**[学习微服务-第3天] ServiceComb内置高性能网关服务**

Edge Service是ServiceComb提供的JAVA网关服务。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。Edge Service以开发框架的形式提供，开发者可以非常简单的搭建一个Edge Service服务，通过简单的配置就可以定义路由转发规则。同时Edge Service支持强大的扩展能力，服务映射、请求解析、加密解密、鉴权等逻辑都可以通过扩展实现。

Edge Service本身也是一个微服务，需遵守所有微服务开发的规则。其本身可以部署为多实例，前端使用负载均衡装置进行负载分发；也可以部署为主备，直接接入用户请求。开发者可以根据Edge Service承载的逻辑和业务访问量、组网情况来规划。

**开发微服务网关**

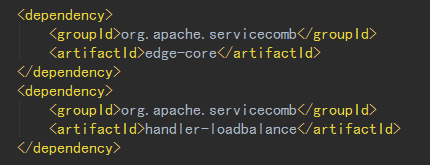
**搭建框架**

使用ServiceComb的内置Edge Service边缘服务

**3步完成搭建微服务网关**

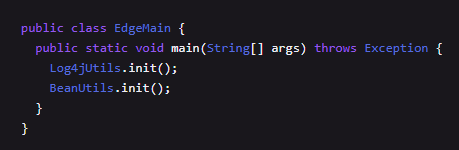
**•配置依赖关系**

在项目中加入edge-core的依赖，就可以启动Edge Service的功能。Edge Service在请求转发的时候，会经过处理链，因此还可以加入相关的处理链的模块的依赖，下面的实例增加的负载均衡的处理链，这个是必须的。



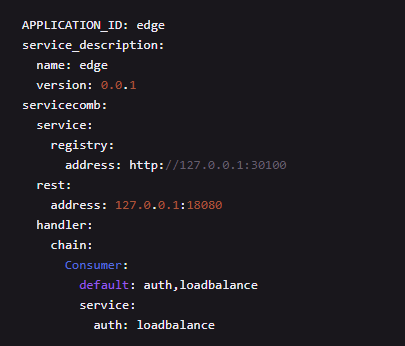
**•定义启动类**

和开发普通微服务一样，可以通过加载Spring的方式将服务拉起来。



**•增加配置文件microservie.yaml**

Edge Service本身也是一个微服务，遵循微服务查找的规则，自己也会进行注册。注意APPLICAIONT\_ID与需要转发的微服务相同。在下面的配置中，指定了Edge Service监听的地址，处理链等信息。其中auth处理链是DEMO项目中自定义的处理链，用于实现认证。同时auth服务本身，不经过这个处理链，相当于不鉴权。

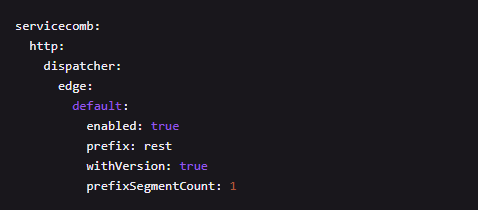


**定制路由规则**

使用Edge Service的核心工作是配置路由规则。场景不同，规则也不同。 路由规则由一系列AbstractEdgeDispatcher组成。Edge Service提供了几个常见的Dispatcher，通过配置即可启用，如果这些Dispatcher不满足业务场景需要，还可以自定义。

**•使用DefaultEdgeDispatcher**

DefaultEdgeDispatcher是一个非常简单、容易管理的Dispatcher，使用这个Dispatcher，用户不用动态管理转发规则，应用于实际的业务场景非常方便，这个也是推荐的一种管理机制。它包含如下几个配置项：



常见的这些配置项的示例及含义如下:

•[prefix=rest;withVersion=true;prefixSegmentCount=1]微服务xService提供的URL为: /xService/v1/abc，通过Edge访问的地址为/rest/xService/v1/abc，请求只转发到[1.0.0-2.0.0)版本的微服务实例。

•[prefix=rest;withVersion=true;prefixSegmentCount=2]微服务xService提供的URL为: /v1/abc，通过Edge访问的地址为/rest/xService/v1/abc，请求只转发到[1.0.0-2.0.0)版本的微服务实例。

•[prefix=rest;withVersion=true;prefixSegmentCount=3]微服务xService提供的URL为: /abc，通过Edge访问的地址为/rest/xService/v1/abc，请求只转发到[1.0.0-2.0.0)版本的微服务实例。

•[prefix=rest;withVersion=false;prefixSegmentCount=1]微服务xService提供的URL为: /xService/v1/abc，通过Edge访问的地址为/rest/xService/v1/abc，请求可能转发到任意微服务实例。•[prefix=rest;withVersion=false;prefixSegmentCount=2]微服务xService提供的URL为: /v1/abc，通过Edge访问的地址为/rest/xService/v1/abc，，请求可能转发到任意微服务实例。

•[prefix=rest;withVersion=false;prefixSegmentCount=2]微服务xService提供的URL为: /abc，通过Edge访问的地址为/rest/xService/abc，，请求可能转发到任意微服务实例。

withVersion配置项提供了客户端灰度规则，可以让客户端指定访问的服务端版本。Edge Service还包含根据接口兼容性自动路由的功能，请求会转发到包含了该接口的实例。假设某微服务，兼容规划为所有高版本必须兼容低版本，部署了以下版本实例：

1.0.0，提供了operation1

1.1.0，提供了operation1、operation2

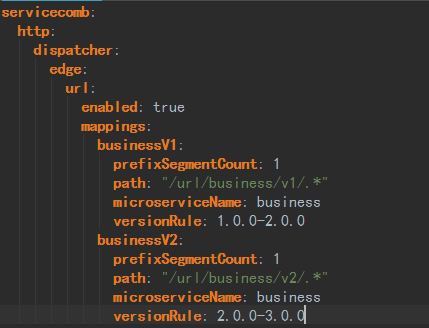
Edge Service在转发operation1时，会自动使用1.0.0+的规则来过滤实例

Edge Service在转发operation2时，会自动使用1.1.0+的规则来过滤实例

以上过程用户不必做任何干预，全自动完成，以避免将新版本的operation转发到旧版本的实例中去。

**•使用URLMappedEdgeDispatcher**

URLMappedEdgeDispatcher允许用户配置URL和微服务的映射关系。使用它可以非常灵活的定义哪些URL转发到哪些微服务。它包含如下几个配置项：



businessV1配置项表示的含义是将请求路径为/usr/business/v1/.的请求，转发到business这个微服务，并且只转发到版本号为1.0.0-2.0.0的实例（不含2.0.0）。转发的时候URL为/business/v1/.。path使用的是JDK的正则表达式，可以查看Pattern类的说明。prefixSegmentCount表示前缀的URL Segment数量，前缀不包含在转发的URL路径中。

有三种形式的versionRule可以指定。2.0.0-3.0.0表示版本范围，含2.0.0，但不含3.0.0；2.0.0+表示大于2.0.0的版本，含2.0.0；2.0.0表示只转发到2.0.0版本。2，2.0等价于2.0.0。从上面的配置可以看出，URLMappedEdgeDispatcher也支持客户端灰度。当然配置项会比DefaultEdgeDispatcher多。URLMappedEdgeDispatcher支持通过配置中心动态的修改配置，调整路由规则。

**•自定义Dispatcher**

**自定义Dispatcher包含两个步骤：**

1.实现AbstractEdgeDispatcher

2.通过SPI发布：增加文件META-INF/services/org.apache.servicecomb.transport.rest.vertx.VertxHttpDispatcher，并写入实现类

详细的代码细节可以参考下面的章节"DEMO功能说明"。开发者也可以参考DefaultEdgeDispatcher等代码来定义自己的Dispatcher。

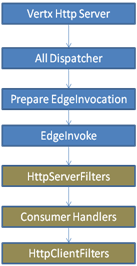
**•进行认证鉴权和其他业务处理**

通过Edge Servie工作流程可以看出，可以通过多种方式来扩展Edge Service的功能，包括Dispatcher、HttpServerFilter、Handler、HttpClientFilter等。比较常用和简单的是通过Handler来扩展。DEMO里面展示了如何通过Handler扩展来实现鉴权。详细的代码细节可以参考下面的章节"DEMO功能说明"。

**工作原理**

**工作流程**

**Edge Service的工作流程如下**



蓝色背景部分在Eventloop线程中执行，黄色背景部分：

•如果工作于reactive模式，则直接在Eventloop线程执行

•如果工作于线程池模式，则在线程池的线程中执行

**工作模式**

**reactive (默认)**

Edge Service默认工作于高性能的reactive模式，此模式要求工作于Edge Service转发流程中的业务代码不能有任何的阻塞操作，包括不限于：

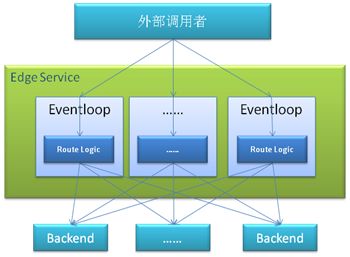
•远程同步调用，比如同步查询数据库、同步调用微服务，或是同步查询远程缓存等等

•任何的sleep调用

•任何的wait调用

•超大的循环

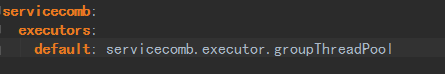
Edge Service的底层是基于netty的vertx，以上约束即是netty的reactive模式约束。



**线程池**

如果业务模型无法满足reactive要求，则需要使用线程池模式。

此时需要在Edge Service的microservice.yaml中配置：



这里的servicecomb.executor.groupThreadPool是ServiceComb内置的默认线程池对应的spring bean的beanId；业务可以定制自己的线程池，并声明为一个bean，其beanId也可以配置到这里。

**DEMO功能说明**

**请参考github上的edge service demo：**

https://github.com/ServiceComb/ServiceComb-Java-Chassis/tree/master/demo/demo-edge

**该demo包含以下工程：**

authentication：微服务：鉴权服务器

edge-service

hiboard-business-1.0.0微服务：business，1.0.0版本，operation add

hiboard-business-1.1.0微服务：business，1.1.0版本，operation add/dec

hiboard-business-2.0.0微服务：business，2.0.0版本，operation add/dec

hiboard-consumer作为一个普通的httpclient，而不是servicecomb consumer

hiboard-model非微服务，仅仅是一些公共的model

通过edge-service访问微服务business的不同版本，并确认是由正确的实例处理的。

**1.注册Dispatcher**

实现接口org.apache.servicecomb.transport.rest.vertx.VertxHttpDispatcher，或从org.apache.servicecomb.edge.core.AbstractEdgeDispatcher继承，实现自己的dispatcher功能。

实现类通过java标准的SPI机制注册到系统中去。

**Dispatcher需要实现2个方法：**

**•getOrder**

Dispatcher需要向vertx注入路由规则，路由规则之间是有优先级顺序关系的。

系统中所有的Dispatcher按照getOrder的返回值按从小到大的方式排序，按顺序初始化。

如果2个Dispatcher的getOrder返回值相同，则2者的顺序不可预知。

**•init**

init方法入参为vertx框架中的io.vertx.ext.web.Router，需要通过该对象实现路由规则的定制。

可以指定满足要求的url，是否需要处理cookie、是否需要处理body、使用哪个自定义方法处理收到的请求等等

更多路由规则细节请参考vertx官方文档：vertx路由机制https://vertx.io/docs/vertx-web/java/#\_routing\_by\_exact\_path

**提示：**

多个Dispatcher可以设置路由规则，覆盖到相同的url。

假设Dispatcher A和B都可以处理同一个url，并且A优先级更高，则：

•如果A处理完，既没应答，也没有调用RoutingContext.next()，则属于bug，本次请求挂死了

•如果A处理完，然后调用了RoutingContext.next()，则会将请求转移给B处理

**2.转发请求**

注册路由时，指定了使用哪个方法来处理请求（下面使用onRequest来指代该方法），在onRequest中实现转发逻辑。

**方法原型为：**

https://mmbiz.qpic.cn/mmbiz_png/jaoydu4O9Yf9dlbAkiaqIicT8KnZCiau3eGuFVR8SRtC06Jx0iasfn8IqF1CjzQUqcSPGNSnRlMiaJB0WI4PEOVK3fA/640?wx_fmt=png&tp5&wx_lazy=1&wx_co=1

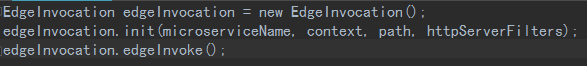
系统封装了org.apache.servicecomb.edge.core.EdgeInvocation来实现转发功能，至少需要准备以下参数：

•microserviceName，业务自行制定规则，可以在url传入，或是根据url查找等等

•context，即onRequest的入参

•path，转发目标的url

•httpServerFilters，Dispatcher父类已经初始化好的成员变量



edgeInvoke调用内部，会作为ServiceComb标准consumer去转发调用。

作为标准consumer，意味着ServiceComb所有标准的治理能力在这里都是生效的。

**3.设置兼容规则**

不同的业务可能有不同的兼容规划，servicecomb默认的兼容规则，要求所有新版本兼容旧版本。如果满足这个要求，则不必做任何特殊的设置。

**还有一种典型的规划：**

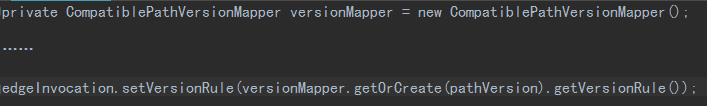
1.0.0-2.0.0内部兼容，url为/microserviceName/v1/….的形式

2.0.0-3.0.0内部兼容，url为/microserviceName/v2/….的形式

……

各大版本之间不兼容

此时，开发人员需要针对EdgeInvocation设置兼容规则：



versionMapper的作用是将v1或是v2这样的串，转为1.0.0-2.0.0或2.0.0-3.0.0这样的兼容规则。

**注意：**

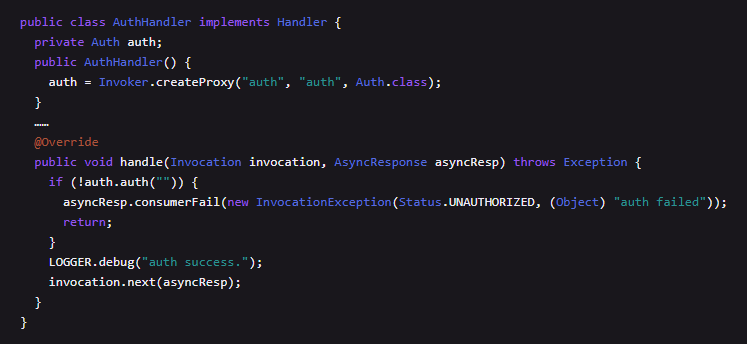
接口不兼容会导致非常多的问题。java chassis要求高版本服务兼容低版本服务，只允许增加接口不允许删除接口。在增加接口后，必须增加微服务的版本号。在开发阶段，接口变更频繁，开发者往往忘记这个规则。当这个约束被打破的时候，需要清理服务中心微服务的信息，并重启微服务和Edge Service(以及依赖于该微服务的其他服务)。否则可能出现请求转发失败等情况。

**4.鉴权**

Edge Service是系统的边界，对于很多请求需要执行鉴权逻辑。

基于标准的ServiceComb机制，可以通过handler来实现这个功能。

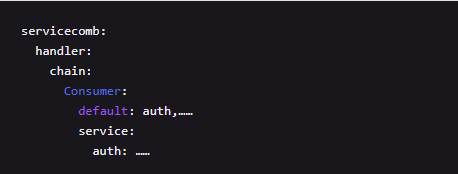
**最简单的示意代码如下：**



Auth表示是鉴权微服务提供的接口，Invoker.createProxy("auth", "auth", Auth.class)是透明RPC开发模式中consumer的底层api，与@ReferenceRpc是等效，只不过不需要依赖spring bean机制。

Auth接口完全由业务定义，这里只是一个示例。

Handler开发完成后，配置到edge service的microservice.yaml中：



这个例子，表示转发请求给所有的微服务都必须经过鉴权，但是调用鉴权微服务时不需要鉴权。