Java Chassis通信处理详解

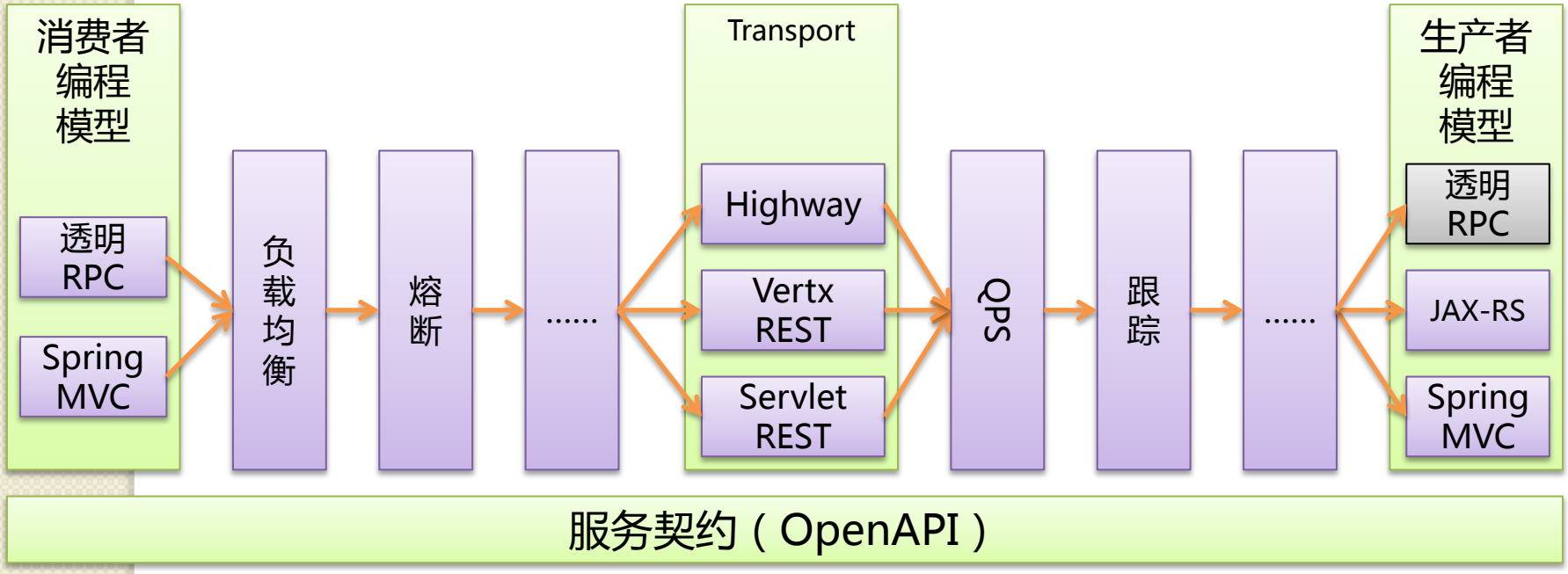
通信优化实践

议题

- 问题与挑战
- 整体线程模型
- Consumer
- Producer

问题与挑战

1. RPC还是REST，传输方式决定编程模型
2. 不同开发人员熟悉不同的编程模型
3. RPC、REST的治理如何进行
4. REST性能低
5. Reactive还是同步



ServiceComb的同步

仅仅是指编程模型上的同步，跟网络通信无关

因为所有场景下网络操作都是异步的

- Producer

```
@GetMapping(path="/sayHello")
public String sayHello() {
  return "hello world";
}
```

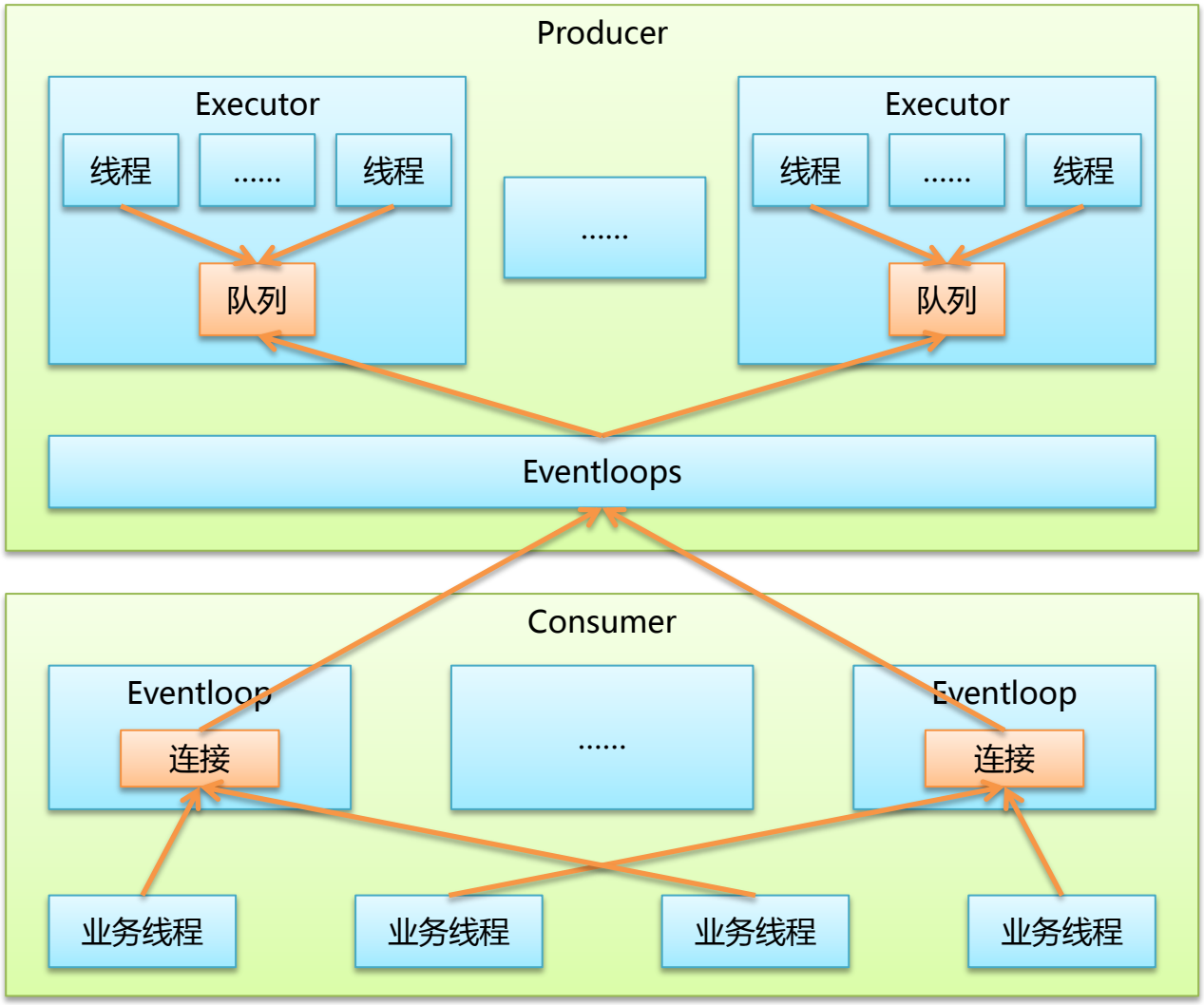
- Consumer

```
interface Hello{
  String sayHello();
}

String result = hello.sayHello();

String result = restTemplate.getForObject("cse://{name}/sayHello", String.class);
```

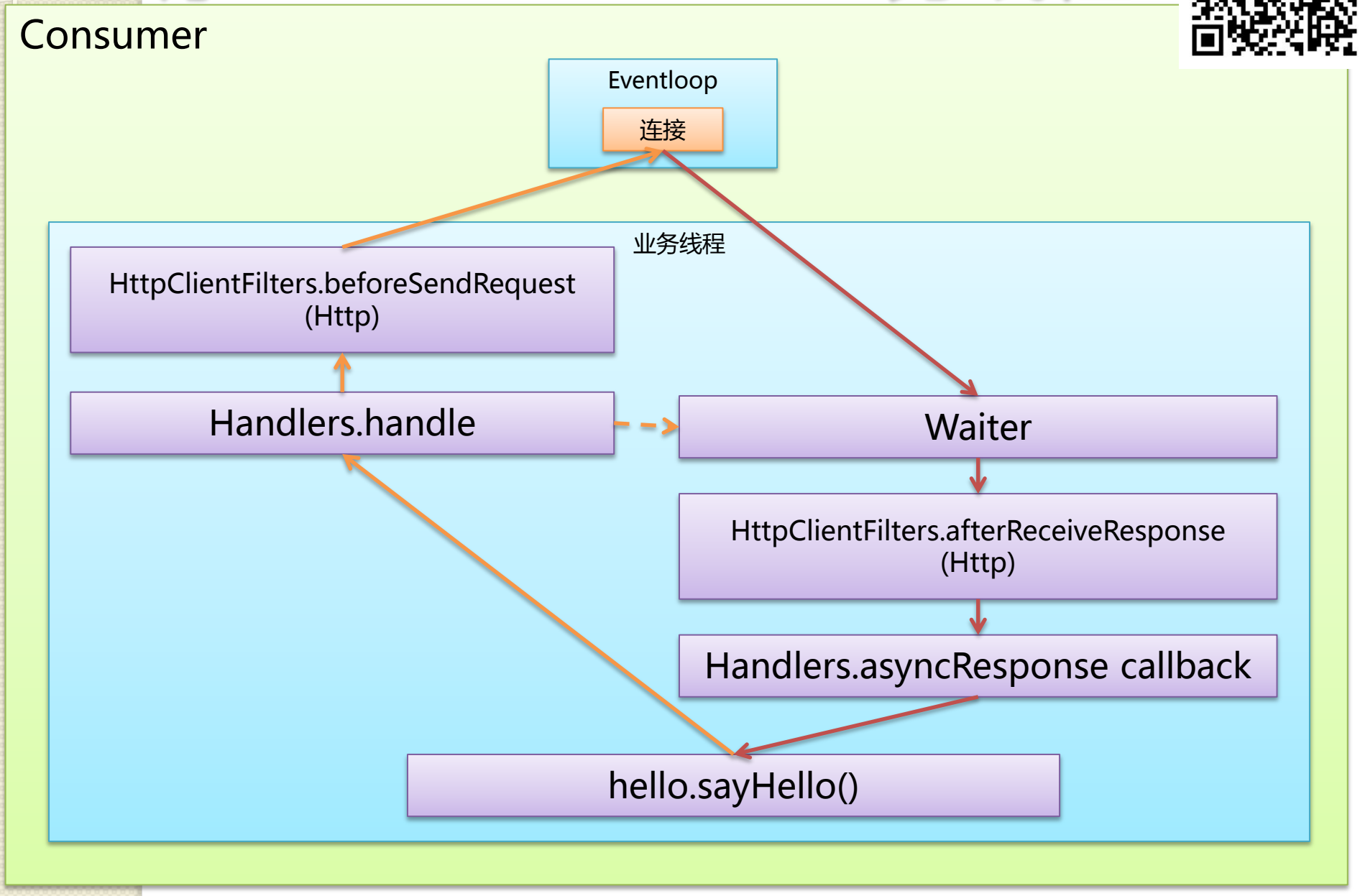
整体线程模型


Producer
Executor
线程
.....
线程
队列
.....
Executor
线程
.....
线程
队列
Eventloops
Consumer
Eventloop
连接
.....
Eventloop
连接
业务线程
业务线程
业务线程
业务线程

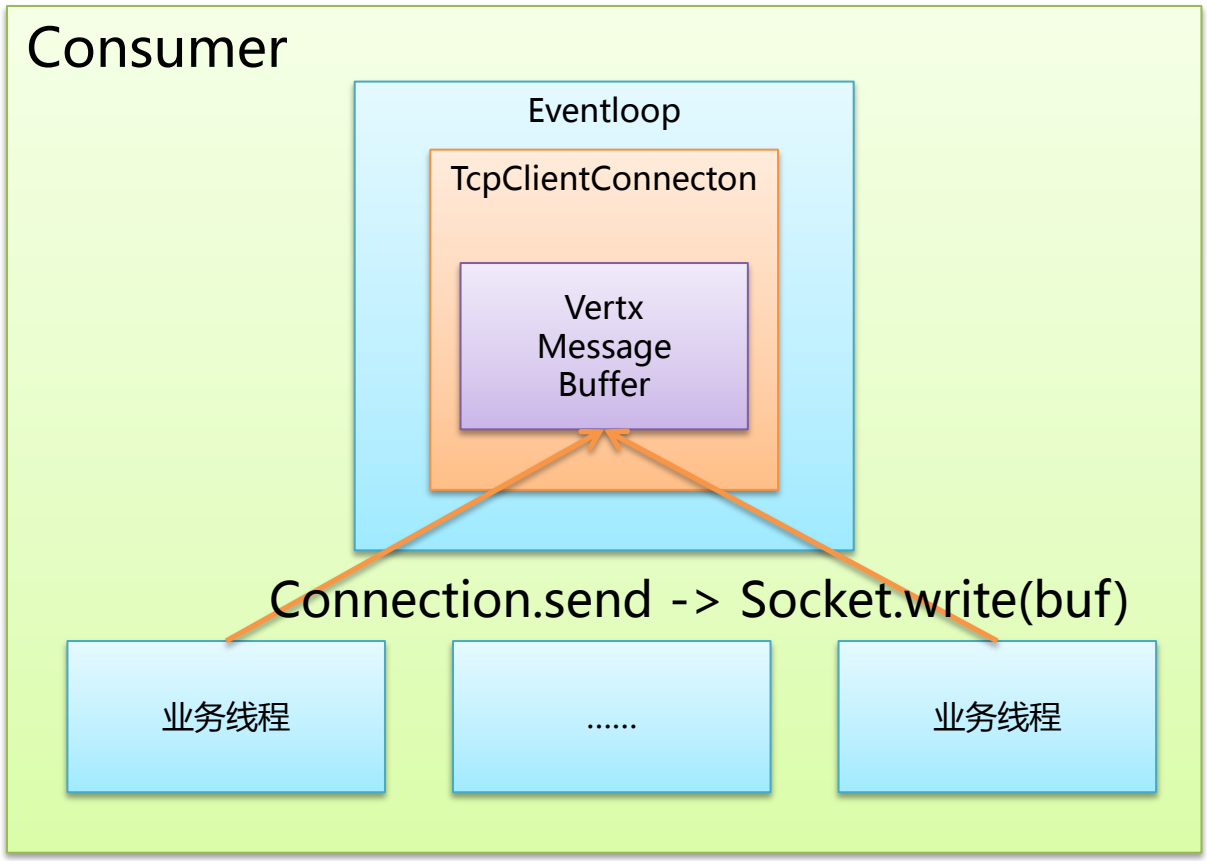
Consumer

消费端，主要需要处理的问题是如何更高效地将请求推送到对端去，然后拿到应答信息。

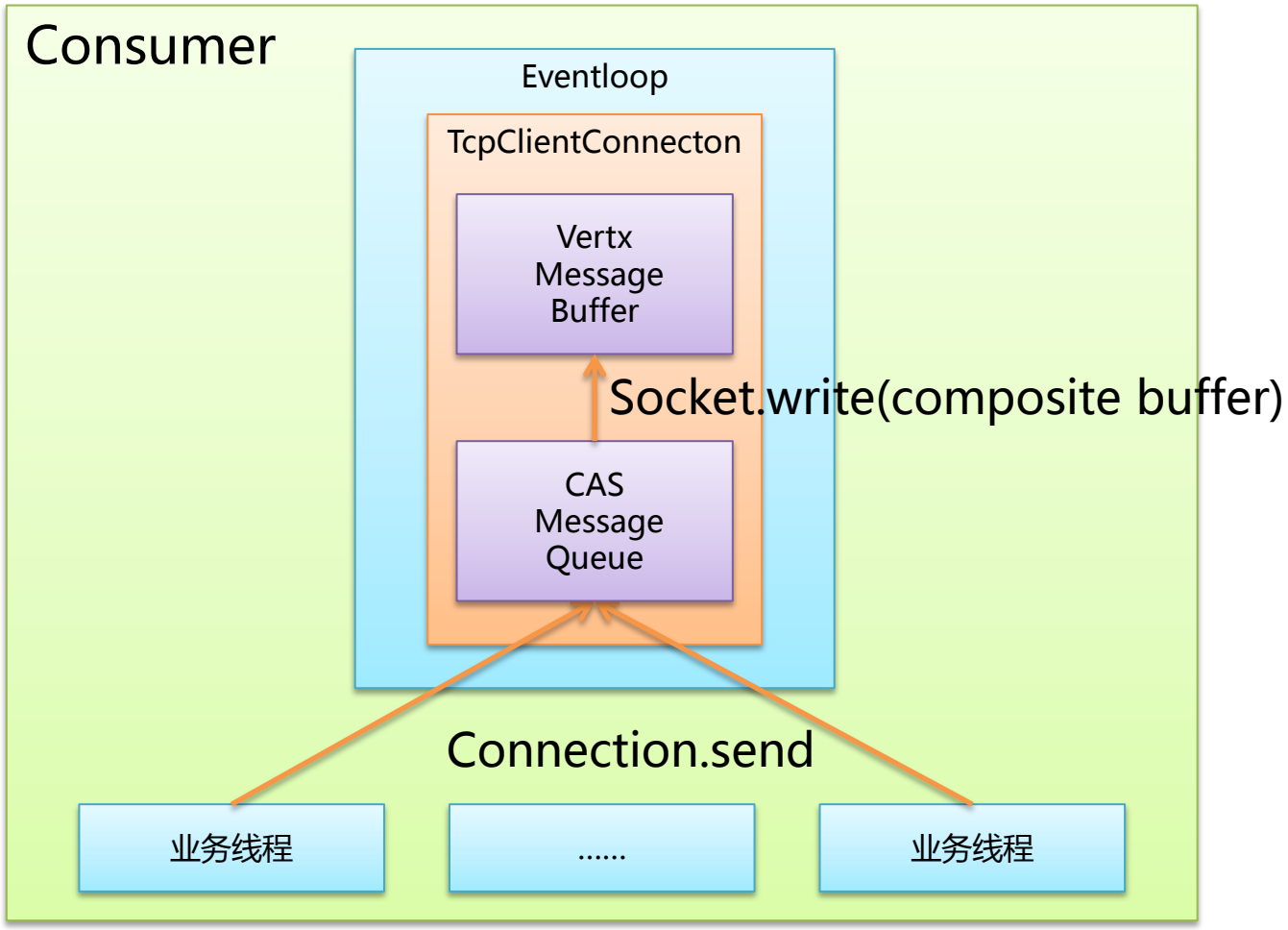
透明RPC Consumer业务线程



Highway client-原始单连接模型

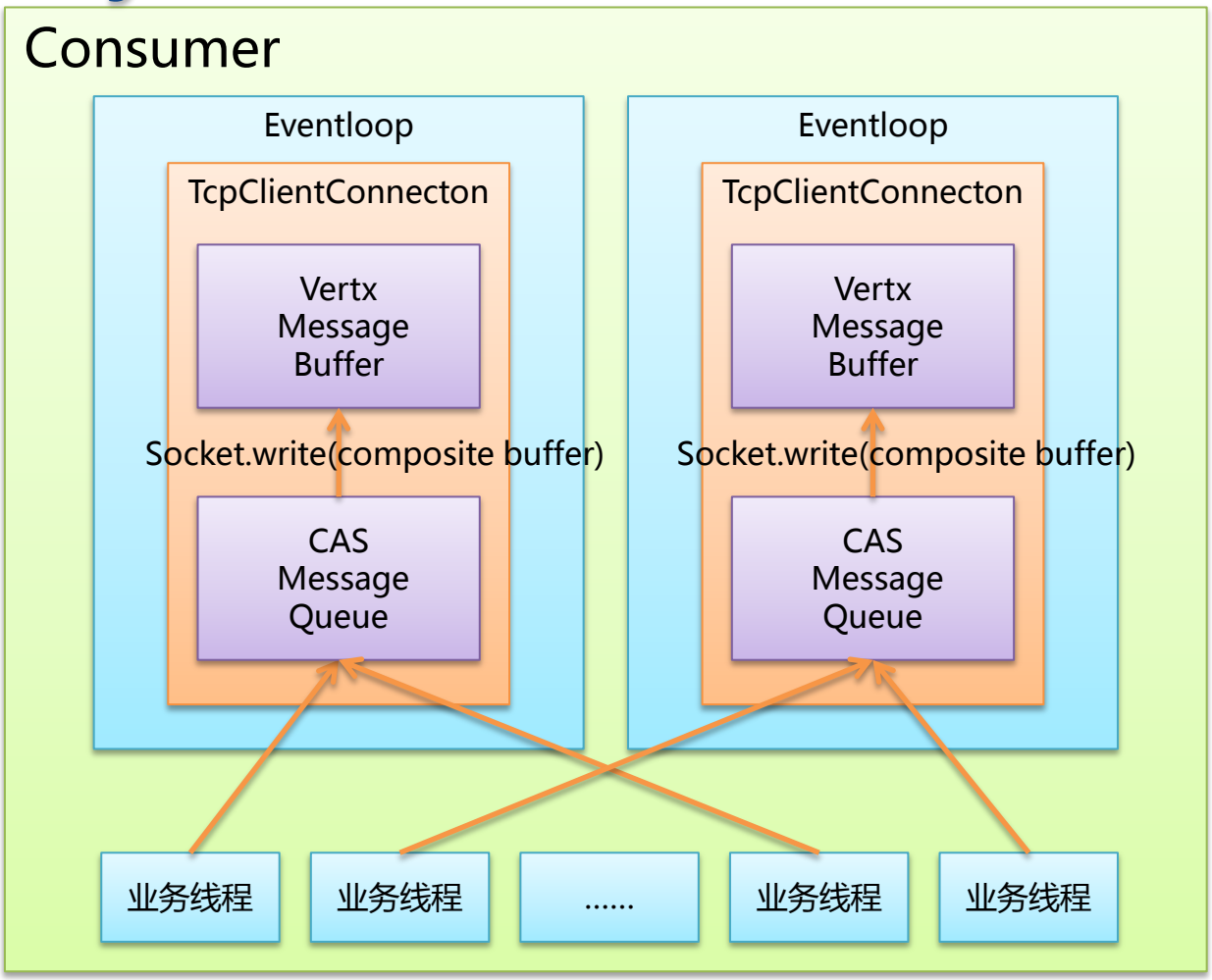


Highway client-优化单连接模型



	TPS	Latency(ms)	CPU	
			Consumer	Producer
优化前	81986	1.22	290%	290%
优化后	145369	0.688	270%	270%

Highway client-多连接模型



	TPS	Latency(ms)	CPU	
			Consumer	Producer
原始单连接*10	543442	0.919	2305%	1766%
CAS单连接*10	939117	0.532	1960%	1758%

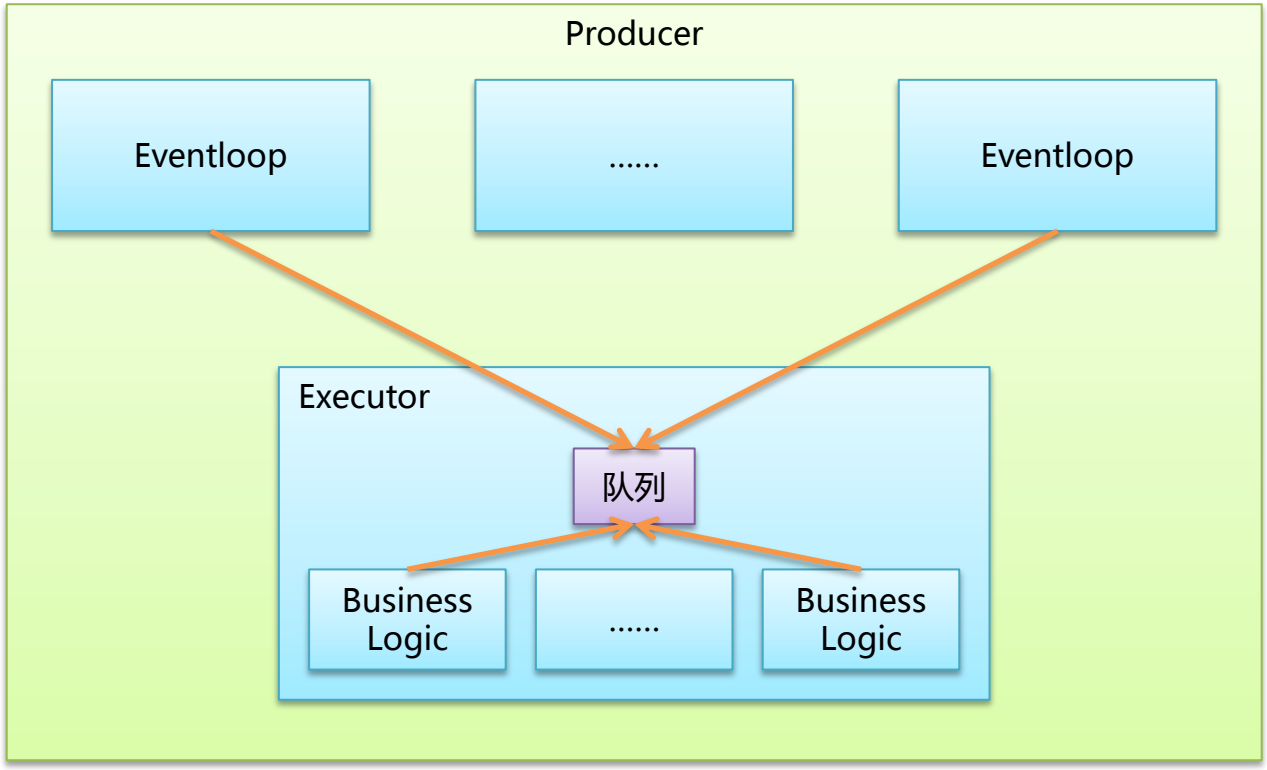
Producer

- 不同于消费者，生产者主要的工作就是等待消费者的请求，在处理之后，返回应答
  在这一端我们更加关注：“如何高效地接收和处理数据”这件事情

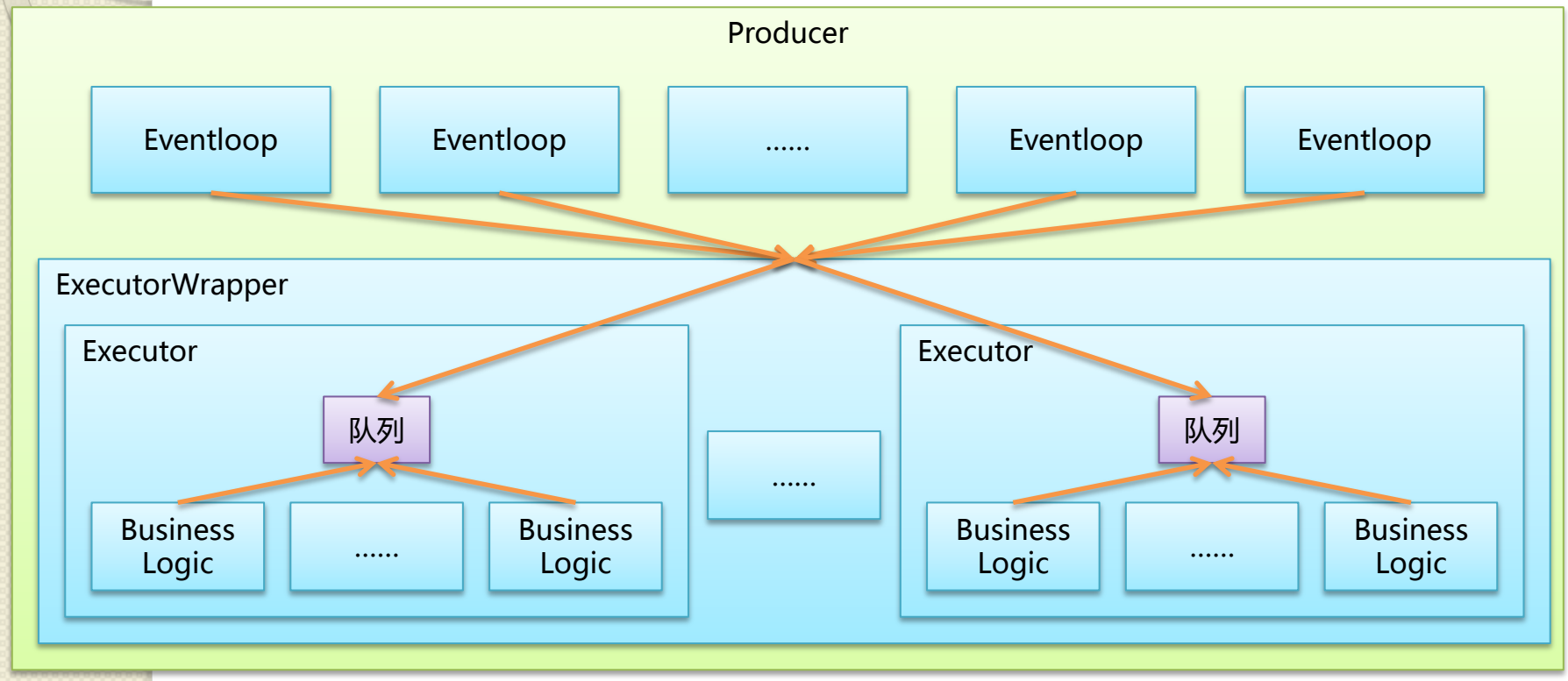
- 同步模式下，业务逻辑与IO逻辑分离，且根据“隔离仓”原则，为了保证整个系统更加稳定和高效地运行，业务逻辑本身也需要在不同的隔离的区域内进行。而这些区域，就是线程池。

所以构建生产者，就需要对线程池进行精细的管理。
下面是针对线程池的各种管理方式。

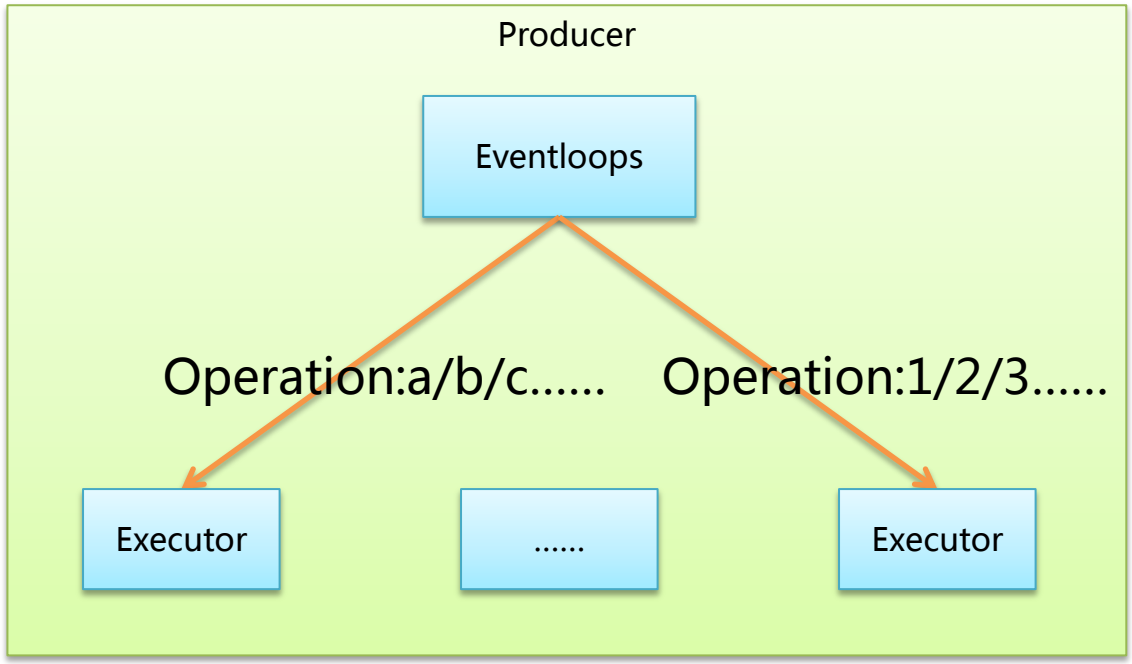
Producer-单线程池



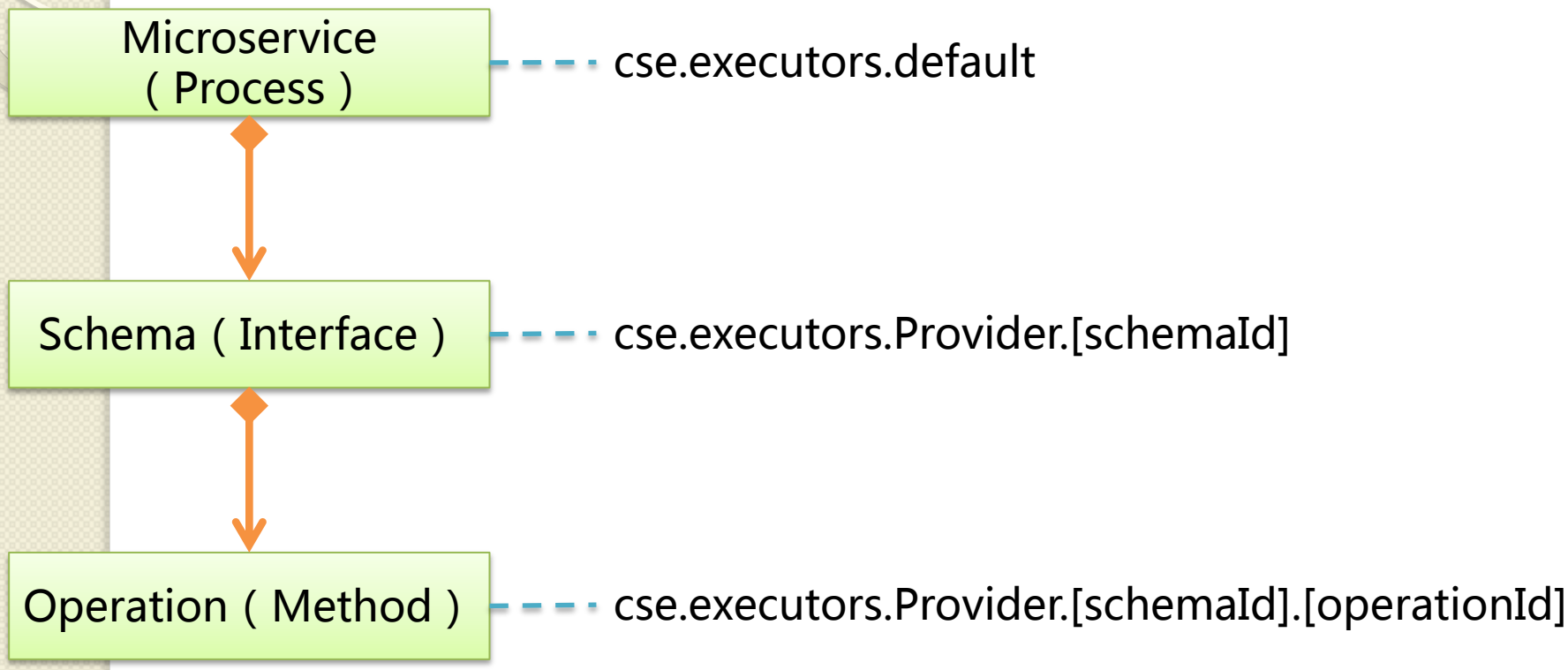
Producer-多线程池



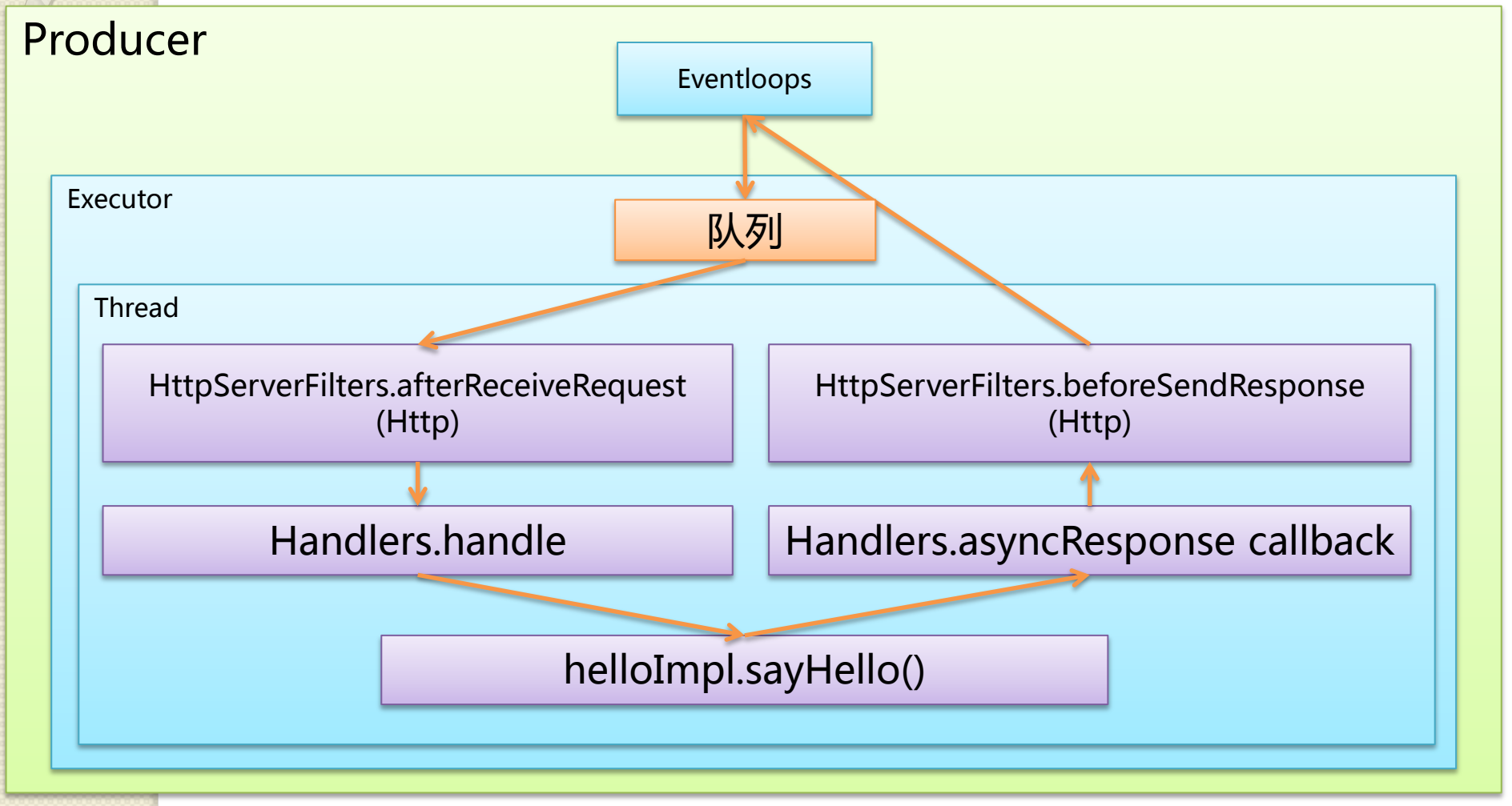
Producer-隔离仓



Producer-灵活的线程池策略



Producer业务线程



微服务引擎商业版：http://www.huaweicloud.com/product/cse.html
ServiceComb Github：https://github.com/apache?q=servicecombServiceComb
官网: http://servicecomb.incubator.apache.org/cn/

ServiceComb

华为 PaaS 微服务开源框架　　让云原生开发更简单

交流论坛

微信入群