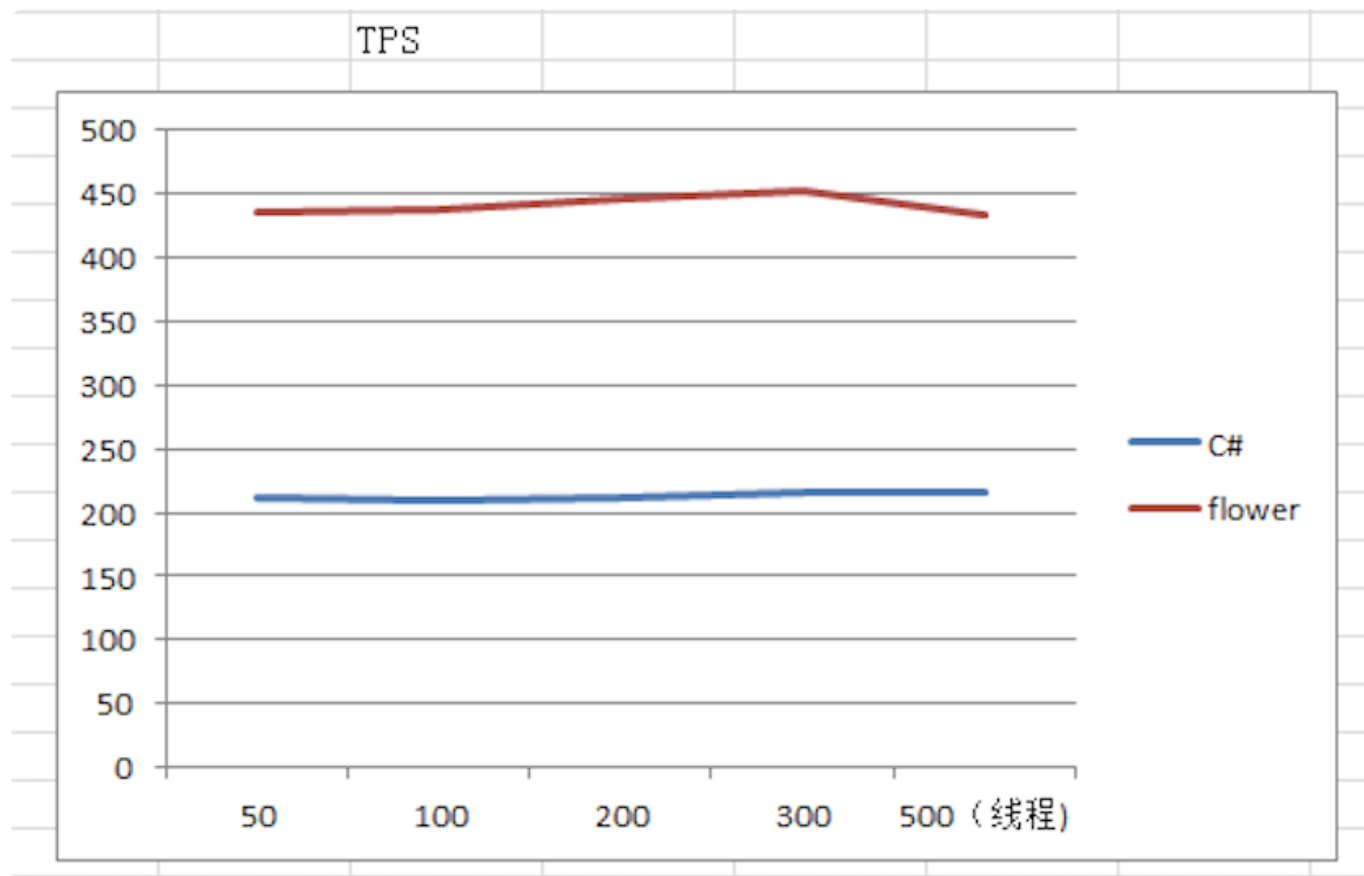


每个 Actor 都依赖一个 Service 接口，而具体的 Service 实现类，比如 MyService，则实现这个 Service 接口。在运行期实例化 Actor 的时候，这个接口被注入具体的 Service 实现类，比如 MyService。在 Flower 中，调用 MyService 对象，其实就是给包装 MyService 对象的 Actor 发消息，Actor 收到消息，执行自己的 onReceive 方法，在这个方法里，Actor 调用 MyService 的 process 方法，并将 onReceive 收到的 Message 对象当做 process 的输入参数传入。

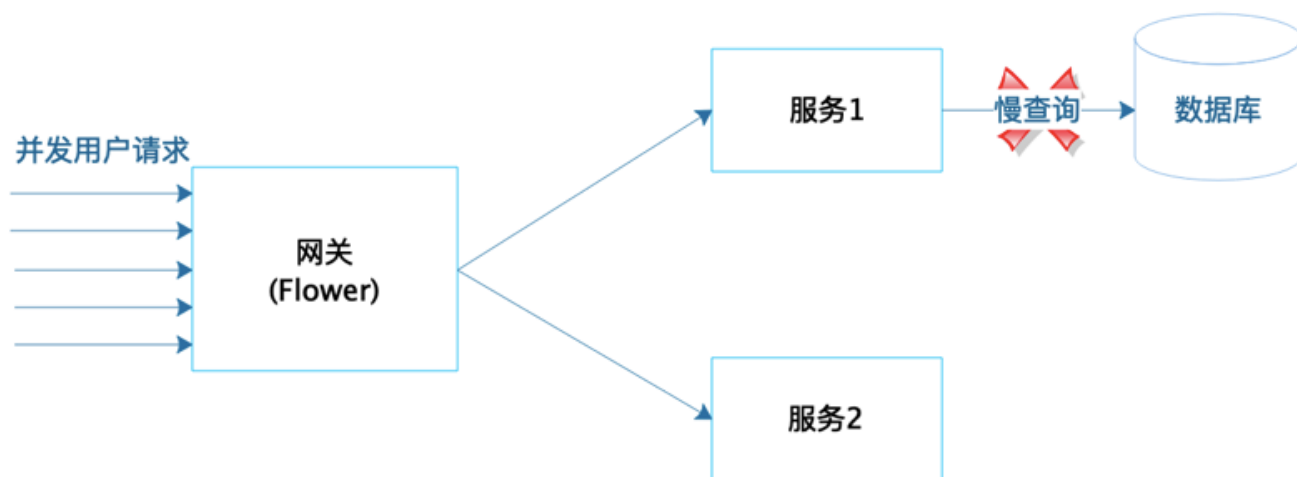
process 处理完成后，返回一个 Object 对象。Actor 会根据编排好的流程，获取 MyService 在流程中的下一个 Service 对应的 Actor，即 nextServiceActor，将 process 返回的 Object 对象当做消息发送给这个 nextServiceActor。这样，Service 之间就根据编排好的流程，异步、无阻塞地调用执行起来了。

## 反应式编程框架 Flower 的落地效果

Flower 框架在部分项目中落地应用，应用效果较为显著，一方面，Flower 可以显著提高系统的性能。这是某个 C# 开发的系统使用 Flower 重构后的 TPS 性能比较，使用 Flower 开发的系统 TPS 差不多是原来 C# 系统的两倍。



另一方面，Flower 对系统可用性也有较大提升，目前常见互联网应用架构如下图：



用户请求通过网关服务器调用微服务完成处理，那么当有某个微服务连接的数据库查询执行较慢时，如图中服务 1，那么按照传统的线程阻塞模型，就会导致服务 1 的线程都被阻塞在这个慢查询的数据库操作上。同样的，网关线程也会阻塞在调用这个延迟比较厉害的服务 1 上。

最终的效果就是，网关所有的线程都被阻塞，即使是不调用服务 1 的用户请求也无法处理，最后整个系统失去响应，应用宕机。使用阻塞式编程，实际的压测效果如下，当服务 1 响应延迟，出错率大幅飙升的时候，通过网关调用正常的服务 2 的出错率也非常高。

请求标签	请求样本数	平均响应时间(ms)	中位值(ms)	90%线(ms)	95%线(ms)	99%线(ms)	最小值(ms)	最大值(ms)	出错率(%)	TPS	接收(KB/s)	发送(KB/s)
服务1	7470	9997	10011	10012	10013	10015	9115	10047	97.4163	25.57	64.93	0.79
服务2	7470	9916	10011	10012	10012	10014	8072	10047	94.6586	25.50	70.7	0.23

使用 Flower 开发的网关，实际压测效果如下，同样服务 1 响应延迟，出错率极高的情况下，通过 Flower 网关调用服务 2 完全不受影响。

请求标签	请求样本数	平均响应时间(ms)	中位值(ms)	90%线(ms)	95%线(ms)	99%线(ms)	最小值(ms)	最大值(ms)	出错率(%)	TPS	接收(KB/s)	发送(KB/s)
服务1	14954	9952	10011	10011	10012	10013	6030	10067	98.034	48.30	123.37	1.13
服务2	14456	12	4	14	52	163	3	231	0	49.33	186.47	8.33

## 小结

事实上，Flower 不仅是一个反应式 Web 编程框架，还是反应式的微服务框架。也就是说，Flower 的 Service 可以远程部署到一个 Service 容器里面，就像我们现在常用的微服务架构一样。Flower 会提供一个独立的 Flower 容器，用于启动一些 Service，这些 Service 在启动了以后，会向注册中心进行注册，而且应用程序可以将这些分布式的 Service 进行流程编排，得到一个分布式非阻塞的微服务系统。整体架构和主流的微服务架



构很像，主要的区别就是 Flower 的服务是异步的，通过流程编排的方式进行服务调用，而不是通过接口依赖的方式进行调用。

你可以点击 [这里](#) 进入 Flower 框架的源代码地址，欢迎你参与 Flower 开发，也欢迎将 Flower 应用到你的系统开发中。你对 Flower 有什么疑问，也欢迎与我交流。

## 思考题

反应式编程虽然能带来性能和可用性方面的提升，但是也带来一些问题，你觉得反应式编程可能存在的问题有哪些？应该如何应对？你是否愿意在工作实践中尝试反应式编程？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流。

点击参加 21 天打卡计划 

## 搞定后端技术基础



扫一扫参与小程序话题



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 设计模式应用：编程框架中的设计模式

下一篇 19 | 组件设计原则：组件的边界在哪里？

## 精选留言 (9)

 写留言



**草原上的奔跑**

2020-01-01

异步的调用一般比较难调试，运行正常的时候性能好、可用性高，但出问题后debug比较困难，不知道李老师和你们是如何处理的

展开 ▾

作者回复: 异步难调试主要是因为异步的代码执行在不同的线程上，导致没有统一的调用栈，异常的时候不知道哪里出错的。Flower通过流程编排的方式管理异步方法代码，出错的时候可以通过流程定位异常，对调试定位有帮助。



4



**Paul Shan**

2020-01-08

高并发系统，传统解决方案的弊端在于调度的单元是线程，当IO操作时，线程做无谓地等待，相应的CPU资源白白浪费，线程数目多到一定程度，系统就会被压垮。

反应式编程的第一个目标即时响应，就是异步非阻塞的，遇到IO操作就返回，通过回调函数来取结果。...

展开 ▾

作者回复: 👍



1



**QQ怪**

2020-01-07

学习到了

展开 ▾



1



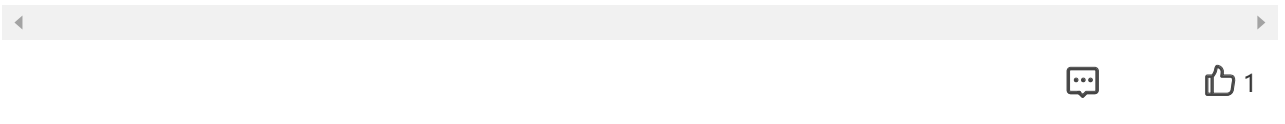
**老男孩**

2020-01-01

李老师和专栏的朋友们新年快乐！哈哈！又一年了。虽然我还在现实的苟且中徘徊，但依然算是不忘初心吧。用一句现在很流行的话就是，愿大家只争朝夕，不负韶华。之前研究了一下spring的webflux发现目前异步编程在数据库驱动这块对关系型数据库的支持不太好，比如myspl。还有诺诺的问一下，mailbox是不是一个内存队列？不知道flower能不能支持用第三方的消息队列来替换mailbox的工作？还有actor里边是不是用到模板设计模...

展开 ▾

作者回复: 新年快乐。异步数据库驱动还是有一些的, 但是因为业界用的不多, 所以都不怎么维护。反应式编程的大气候还是没起来。



1

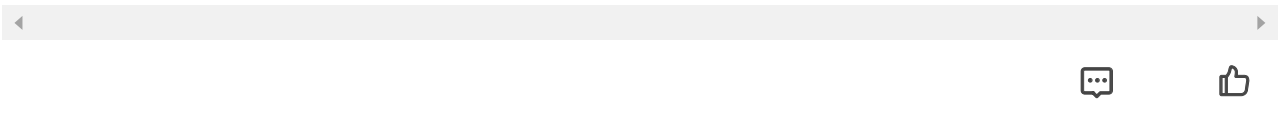


饭饭

2020-01-10

这样流程编排的为什么不用akka stream?有什么考虑嘛?

作者回复: 编程模型不同, 我们认为Flower编程模型更加友好, 事实如何, 我们将来再看。

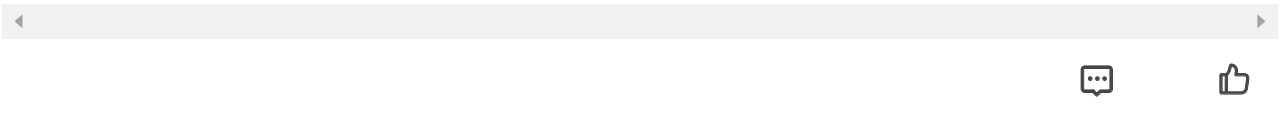


你的美

2020-01-03

老师这篇文章我一时还掌握不了, 课程学完后, 以后多久就不能打开看了呢?

作者回复: 购买以后可以一直看的



鹏酱

2020-01-02

最终一致性和结果查询, 处理方法是, 补偿机制和定时查询



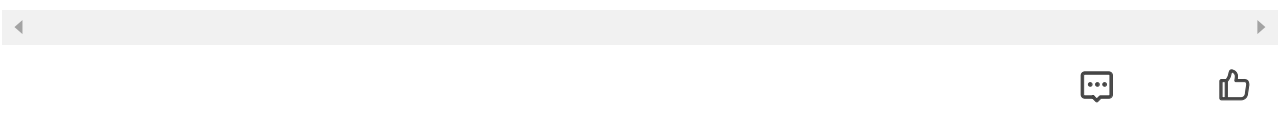
七七的首席铲屎官

2020-01-01

老师您好 对响应式我有一点困惑 响应式通过异步消息来减少同步造成的时间等待 但是在实际实践的时候对于需要同步的需求要怎么实现呢 比如在页面对某个字段进行更新时响应式接口无法立即得到最终的结果 对于更新失败的情况怎么在前端展示呢?

展开

作者回复: 反应式实践其实更多落地就是在前端, 发送请求后就不管了, 然后等到异步响应到达后异步更新页面。



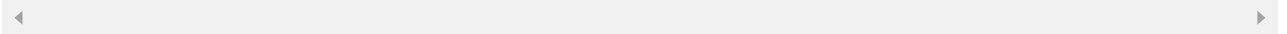


a、

2020-01-01

老师，我目前使用的play框架也是基于akka做异步调用的，你觉得play框架和flower框架有哪些区别？

作者回复: 背后的运行原理一致，但是编程模型有很大区别，Flower是基于流程进行编程的，更符合开发者的思维习惯。



💬 1

