Spring Cloud Netflix概览和架构设计

2017-04-08 Mavlarn Docker

Spring Cloud简介

Spring Cloud是基于Spring Boot的一整套实现微服务的框架。他提供了微服务开发所需的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等组件。最重要的是，跟spring boot框架一起使用的话，会让你开发微服务架构的云服务非常好的方便。

Spring Cloud包含了非常多的子框架，其中，Spring Cloud Netflix是其中一套框架，由Netflix开发后来又并入Spring Cloud大家庭，它主要提供的模块包括：服务发现、断路器和监控、智能路由、客户端负载均衡等。

Spring Cloud Netflix项目的时间还不长，并入Spring Cloud大家族还是2年前，所以相关的使用文档还比较少，除了官方文档，国内也有一个中文社区。但是，如果是刚开始接触这个，想使用它搭建一套微服务的应用架构，总是会有不知如何下手的感觉。所以，这篇文章就是从整体上来看看这个框架的各个组件、用处是什么、如何相互作用。最后再结合实际的经验，介绍一下可能会出现的问题，以及针对一些问题，采用什么样的解决方案。

微服务架构

首先，我们来看看一般的微服务架构需要的功能或使用场景：

- 我们把整个系统根据业务拆分成几个子系统。
- 每个子系统可以部署多个应用，多个应用之间使用负载均衡。
- 需要一个服务注册中心，所有的服务都在注册中心注册，负载均衡也是通过在注册中心注册的服务来使用一定策略来实现。
- 所有的客户端都通过同一个网关地址访问后台的服务，通过路由配置，网关来判断一个URL请求由哪个服务处理。请求转发到服务上的时候也使用负载均衡。
- 服务之间有时候也需要相互访问。例如有一个用户模块，其他服务在处理一些业务的时候，要获取用户服务的用户数据。
- 需要一个断路器，及时处理服务调用时的超时和错误，防止由于其中一个服务的问题而导致整体系统的瘫痪。
- 还需要一个监控功能，监控每个服务调用花费的时间等。

Spring Cloud Netflix组件以及部署

Spring Cloud Netflix框架刚好就满足了上面所有的需求，而且最重要的是，使用起来非常的简单。Spring Cloud Netflix包含的组件及其主要功能大致如下：

- Eureka，服务注册和发现，它提供了一个服务注册中心、服务发现的客户端，还有一个方便的查看所有注册的服务的界面。 所有的服务使用Eureka的服务发现客户端来将自己注册到Eureka的服务器上。
- Zuul，网关，所有的客户端请求通过这个网关访问后台的服务。他可以使用一定的路由配置来判断某一个URL由哪个服务来处理。并从Eureka获取注册的服务来转发请求。
- Ribbon，即负载均衡，Zuul网关将一个请求发送给某一个服务的应用的时候，如果一个服务启动了多个实例，就会通过Ribbon来通过一定的负载均衡策略来发送给某一个服务实例。

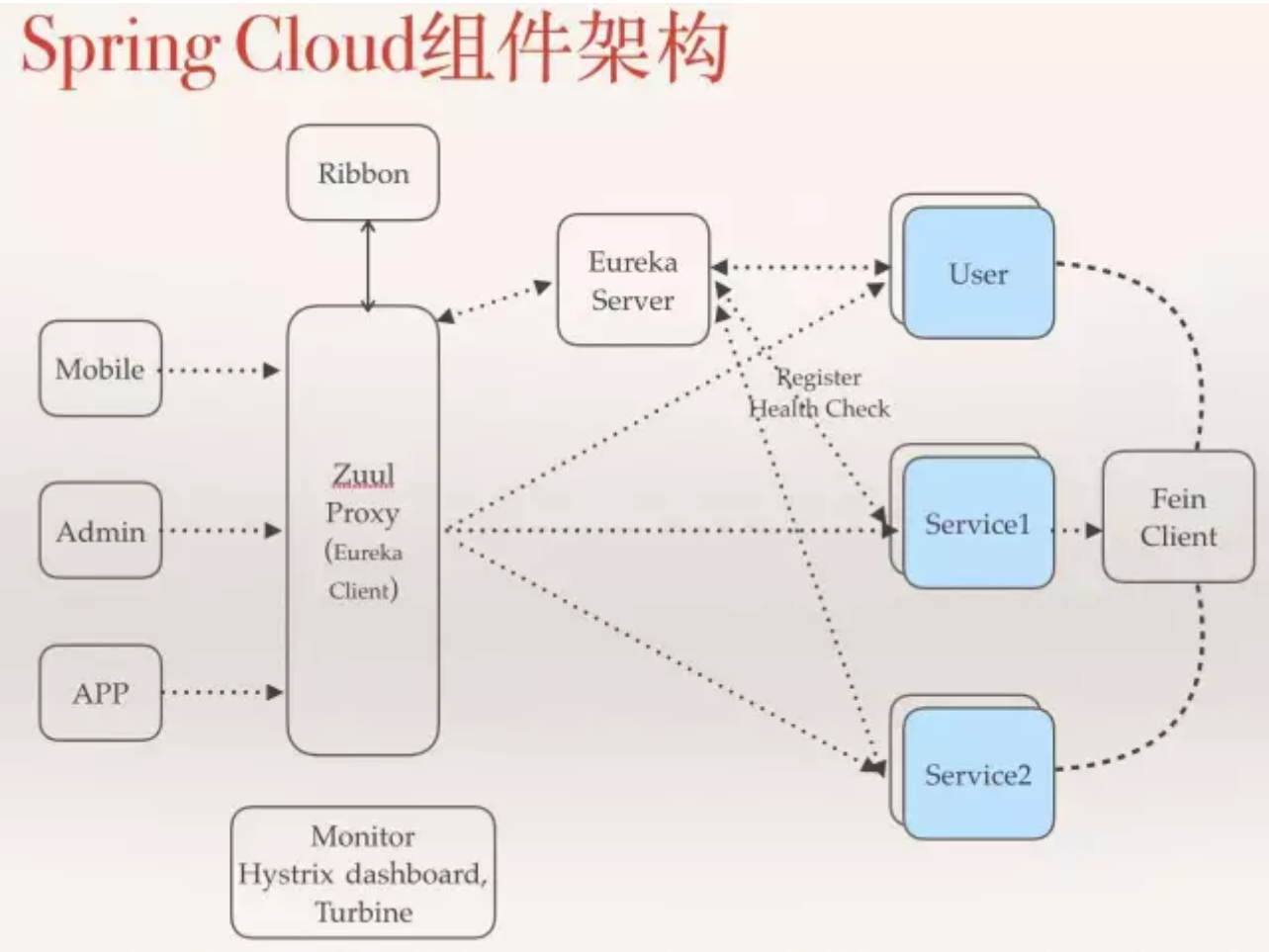
- Feign，服务客户端，服务之间如果需要相互访问，可以使用RestTemplate，也可以使用Feign客户端访问。它默认会使用Ribbon来实现负载均衡。

- Hystrix，监控和断路器。我们只需要在服务接口上添加Hystrix标签，就可以实现对这个接口的监控和断路器功能。

- Hystrix Dashboard，监控面板，他提供了一个界面，可以监控各个服务上的服务调用所消耗的时间等。

- Turbine，监控聚合，使用Hystrix监控，我们需要打开每一个服务实例的监控信息来查看。而Turbine可以帮助我们把所有的服务实例的监控信息聚合到一个地方统一查看。这样就不需要挨个打开一个个的页面一个个查看。

下面就是使用上述的子框架实现的为服务架构的组架构图：



在上图中，有几个需要说明的地方：

- Zuul网关也在注册中心注册，把它也当成一个服务来统一查看。 负载均衡不是一个独立的组件，它运行在网关、服务调用等地方，每当需要访问一个服务的时候，就会通过Ribbon来获得一个该服务的实例去掉用。Ribbon从Eureka注册中心获得服务和实例的列表，而不是发送每个请求的时候从注册中心获得。

- 我们可以使用RestTemplate来进行服务间调用，也可以配置FeignClient来使用，不管什么方式，只要使用服务注册，就会默认使用Ribbon负载均衡。（RestTemplate需要添加@LoadBalanced）

- 每个服务都可以开启监控功能，开启监控的服务会提供一个servlet接口/hystrix.stream，如果你需要监控这个服务的某一个方法的运行统计，就在这个方法上加一个@HystrixCommand的标签。

- 查看监控信息，就是在Hystrix Dashboard上输入这个服务的监控url: http://serviceIp:port/hystrix.stream，就可以用图表的方式查看运行监控信息。

- 如果要把所有的服务的监控信息聚合在一起统一查看，就需要使用Turbine来聚合所需要的服务的监控信息。

我们也可以从上图中看出该架构的部署方式:

- 独立部署一个网关应用

- 服务注册中心和监控可以配置在一个应用里，也可以是2个应用。

- 服务注册中心也可以部署多个，通过区域zone来区分，来实现高可用。

- 每个服务，根据负载和高可用的需要，部署一个或多个实例。

Spring Cloud Netflix组件开发

上面说到，开发基于Spring Cloud Netflix的微服务非常简单，一般我们是和Spring Boot一起使用，如果你想在自己原先的Java Web应用中使用也可以通过添加相关配置来实践。

有关开发的详细内容，可以参考Spring Cloud中文社区的这个系列文章，里面详细介绍了每一

种组件的开发。这里，就只是来看一下服务注册中和监控模块的开发，还有服务调用的开发，其他的可以直接参考上面的系列文章。

注册和监控中心的开发

这个非常简单，就下面一个类：

```
// 省略import
@SpringBootApplication
@EnableEurekaServer
@EnableHystrixDashboard
public class ApplicationRegistry {
public static void main(String[] args) {
    new SpringApplicationBuilder(Application.class).web(true).run
}
}
```

这里使用Spring Boot标签的@SpringBootApplication说明当前的应用是一个Spring Boot应用。这样我就可以直接用main函数在IDE里面启动这个应用，也可以打包后用命令行启动。当然也可以把打包的war包用Tomcat之类的服务器启动。

使用标签@EnableEurekaServer，就能在启动过程中启动Eureka服务注册中心的组件。它会监听一个端口，默认是8761，来接收服务注册。并提供一个Web页面，打开以后，可以看到注册的服务。

添加@EnableHystrixDashboard就会提供一个监控的页面，我们可以在上面输入要监控的服务的地址，就可以查看启用了Hystrix监控的接口的调用情况。

当然，为了使用上面的组件，我们需要在Maven的POM文件里添加相应的依赖，比如使用spring-boot-starter-parent，依赖spring-cloud-starter-eureka-server和spring-cloud-starter-hystrix-dashboard等。

服务间调用

在网上的各种文档中，对服务间调用，都没有说明的很清楚，所以这里特别说明一下这个如何

开发。

有两种方式可以进行服务调用，RestTemplate和FeignClient。不管是什么方式，他都是通过REST接口调用服务的http接口，参数和结果默认都是通过jackson序列化和反序列化。因为Spring MVC的RestController定义的接口，返回的数据都是通过Jackson序列化成JSON数据。

RestTemplate

使用这种方式，只需要定义一个RestTemplate的Bean，设置成LoadBalanced即可：

```
@Configuration
public class SomeCloudConfiguration {
@LoadBalanced
@Bean
RestTemplate restTemplate() {
    return new RestTemplate();
}
}
```

这样我们就可以在需要用的地方注入这个bean使用：

```
public class SomeServiceClass {
@Autowired
private RestTemplate restTemplate;
public String getUserById(Long userId) {
    UserDTO results = restTemplate.getForObject("http://users/get
    return results;
}
}
```

其中，users是服务ID，Ribbon会从服务实例列表获得这个服务的一个实例，发送请求，并获得结果。对象UserDTO需要序列号，它的反序列号会自动完成。

FeignClient

除了上面的方式，我们还可以用FeignClient。还是直接看代码：

```
@FeignClient(value = "users", path = "/users")
public interface UserCompositeService {
@RequestMapping(value = "/getUserDetail/{id}",
        method = RequestMethod.GET,
        produces = MediaType.APPLICATION_JSON_VALUE)
UserDTO getUserById(@PathVariable Long id);
}
```

我们只需要使用@FeignClient定义一个接口，Spring Cloud Feign会帮我们生成一个它的实现，从相应的users服务获取数据。

其中，@FeignClient(value = "users", path = "/users/getUserDetail")里面的value是服务ID，path是这一组接口的path前缀。

在下面的方法定义里，就好像设置Spring MVC的接口一样，对于这个方法，它对应的URL是/users/getUserDetail/{id}。

然后，在使用它的时候，就像注入一个一般的服务一样注入后使用即可：

```
public class SomeOtherServiceClass {
@Autowired
private UserCompositeService userService;
public void doSomething() {
    // .....
    UserDTO results = userService.getUserById(userId);
    // other operation...
```

```
}
}
```

遇到的问题

由于Spring Cloud说明文档较少，微服务的架构相对来说也比较复杂，在开发的时候，难免会遇到很多问题，有一些是如何更好地使用这套框架去搭建架构，也有一些问题是如何配置。这里就一些我在搭建微服务架构的时候遇到的问题提供一些方法。

请求超时问题

Zuul网关默认的超时时间非常短，这是为了保证调用服务的时候能够很快的响应。但是，我们会有一些业务方法运行的时间比较长，特别是在测试服务器。这时候，就需要调整超时时间。这个超时有几个地方：

1. 负载均衡Ribbon，负载均衡有一个超时的设置，包括链接时间和读取时间

2. Hystrix断路器也有一个超时设置，它需要在适当的时候返回，而不是一直等在一个请求上。

对应的配置如下：

```
hystrix.command.default.execution.isolation.thread.timeoutInMilli
ribbon:
ReadTimeout: 30000
ConnectTimeout: 15000
```

服务ID的问题

服务的ID，也就是服务名，可以通过在application.yml或者bootstrap.yml里面设置：

```
spring:
application:
    name: users
```

管理路径的问题

Spring Boot的应用默认都是开放一些管理的接口，如/info、/health和metrics监控的接口/metrics等。如果你使用默认的路径，使用Hystrix监控、服务注册中心的监听服务状态都不会有问题，但是，如果你想使用别的路径，例如/management/info、/management/health，那就牵扯到很多地方，而且，每个版本可能会或多或少的有一些问题，导致你遇到的问题还会不一样。我遇到过的问题有：

注册成功却找不到服务

首先，注册可以成功，在Eureka服务器页面上也可以看到各个服务。但是，当你通过网关调用的时候，却总是提示服务找不到。这时候可能就需要在每个服务的application.yml里面进行如下配置：

```
eureka:
instance:
    nonSecurePort: ${server.port}
    appname: ${spring.application.name}
    statusPageUrlPath: ${management.context-path}/info
    healthCheckUrlPath: ${management.context-path}/health
```

简单来说，这就是告诉在注册的时候，同时告诉Eureka服务器，服务的端口是什么，用来监听状态的路径是什么。这是因为我们使用了不同的管理接口路径，而Eureka服务器没有使用相应的路径。

如果一切正常，你在Eureka服务器上点击一个注册的服务，应该能打开一个info页面。他可能是空白的，但是，至少Eureka服务器能通过这个知道服务的运行正常。

这个问题也不是在所有的版本都存在，只是在某一些Spring Cloud的版本存在。

设置了管理路径的Hystrix监控

刚才说了Hystrix监控的路径是http://serviceIp:port/hystrix.stream，如果你设置了管理接口的路径，那么这个监控路径也会变成：

```
http://serviceIp:port/${management.context-path}/hystrix.stream
```

如果这时候，你再想使用Turbine聚合，Turbine就会找不到了，因为它默认使用Eureka服务器上的服务器地址和端口，在后面添加/hystrix.stream。这时候，你就需要设置Turbine：

```
turbine:
aggregator:
    clusterConfig: USER
appConfig: USER
instanceUrlSuffix:
    USER: /user/hystrix.stream
```

管理路径的安全性

对于微服务部署的几台机器，可以通过开通防火墙来控制谁可以访问管理接口，但是，即使是这样，为了安全性等，我一般还是会把管理端接口也用Spring Security来保护。这样一来，监控接口就没法直接访问了。

服务间调用的权限验证

一般我们的API接口都需要某种授权才能访问，登陆成功以后，然后通过token或者cookie等方式才能调用接口。

使用Spring Cloud Netfix框架的话，登录的时候，把登录请求转发到相应的用户服务上，登陆成功后，会设置cookie或header token等。然后客户端接下来的请求就会带着这些验证信息，从Zuul网关传到相应的服务上进行验证。

Zuul网关在把请求转发到后台的服务的时候，会默认把一些header传到服务端，如：Cookie、

Set-Cookie、Authorization。这样，客户端请求的相关headers就可以传递到服务端，服务端设置的cookie也可以传到客户端。

但是，如果你想禁止某些header透传到服务端，可以在Zuul网关的application.yml配置里通过下面的方式禁用：

```
zuul:
routes:
users:
 path: /users/**
 sensitiveHeaders: Cookie,Set-Cookie,Authorization
 serviceId: user
```

刚才说了我们的某个服务有时候需要调用另一个服务，这时候，这个请求不是客户端发起，他的请求的header里面也不会有任何验证信息。这时候，要么，通过防火墙等设置，保证服务间调用的接口，只能某几个地址访问；要么，就通过某种方式设置header。

同时，如果你想在某个服务里面获得这个请求的真是IP，（因为请求的通过网关转发而来，你直接通过request获得ip得到的是网关的IP），就可以从headerX-Forwarded-Host获得。如果想禁用这个header，也可以：

```
zuul.addProxyHeaders = false
```

如果你使用RestTemplate的方式调用，可以在请求里面添加一个有header的Options。

也可以通过如下的拦截器的方式设置，它对RestTemplate方式和FeignClient的方式都可以起作用：

```
@Bean
public RequestInterceptor requestInterceptor() {
return new RequestInterceptor() {
    @Override
```

```
    public void apply(RequestTemplate template) {
        String authToken = getToken();
        template.header(AUTH_TOKEN_HEADER, authToken);
    }
};
}
```

3 天烧脑式Kubernetes训练营

本次培训涉及：Kubernetes概述和架构、部署和核心机制分析、进阶篇——Kubernetes工作原理和代码分析等，扫描下方二维码可查看具体培训内容。

点击阅读原文链接可直接报名。

阅读原文