摘要

本文档描述的是我这半个月来研究微服务的成果。微服务进程挂在在Tomcat服务器上运行，本文档描述的都是微服务调用微服务，将从微服务的定义开始引出Srping-Cloud。微服务之间可以通过地址直接相互调用，但是这并不是生产友好的做法。因此需要在微服务之间添加一个中间件来管理所有微服务的地址，调用需要从中间件查询目标地址才能调用。这个中间件可以通过在Spring Application中集成Eureka来实现。这就是服务发现组件

服务发现组件可以互相注册，形成集群，这样可以保持发现组件的稳定性。

在服务发现组件中，可以注册多个同名的，但地址不同的微服务，多个微服务的同时存在可以实现分布部署，实现负载均衡。

在微服务中的负载均衡通过在客户端中整合Ribbon实现,这样可以很方便地实现负载均衡。

在调用的时候，需要构造URL，但是如果参数很多的时候就不是很方便。因此可以在微服务客户端中整合Feign来显示地声明接口，直接调用。

在生产中，一个应用可以直接调用微服务实现功能，但是实际中，一个大功能的实现可能需要很多小功能，需要调用很多微服务，会产生很多流量，如果是一个手机APP，这些流量是可以避免的，因此在用户与服务之间增加一个网关，这样就可以隔离用户和服务群。

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

微服务概述

微服务架构风格是一种将应用程序开发为一组小型微服务的方法，每个服务运行在自己的进程中,服务间通信采用轻量级通信机制。

简单地说就是将一个由多个模块的应用分割为多个小型服务，各服务之间通过轻量级的通信机制通信.

Spring-Cloud概述

本文档讲述的微服务是基于Spring-Cloud及其组件来实现的，本文档将展示一个电影微服务调用用户微服务的例子。每个项目都是基于Maven工程实现。

这是在Maven中集成Spring-Cloud的代码,其中JPA与MySql不是必须的。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>microservice-marven</groupId>
    <artifactId>microservice-marven</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- MySql数据库驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Camden.SR4</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
        </plugins>
    </build>

</project>
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

本节将展示第一个例子，实现两个相互调用的微服务。没有中间发现服务器。(详见Github第一个例子)

服务注册与发现

本节讲解如何通过在微服务中集成Eureka实现服务注册.

Eureka Server

编写Eureka Server

首先来编写发现服务器.

创建一个Maven工程.在Pom中添加依赖.由于不需要数据库，所以不需要添加数据库相关的依赖，也不需要添加web依赖.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
    </dependency>
</dependencies>
```

在resource包中添加文件application.yml。

```
server:
  port: 8761
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone:  http://localhost:8761/eureka/
```

在java包中添加EurekaApplication.java

```
package app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/**
 * Created by 吴文韬 on 2017/9/26.
 */

@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {
    public static void main(String[] args){
        SpringApplication.run(EurekaApplication.class,args);
    }
}
```

这样一个发现服务就编写完成了。

编写Provider User微服务

首先创建一个Maven工程，也可以创建Spring-initilizer工程。编写Pom文件.其中actuator可以监视一些情况，可以暂时不管，但其他的依赖必须添加。spring-cloud-starter-eureka-server是为了能够将自己注册到发现组件上，也可以写成spring-cloud-starter-eureka

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

编写配置文件application.yml

```
server:
  port: 8000
spring:
  application:
    name: microservice-provider-user
  jpa:
    generate-ddl: false
    show-sql: true
    hibernate:
      ddl-auto: none
  datasource:
    driverClassName:  com.mysql.jdbc.Driver
    url:  jdbc:mysql://127.0.0.1:3306/test?useSSL=false
    username: root
    password: 123456
logging:                                # 配置日志级别，让hibernate打印出执行的SQL
  level:
    root: INFO
    org.hibernate: INFO
    org.hibernate.type.descriptor.sql.BasicBinder: TRACE
    org.hibernate.type.descriptor.sql.BasicExtractor: TRACE
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

在java包中创建与数据库对应的类，User与UserRepository.java来访问数据库。

```
// User
package app.entity;

import javax.persistence.*;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
    private Long id;

    @Column
    private String username;

    @Column
    private String email;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }


}
```

```
package app.entity;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@Repository
public interface UserRepository extends JpaRepository<User,Long> {
}
```

编写UserController来实现外部访问

```
package app.controller;

import app.entity.User;
import app.entity.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@RestController
public class UserController {
    @Autowired
    private UserRepository userRepository;

    @GetMapping("/{id}")
    public User findById(@PathVariable Long id){
        User findOne=this.userRepository.findOne(id);
        return findOne;
    }
}
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

编写主程序入口,ProviderUserApplication

```
package app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@SpringBootApplication
@EnableDiscoveryClient
public class ProviderUserApplication {

    public static void main(String[] args){
        SpringApplication.run(ProviderUserApplication.class,args);
    }
}
```

编写服务客户端microservice-consumer-movie

新建工程，编写Pom依赖如下。其他部分省略

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
</dependencies>
```

编写application.yml

```
server:
  port: 8010
spring:
  application:
    name: microservice-consumer-movie
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

编写User.java

```
package app.entity;

import javax.persistence.*;

/**
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
 * Created by 吴文韬 on 2017/9/26.
 */
public class User {
    private Long id;
    private String username;
    private String email;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

编写MovieController.java

```
package app.controller;

import app.entity.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@RestController
public class MovieController {
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/user/{id}")
    public User findById(@PathVariable Long id) {
        return this.restTemplate.getForObject("http://localhost:8000/" + id, User.class);
    }
}
```

编写ConsumerMovieApplication.java

```
package app;

import app.entity.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.context.annotation.Bean;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
```

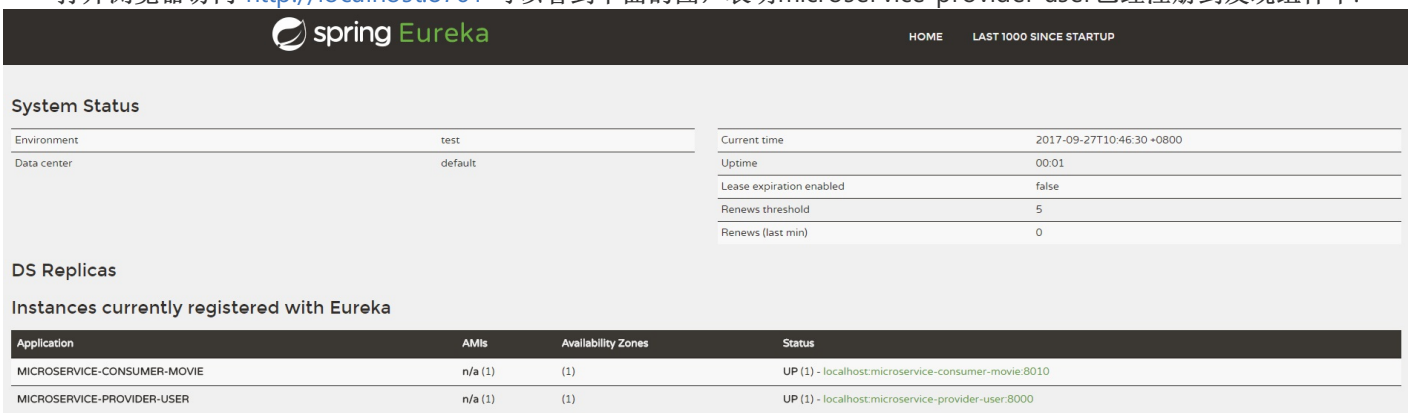

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
/**
 * Created by 吴文韬 on 2017/9/26.
 */
@SpringBootApplication
@EnableDiscoveryClient
public class ConsumerMovieApplication {
    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

    public static void main(String[] args){
        SpringApplication.run(ConsumerMovieApplication.class,args);
    }
}
```

运行程序

运行 microservice-discovery-eureka
运行 microservice-provider-user
运行 microservice-consumer-movie
打开浏览器访问 http://localhost:8761 可以看到下面的图，表明microservice-provider-user已经注册到发现组件中.



访问 http://localhost:8010 返回如下内容

```
{"id":1,"username":"吴文韬","email":"lkysyzxz@outlook.com"}
```

这样一个发现组件就编写完成了，不过在microservice-consumer-movie中使用的还是硬编码。

Eureka Server 集群

取上一个例子中的microservice-discovery-eureka修改

修改application.yml

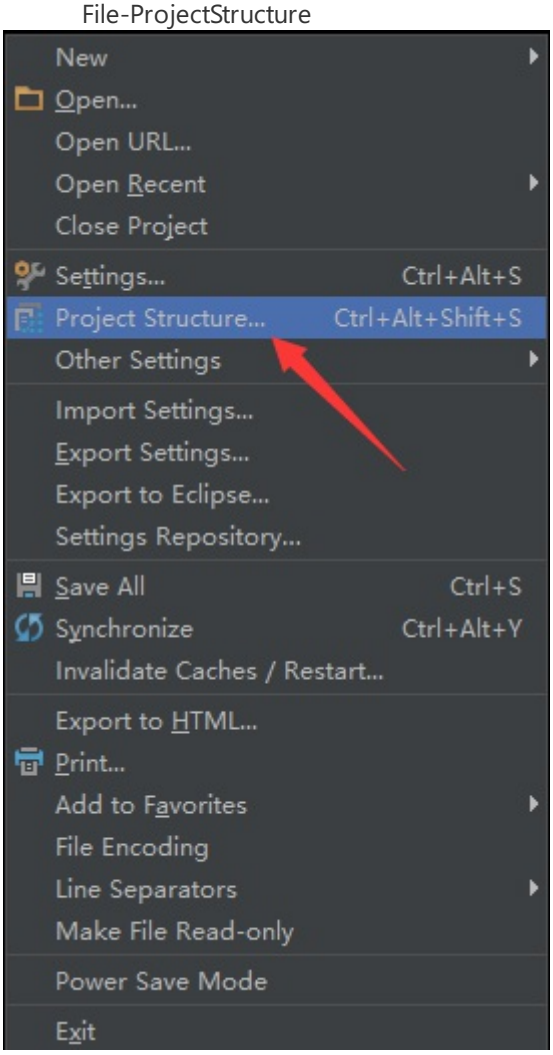
```
spring:
  application:
    name: microservice-discovery-eureka-ha
---
spring:
  profiles: peer1                                # 指定profile=peer1
server:
  port: 8761
eureka:
  instance:
    hostname: localhost                            # 指定当profile=peer1时，主机名是peer1
  client:
    serviceUrl:
      defaultZone: http://localhost:8762/eureka/      # 将自己注册到peer2这个Eureka上面去
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
---
spring:
  profiles: peer2
server:
  port: 8762
eureka:
  instance:
    hostname: localhost
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

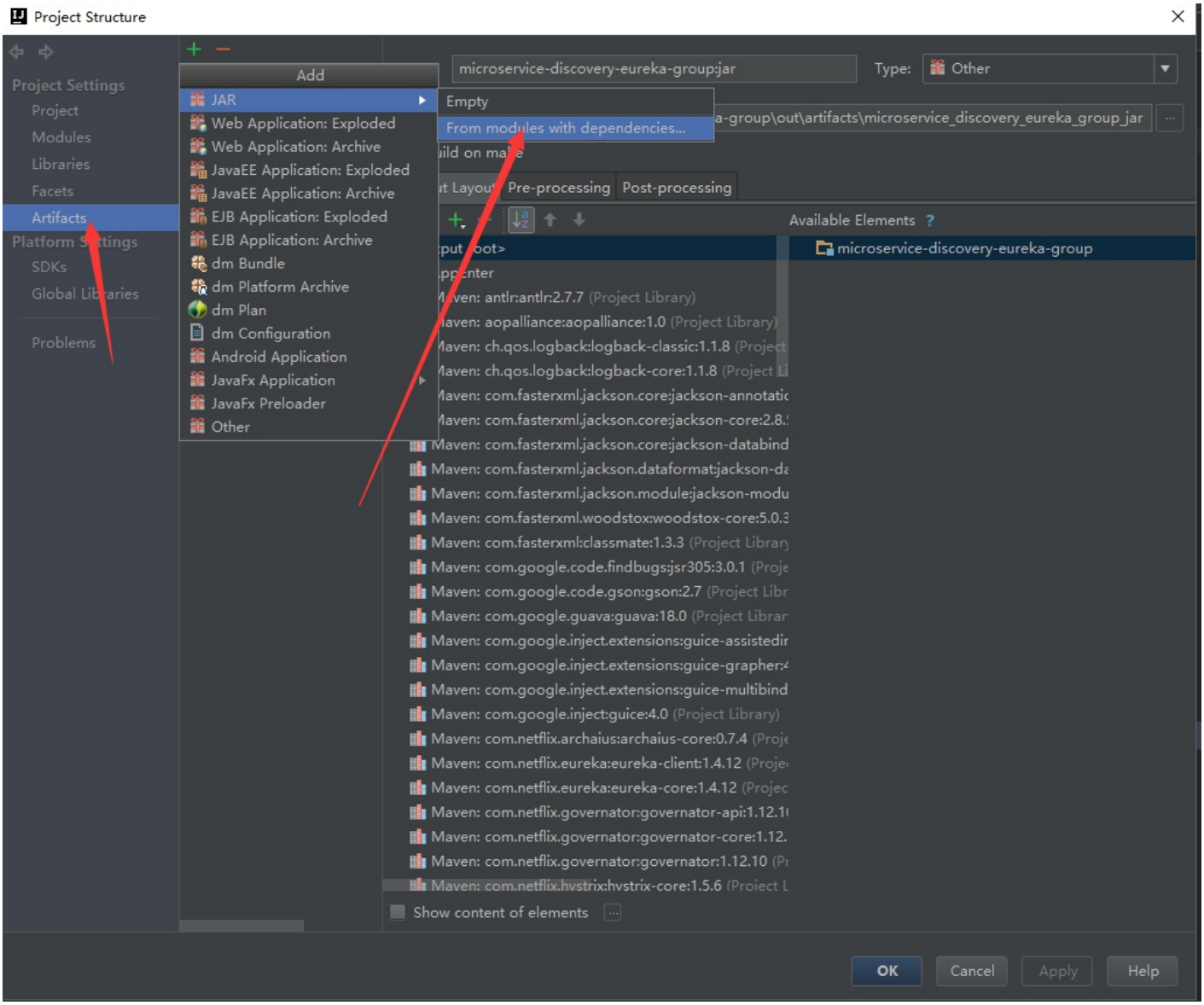
将工程打包成jar包输出

File-ProjectStructure



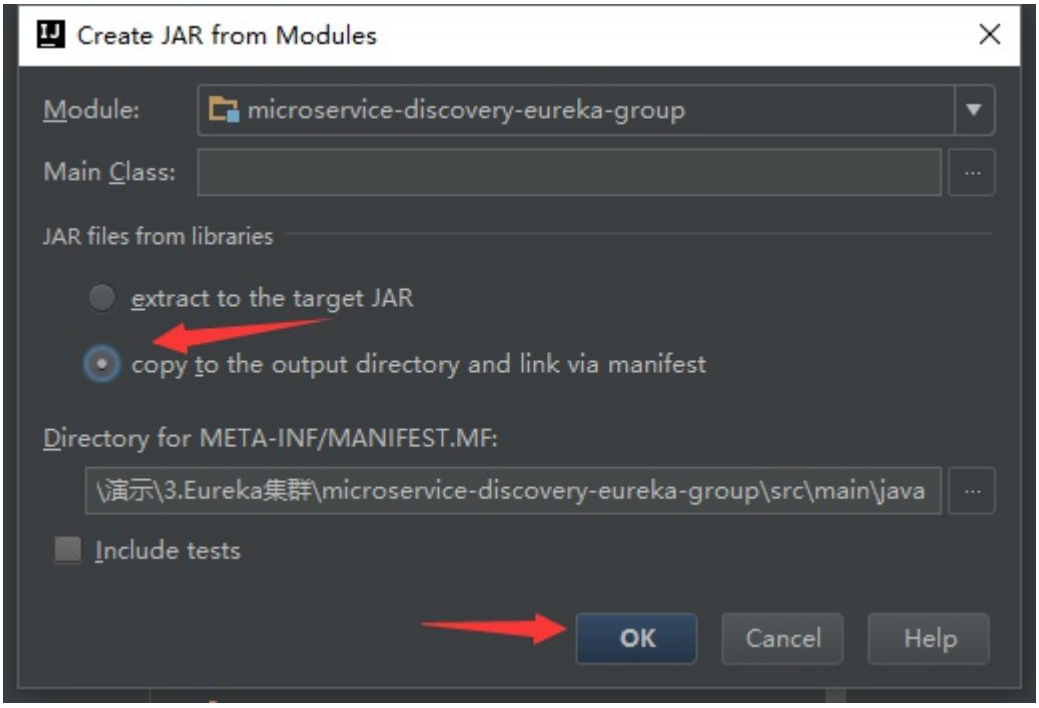
添加jar包，选择From model with depencies,把所依赖的jar包打包在一起

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪



按如图所示选择

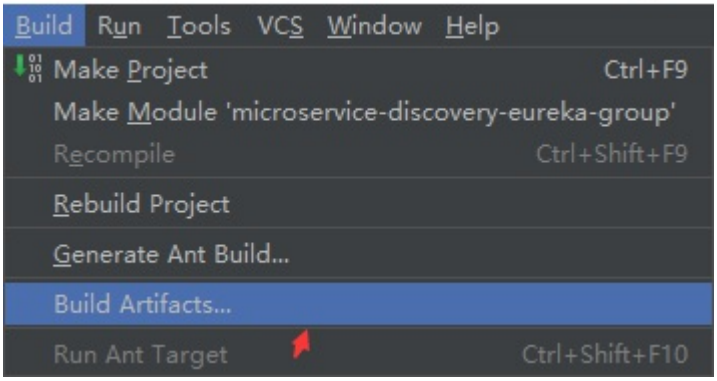
© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪



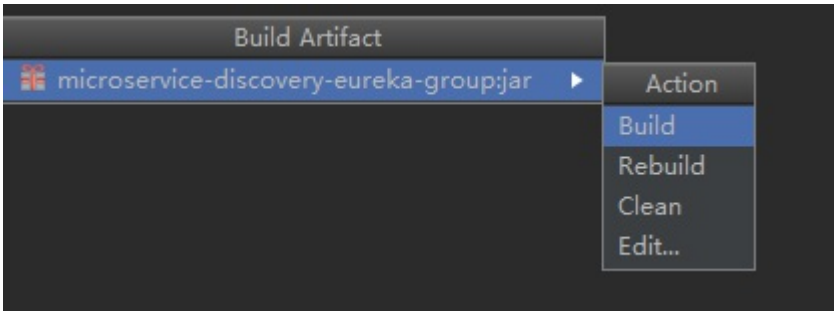
图中microservice-discovery-eureka.jar就是包含主程序的jar包,点ok

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

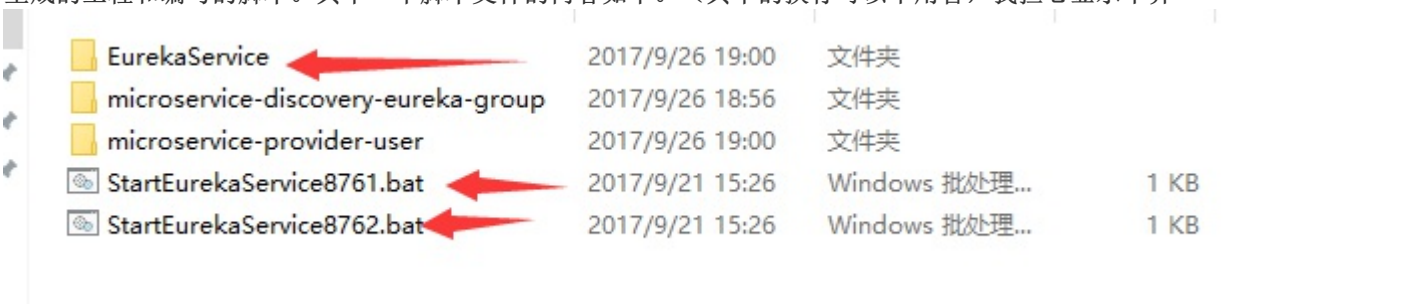
开始生成



© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

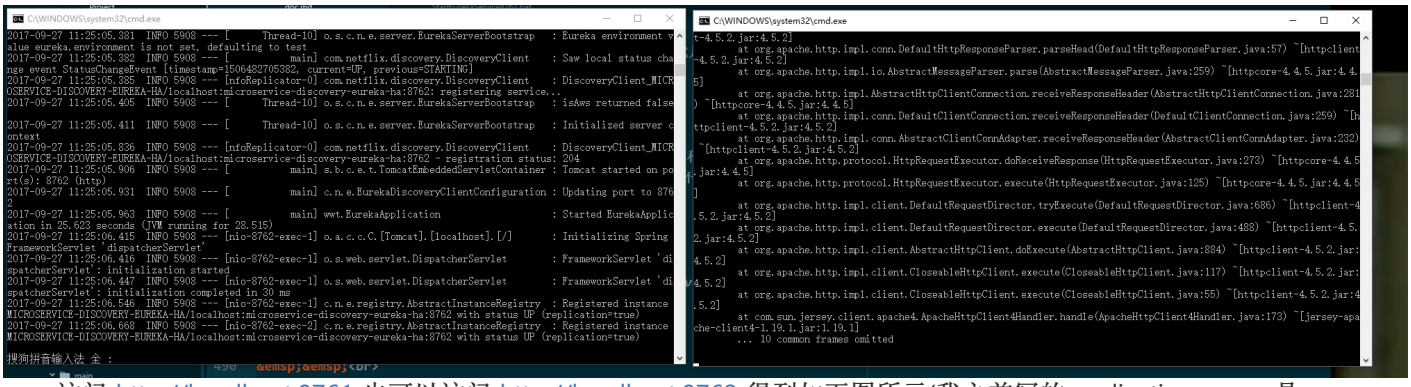


在工程目录中找到out文件夹，在里边把主程序和他的依赖jar包一起复制出来，编写Windows/Linux脚本,这是我之前生成的工程和编写的脚本。其中一个脚本文件的内容如下。（其中的换行可以不用管，我担心显示不齐

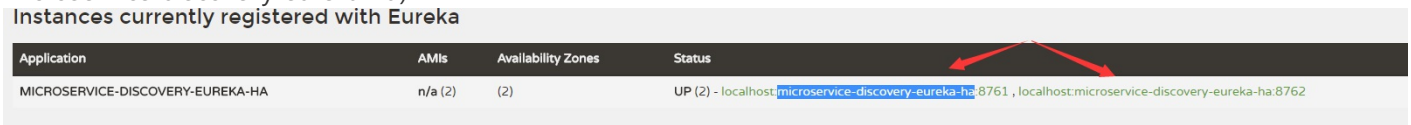



```
java -jar
C:\Users\吴文韬\Desktop\批处理指令\
EurekaService\microservice_discovery_eureka_jar/microservice-discovery-eureka.jar
--spring.profiles.active=peer1
```

接着运行两个不同的脚本.得到如下图所示



访问 http://localhost:8761 也可以访问 http://localhost:8762 得到如下图所示(我之前写的application name 是 microservice-discovery-eureka-ha)



使用Ribbon实现负载均衡

Ribbon需要整合在客户端中，在microservice-consumer-movie中添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
```

修改ConsumerMovieApplication中的restTemplate,增加注解@LoadBalanced

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

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

修改MovieController，使其可以查看当前选择的服务地址

```
package app.controller;

import app.entity.User;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

/**
 * Created by 吴文韬 on 2017/9/26.
 */
@RestController
public class MovieController {
    private static final Logger LOGGER = LoggerFactory.getLogger(MovieController.class);

    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private LoadBalancerClient loadBalancerClient;

    @GetMapping("/user/{id}")
    public User findById(@PathVariable Long id) {
        return this.restTemplate.getForObject("http://microservice-provider-user/" + id, User.class);
    }

    @GetMapping("/log-instance")
    public void logUserInstance(){
        ServiceInstance serviceInstance = this.loadBalancerClient.choose("microservice-provider-user");
        MovieController.LOGGER.info("{}:{}:{}",serviceInstance.getServiceId(),serviceInstance.getHost(),serviceInstance.getPort());
    }
}
```

Feign实现声明式REST调用

在前例中的调用，使用字符串来拼接调用地址，这样很不方便，Spring-Cloud中添加了Feign组件，可以显示声明调用接口.需要在客户端中添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
    <version>1.2.4.RELEASE</version>
</dependency>
```

Feign不仅能简化调用接口，还整合了Ribbon，添加了@FeignClient注解，同时会添加Ribbon相关注解

Feign简化调用接口

在microservice-consumer-movie中添加依赖

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
    <version>1.2.4.RELEASE</version>
</dependency>
```

修改ConsumerMovieApplication

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class ConsumerMovieApplication {
    public static void main(String[] args){
        SpringApplication.run(ConsumerMovieApplication.class,args);
    }
}
```

创建Feign接口

```
@FeignClient(name = "microservice-provider-user")
public interface UserFeignClient {
    @RequestMapping(value = "/{id}",method = RequestMethod.GET)
    public User findById(@PathVariable("id") Long id);
}
```

修改MovieController，让其使用UserFeignClient接口

```
@RestController
public class MovieController {
    private static final Logger LOGGER = LoggerFactory.getLogger(MovieController.class);

    @Autowired
    private UserFeignClient userFeignClient;

    @Autowired
    private LoadBalancerClient loadBalancerClient;

    @GetMapping("/user/{id}")
    public User findById(@PathVariable Long id) {
        return this.userFeignClient.findById(id);
    }

    @GetMapping("/log-instance")
    public void logUserInstance(){
        ServiceInstance serviceInstance = this.loadBalancerClient.choose("microservice-provider-user");
        MovieController.LOGGER.info("{}:{}:{}",serviceInstance.getServiceId(),serviceInstance.getHost(),serviceInstance.getPort());
    }
}
```

这样调用就比较简单了.

Feign实现多参数接口

在生产中往往会涉及多个参数
下面我将修改microservice-provider-user和microservice-consumer-movie来实现多参数接口

GET方法

在microservice-provider-user中的UserController中添加一下代码.

```
@GetMapping("/user/get1")
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
public User get1(User user){
    return user;
}

@GetMapping("/user/get2")
public User get2(User user){
    return user;
}
```

在microservice-consumer-movie中的添加Feign依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
    <version>1.2.4.RELEASE</version>
</dependency>
```

修改ConsumerMovieApplication,添加@FeignClient注解

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class ConsumerMovieApplication {
    public static void main(String[] args){
        SpringApplication.run(ConsumerMovieApplication.class,args);
    }
}
```

为microservice-consumer-movie编写FeignClient接口

```
@FeignClient(name = "microservice-provider-user-multiple-params")
public interface UserFeignClient {
    @RequestMapping(value = "/{id}",method = RequestMethod.GET)
    public User findById(@PathVariable("id") Long id);

    @RequestMapping(value = "/user/get1",method = RequestMethod.GET)
    public User get1(@RequestParam("id") Long id,@RequestParam("username") String username);

    @RequestMapping(value = "/user/get2",method = RequestMethod.GET)
    public User get2(@RequestParam Map<String,Object> map);
}
```

修改microservice-consumer-movie中的MovieController，添加以下方法.

```
@GetMapping("/user/get1")
public User get1(User user){
    return userFeignClient.get1(user.getId(),user.getUsername());
}

@GetMapping("/user/get2")
public User get2(User user){
    HashMap<String, Object> map=new HashMap<String,Object>();
    map.put("id",user.getId());
    map.put("username",user.getUsername());
    return userFeignClient.get2(map);
}
```

接着按 发现组件-微服务 顺序运行
试着用浏览器访问查看效果

POST方法

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

在microservice-provider-user中的UserController添加post方法的映射

```
@PostMapping("/user/post")
public User post(@RequestBody User user){
    return user;
}
```

为microservice-consumer-movie中的UserFeignClient添加post接口

```
@RequestMapping(value = "/user/post",method = RequestMethod.POST)
public User post(@RequestBody User user);
```

在microservice-consumer-movie中的MovieController添加对应的方法来调用这个接口

```
@GetMapping("/user/post")
public User post(User user) {
    return this.userFeignClient.post(user);
}
```

按顺序运行程序，并通过浏览器访问相关的URL查看效果

Zuul构建微服务网关

外部客户端可能需要调用很多个微服务接口才能完成一个业务需求，这样就会导致调用频繁，客户端依赖太多，需要多方认证，也使得客户端变得很复杂。

因此需要在客户端和服务群中增加一个中间服务器，该服务器负责转发调用，验证，它甚至可以屏蔽某些服务的调用，只负责部分端口的调用。

本文档只展示一个Zuul的微服务网关，用于代理Eureka Server上的所有微服务

编写Zuul网关

创建一个Maven工程,编写pom文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>HJX-GROUP</groupId>
    <artifactId>microservice-gateway-zuul-simple</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <!-- 引入spring boot的依赖 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.3.RELEASE</version>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-zuul</artifactId>
```


© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

```
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
    </dependencies>

    <!-- 引入spring cloud的依赖 -->
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Camden.SR4</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <!-- 添加spring-boot的maven插件 -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

编写application.yml

```
server:
  port: 8040
spring:
  application:
    name: microservice-gateway-zuul
eureka:
  client:
    service-url:
      defaultZone:  http://localhost:8761/eureka/,http://localhost:8762/eureka/
```

编写ZuulApplication.java

```
@SpringBootApplication
@EnableZuulProxy
public class ZuulApplication {
    public static void main(String[] args){
        SpringApplication.run(ZuulApplication.class,args);
    }
}
```

按顺序运行程序 发现组件-微服务-微服务网关
访问 http://localhost:8040/微服务ApplicationName/** 来调用微服务。这样可以把调用转发到相应的微服务中。

Zuul集群

Zuul客户端注册到Eureka Server上

当Zuul客户端也注册到了Eureka Server中时，可以运行多个Zuul节点，客户端通过Ribbon负载均衡地访问这些节点

Zuul客户端未注册到Eureka Server上

如果Zuul客户端没有注册到Eureka Server上（这也是生成中常见的现象），只需要在客户端上添加额外的负载均衡器

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪

(例如Nginx,HAProxy,F5等等)来实现Zuul的高可用。

Docker上部署微服务

Docker在Windows中运行得并不是很好，所以我还没有能够成功地将微服务部署到Docker容器中运行

其他

该文档中阐述了如何编写微服务和部署微服务，但是还没有涉及到很多其中的内容，这些内容包括Hystrix的容错处理，以及微服务的安全认证及其组件中的安全认证机制。还有一些组件的配置如Feign配置的修改，Ribbon配置的修改，微服务配置的统一管理。本文档也没有涉及微服务的跟踪。

github: https://github.com/lkysyzxz/SpringBoot/tree/master/演示

Author:寒江雪
Date:2017.9.27

© Copyright Wednesday, Sep 27, 2017, 1:34 PM by 寒江雪