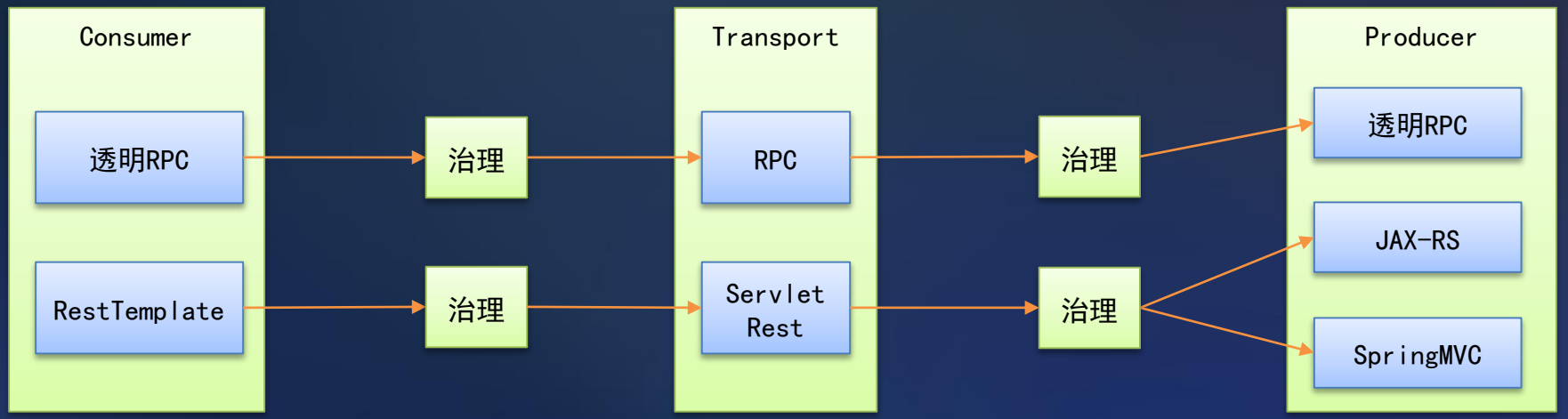
ServiceComb java SDK详解

AGENDA

- 设计演进及架构
- 服务发现
- 微服务调用
- Edge Service
- Metrics
- 性能调优

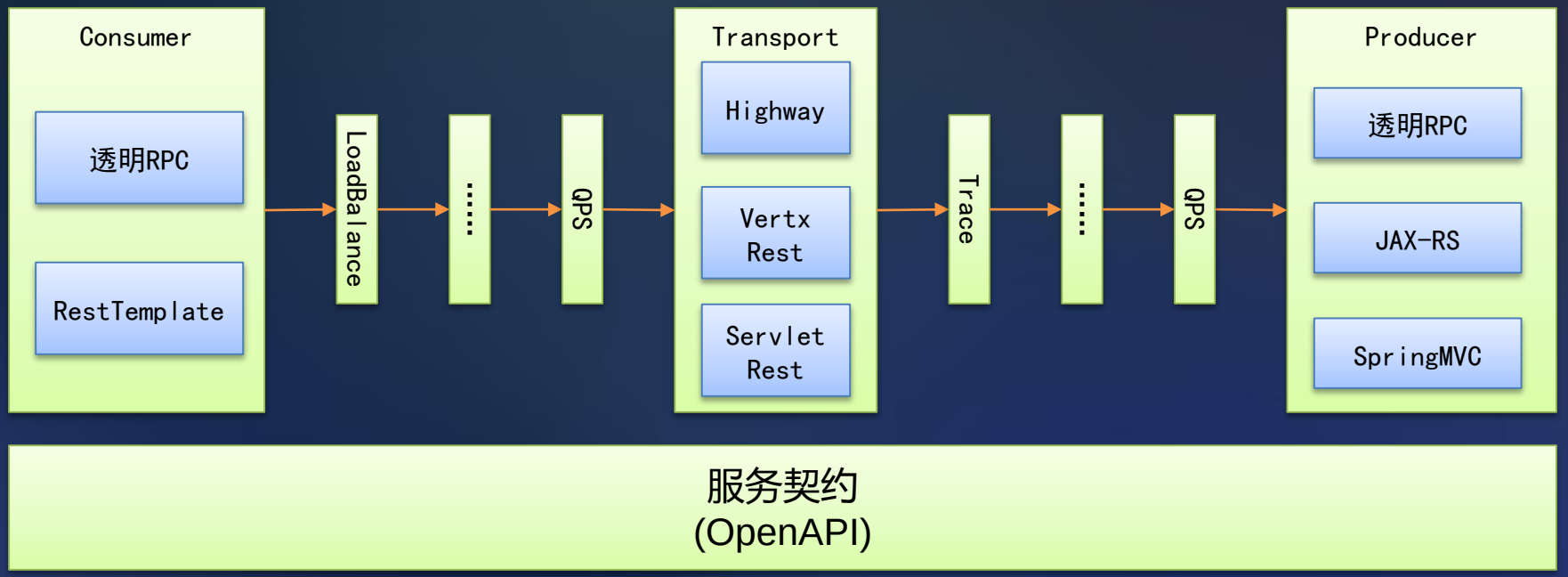
website: http://servicecomb.incubator.apache.org/
gitter: https://gitter.im/ServiceCombUsers/Lobby

设计演进-初始



- 传输绑架开发模式
- 传输可能对业务代码产生严重的侵入
- 治理逻辑与业务代码或传输耦合，需要针对各种场景独立开发

设计演进



以契约为核心，贯穿始终

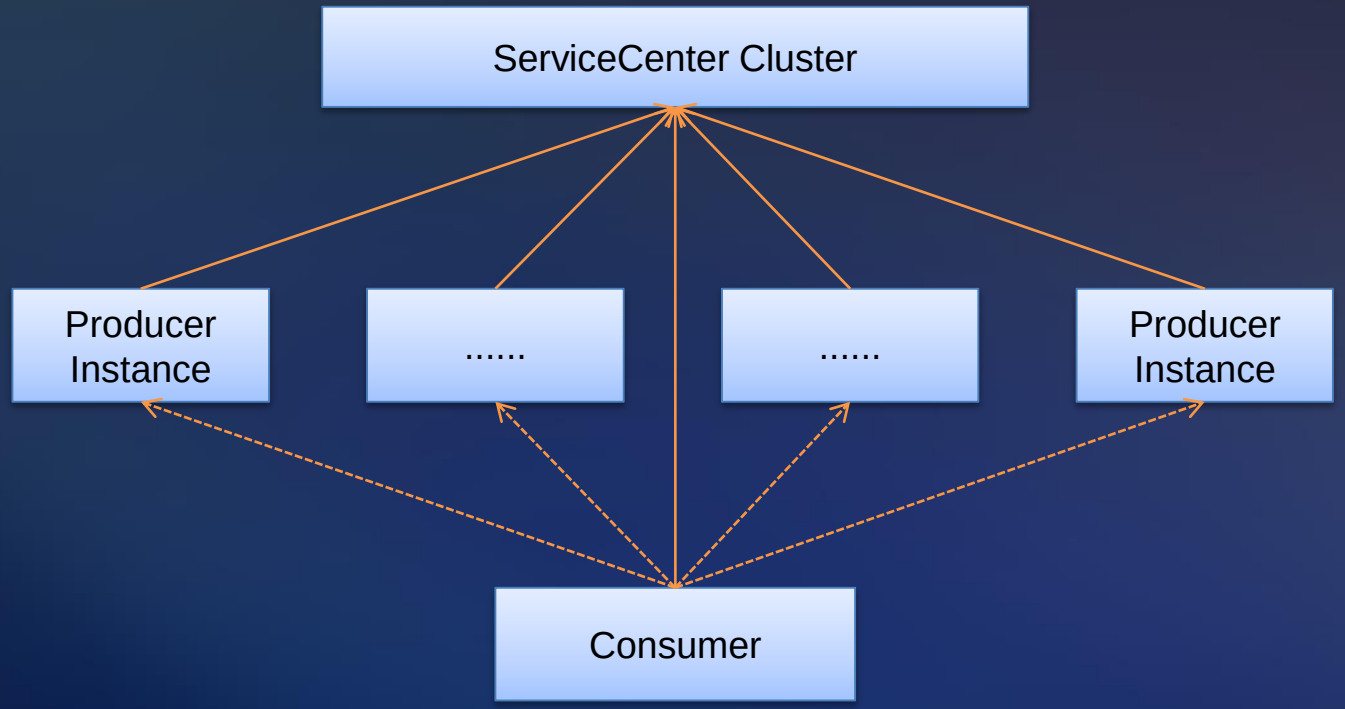
- 开发模式(业务代码)、治理、传输三方解耦，互不感知
- 以高性能的Reactive为基础，同时兼容传统的同步开发模式，且支持传统的Servlet传输

架构


SpringBoot Starters
Implementation
RestTemplate
Trace
Highway
SpringMVC
QPS
RESTful Vertx
Jaxrs
Metrics-publishers
透明RPC
LB
RESTful Servlet
透明RPC
Metrics-core
ServiceComb Core
Definition
Consumer
Handler
Transport
Producer
……
Swagger
Generator
透明RPC
Jaxrs
SpringMVC
Invocation
透明RPC
Jaxrs
SpringMVC
Registry
Common
javassist
protobuf
rest
….
Foundation
common
config
vertx
ssl
metrics
….

服务发现-实例管理

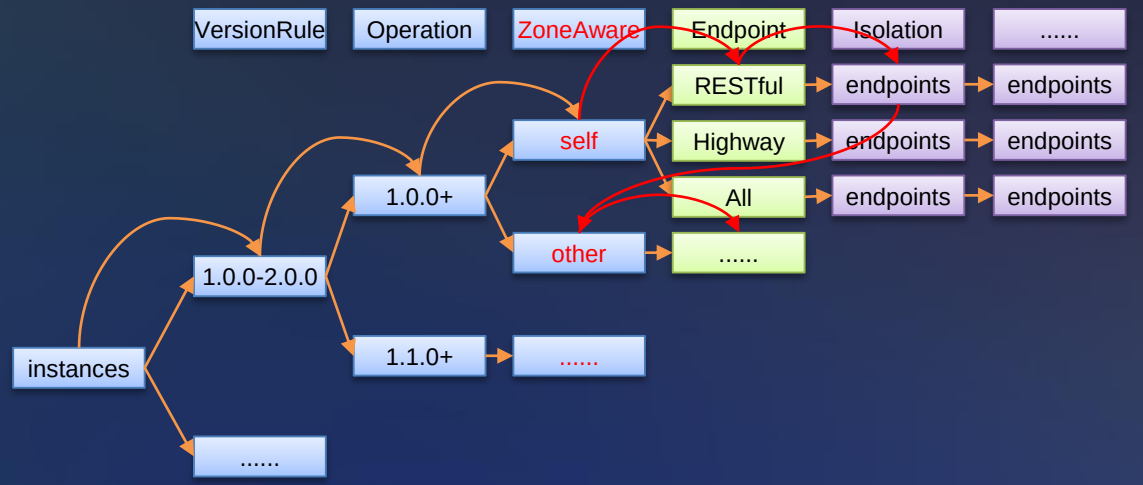
- 每个微服务实例注册到服务中心，并与服务中心保持心跳
- 消费者周期性从服务中心更新目标微服务实例集合
- 微服务实例与服务中心之间的如果成功建立WebSocket连接，则服务中心会将目标实例变更事件尽快推送给相关的消费者



服务发现-LoadBalance

1. 将从服务中心取得的实例集合，进行过滤，得到Endpoint集合
   - 根据兼容规则，将实例进行分组
     只有edge service场景下，对于同一个目标微服务，存在多个版本分组
   - 兼容分组内，新版本功能集合大于旧版本，自动根据url选择合适的版本组
   - AZ亲和性
   - 根据传输通道分组
   - 根据实际调用统计，隔离暂时不可用的调用目标
   - 开发人员通过SPI扩展过滤功能

2. 以过滤得到的Endpoint集合作为输入，根据配置的策略进行LoadBalance：
   - 轮询、响应时间权重、随机
   - 基于Netflix ribbon，开发人员可以根据需要自定义IRule的实现



微服务调用-consumer

透明RPC：

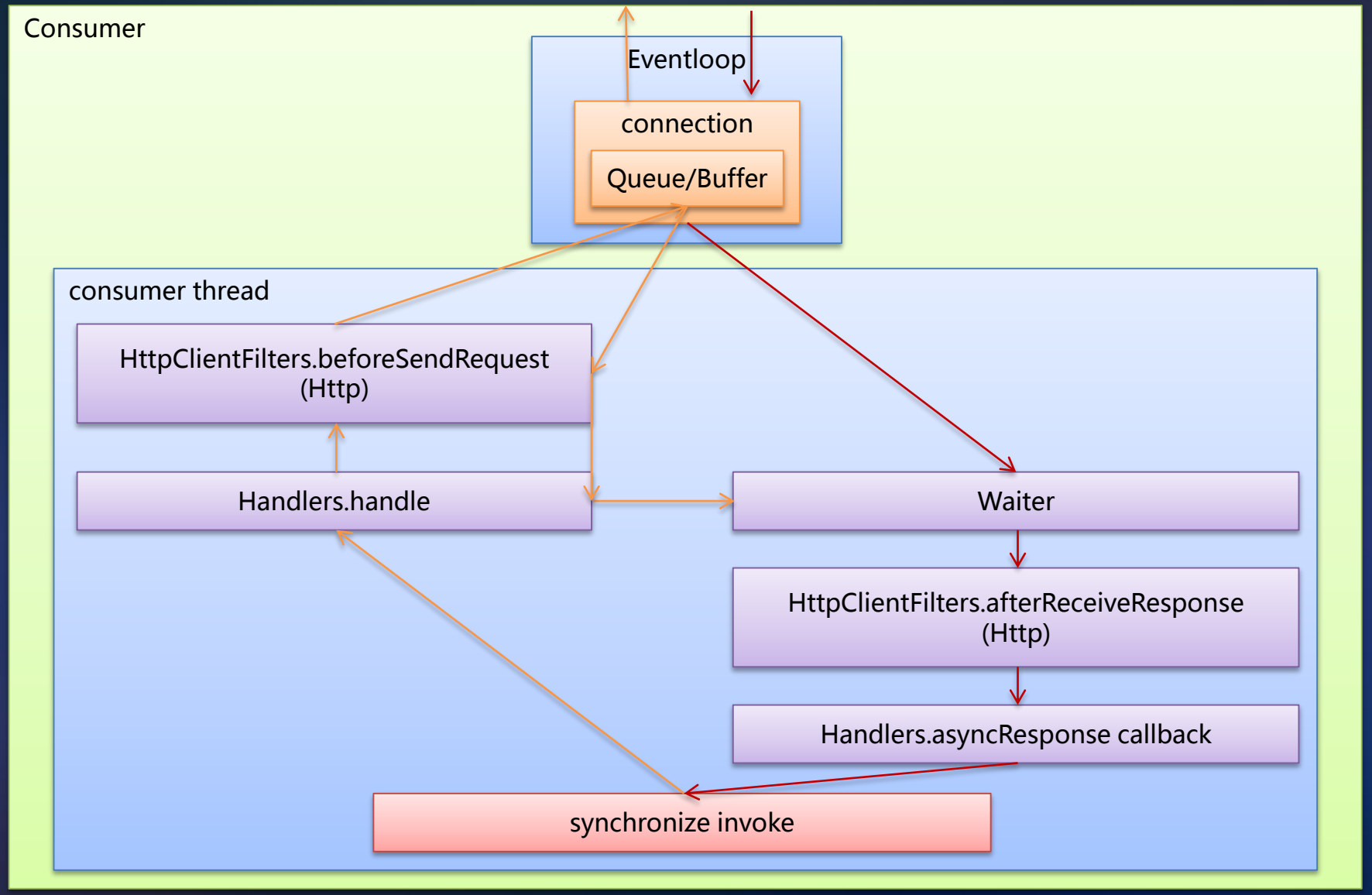
同步、异步声明，可以分开进行，也可以在同一个接口中声明

```
interface Schema {
  String hello(String name);
}

interface Schema {
  CompletableFuture<String> hello(String name);
}

interface Schema {
  String hello(String name);

  @ApiOperation(nickname = "hello", value = "reactive method for hello")
  CompletableFuture<String> helloReactive(String name);
}
```

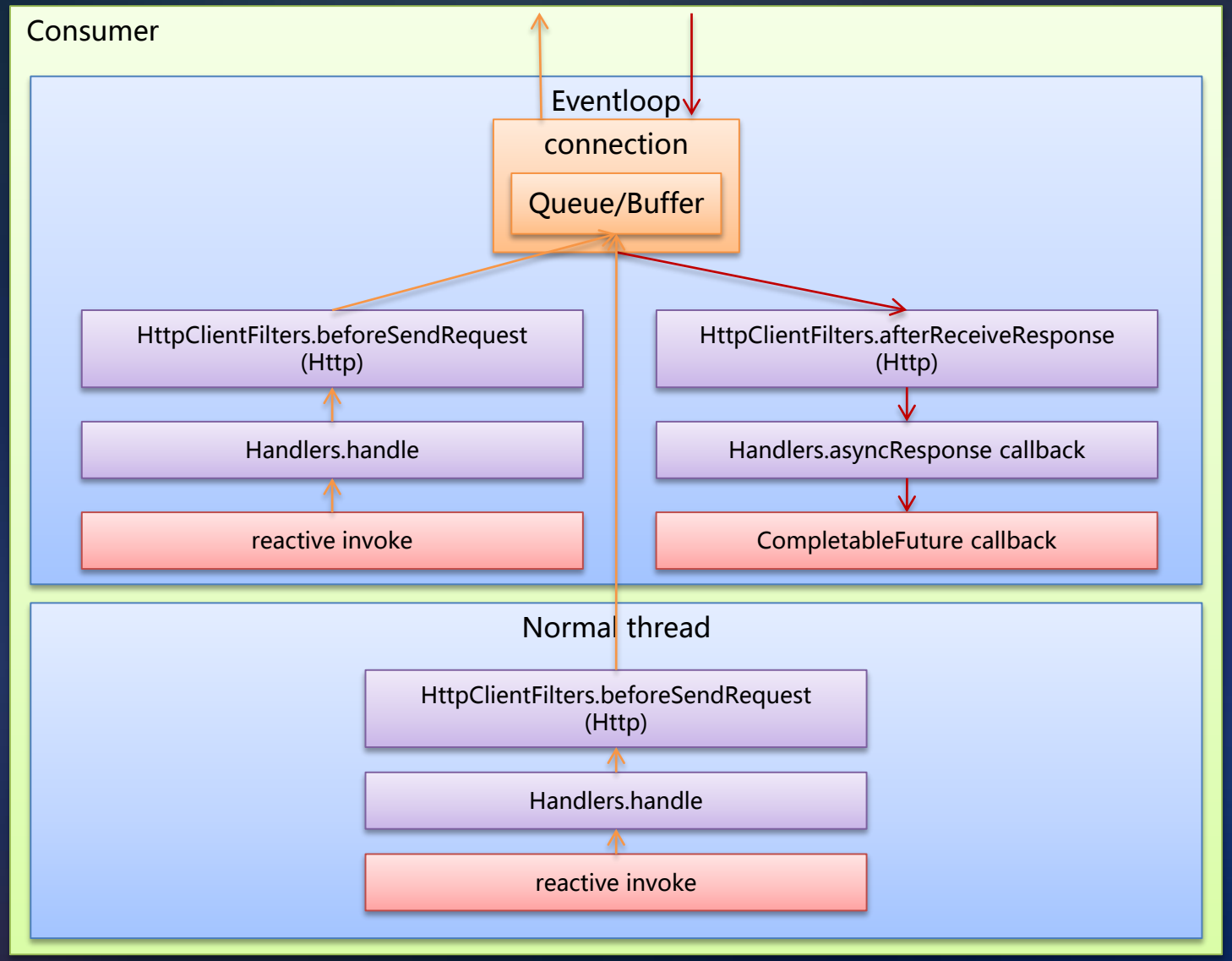
RestTemplate

AsyncRestTemplate

微服务调用-同步consumer



微服务调用-reactive consumer

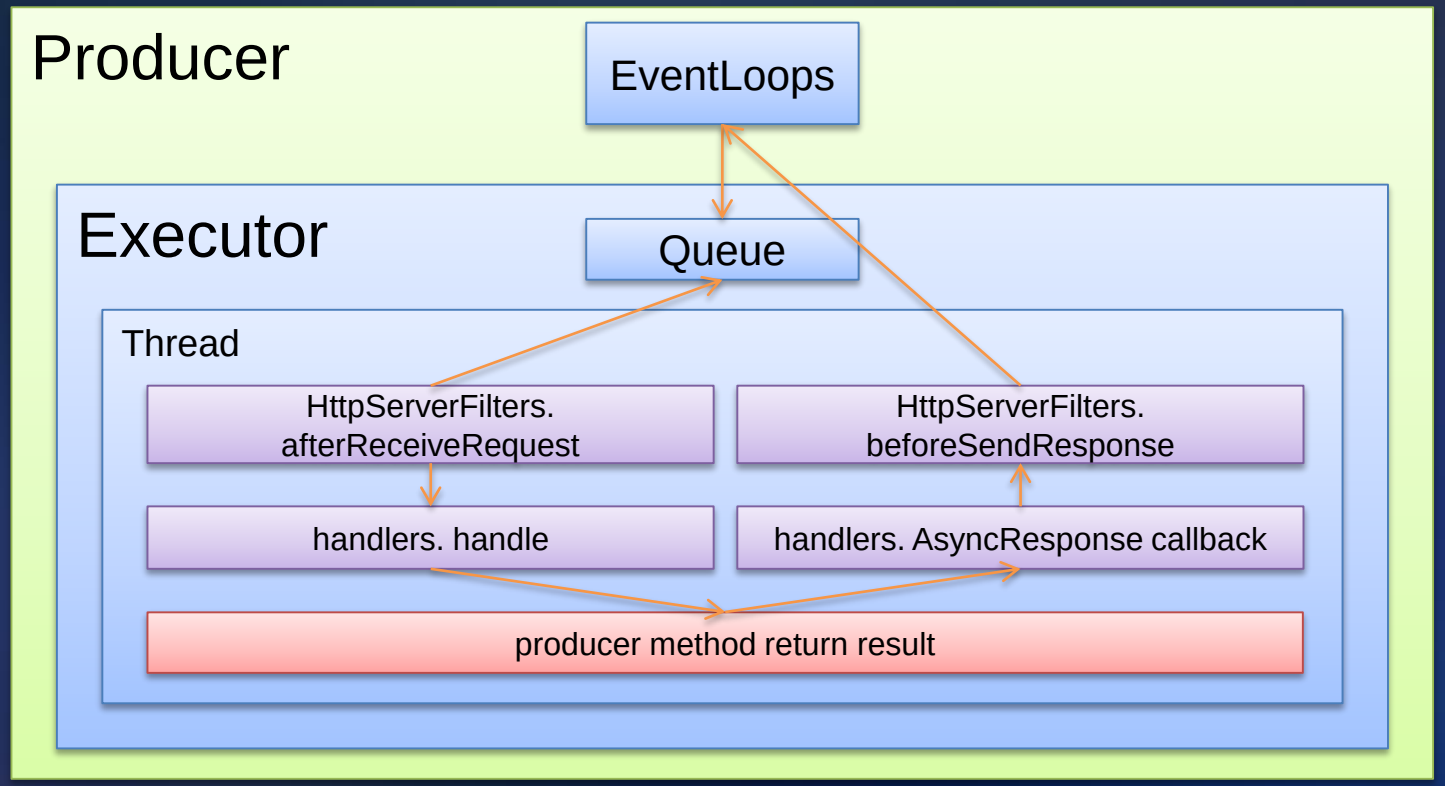


微服务调用-producer

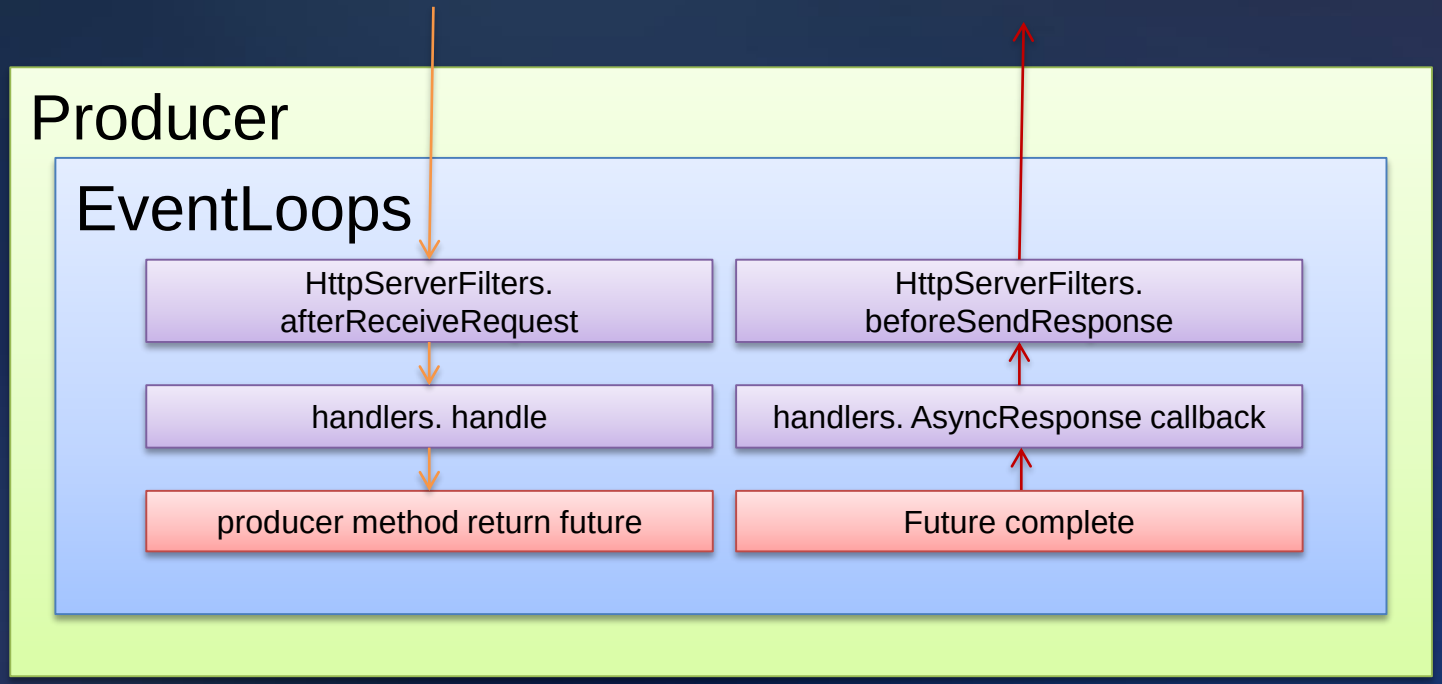
透明RPC、JAX-RS、SpringMVC统一规则：

```
class Schema {
  // 传统同步调用，默认在线程池中执行
  public String hello1(String name) {
    return “hello “ + name;
  }

  // 默认为reactive模式，不通过线程池执行
  public CompletableFuture<String> hello2(String name) {
    return CompletableFuture.completedFuture(“hello “ + name);
  }
}
```

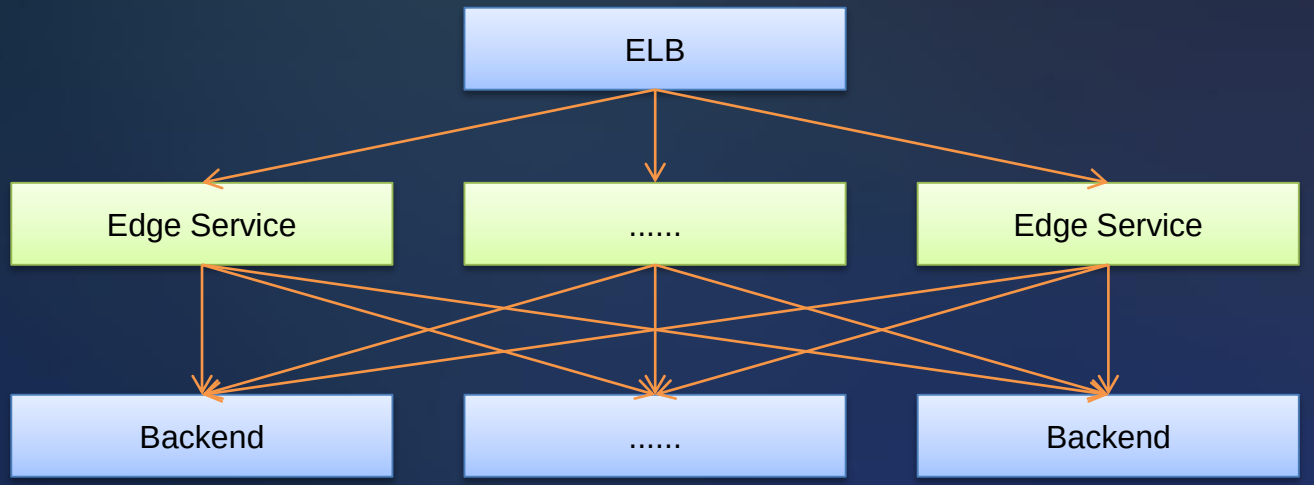
微服务调用-同步producer



微服务调用-reactive producer



Edge Service



- 作为整个微服务系统对外的接口，向最终用户提供服务，接入RESTful请求，转发给内部微服务
- 基于ServiceComb标准consumer流程转发请求到内部微服务，所有consumer治理能力，不必独立开发，直接重用

Edge Service-自动路由1

Edge URL规划为：/{pefix}/{microserviceName}/{v1/v2/.....}/path

- 第一个segment为固定前缀，表示这是一个微服务调用，可用于让前置LB识别这是一个需要转发到edge的请求
  这里假设前缀为rest
- 第二个segment为微服务名
- 第三个segment表示兼容规则，v1表示调用1.0.0-2.0.0的微服务,v2表示2.0.0-3.0.0的微服务

如果内部微服务URL规则为：

/{microserviceName}/{v1/v2/.....}/path

则可以配置为

```
servicecomb:
  http:
    dispatcher:
      edge:
        default:
          enabled: true
          prefix: rest
          withVersion: true
          prefixSegmentCount: 1
```

如果内部微服务URL规则为：

/{v1/v2/.....}/path

则可以配置为

```
servicecomb:
  http:
    dispatcher:
      edge:
        default:
          enabled: true
          prefix: rest
          withVersion: true
          prefixSegmentCount: 2
```

如果内部微服务URL规则为：

/path

则可以配置为

```
servicecomb:
  http:
    dispatcher:
      edge:
        default:
          enabled: true
          prefix: rest
          withVersion: true
          prefixSegmentCount: 3
```

Edge Service-自动路由2

有的场景下，业务是从传统应用移植过来的，有的URL已经固定，并且不满足自动路由规则，此时必须考虑兼容问题
对于这些不规则的URL，可以追加通过枚举的方式来指定路由规则

```
servicecomb:
  http:
    dispatcher:
      edge:
        url:
          enabled: true
          mappings:
            businessAV1:
              path: "/a/b/.*"
              microserviceName: businessA
              versionRule: 1.0.0-2.0.0
            businessBV2:
              path: "/c/d/.*"
              microserviceName: businessB
              versionRule: 2.0.0-3.0.0
```

这些路由配置，可以从配置中心动态下发，避免要发布新微服务时，需要重启Edge

Edge Service-自定义路由

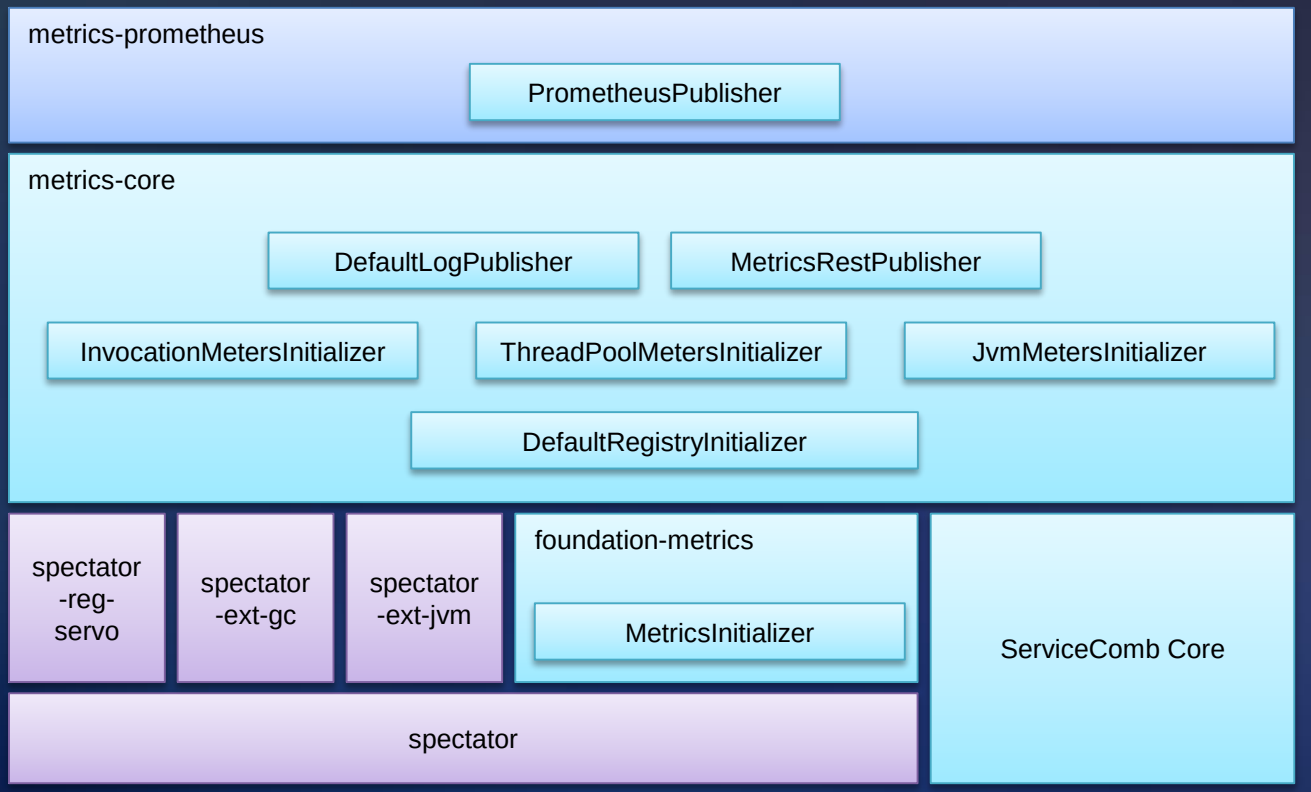
如果内置的路由规则无法满足业务需求，比如需要复杂的剪切修改url，或是修改码流
此时可以通过自定义路由进行

内置自动路由规则基于同样的机制实现：

- 通过SPI声明VertxHttpDispatcher的扩展
- 通过getOrder声明相对其他dispatcher的执行顺序
- 自己能处理的url，拦截处理
  - 根据自定义规则获取目标微服务名、版本规则、新url等等参数
  - 通过EdgeInvocation转发请求
- 自己不能处理的url，放通给其他dispatcher处理

Metrics-架构

- 基于netflix spectator
- Foundation-metrics通过SPI机制加载所有MetricsInitializer实现，实现者可以通过MetricsInitializer中的getOrder规划执行顺序，order数字越小，越先执行。
- Metrics-core实现3类MetricsInitializer：
  - DefaultRegistryInitializer：实例化并注册spectator-reg-servo，设置较小的order，保证比下面2类MetricsInitializer先执行
  - Meters Initializer：实现TPS、时延、线程池、jvm资源等等数据的统计
  - Publisher：输出统计结果，内置了日志输出，以及通过RESTful接口输出
- Metrics-prometheus提供与prometheus对接的能力



Metrics-log

RESTful/prometheus输出内容为原始数据

日志输出内容是根据原始数据分析计算后的结果

- eventLoopContext-created

  如果持续增加，说明有计划外的线程切换，需要定位

- threadPool

  以线程池为单位进行统计，输出最小线程数、最大线程数、当前线程数、当前排队任务数，以及平均每秒提交任务数、完成任务数等等数据

- consumer

  以transport、调用结果两个维度进行分组，以operation为单位输出tps、平均时延、最大时延

- producer

  以transport、调用结果两个维度进行分组，以operation为单位输出tps、平均时延、最大时延、平均排队时间、最大排队时间、业务逻辑平均执行时间、最大执行时间

TODO: cpu/mem/gc.....

```
vertx:
  name       eventLoopContext-created
  registry   4
  transport  54
threadPool:
  corePoolSize maxThreads poolSize currentThreadsBusy queueSize taskCount completedTaskCount name
  8            8          8        0                  6         2839.0    2835.0             cse.executor.groupThreadPool-group1
  8            8          8        0                  0         3037.0    3036.0             cse.executor.groupThreadPool-group0
consumer:
  tps     latency(ms) max-latency(ms) operation
  rest.200:
  6060    1.635       29.688          perf1.impl.syncQuery
  53334   0.371       5.732           perf1.impl.asyncQuery
  59394   0.500       29.688
producer:
  tps     latency(ms) max-latency(ms) queue(ms) max-queue(ms) execute(ms) max-execute(ms) operation
  rest.200:
  6063    0.757       25.703          0.667     25.389        0.090       5.372           perf1.impl.syncQuery
  53334   0.016       2.712           0.002     0.363         0.014       2.710           perf1.impl.asyncQuery
  59397   0.092       25.703          0.070     25.389        0.022       5.372
```

性能调优

- 网络相关配置参数

	rest server verticle	rest client verticle	rest client max pool size	highway server verticle	highway client verticle
配置项	rest.server.thread-count	rest.client.thread-count	rest.client.connection.maxPoolSize	highway.server.thread-count	highway.client.thread-count
默认值	1	1	5	1	1
			每个client verticle实例中 针对同一个IP:PORT，最多同时建立的连接数		

  - 系统参数比较保守，需要根据实际情况设置合适的参数

- Producer线程池
  默认2个线程池，每个池中线程数等于cpu数
  - 参考metrics统计结果

- cpu
  带宽允许的情况下，cpu应该基本占满

- 网络带宽
  关注是否已经占满带宽

- RSS
  如果环境（硬件、os、驱动）支持RSS特性，确认队列数是否正确配置、中断是否均匀绑定了cpu、是否关闭了自动软中断负载均衡（不关闭会导致中断绑定，在一段时间后失效）

Thank You.

Website: http://servicecomb.incubator.apache.org/

Gitter: https://gitter.im/ServiceCombUsers/Lobby