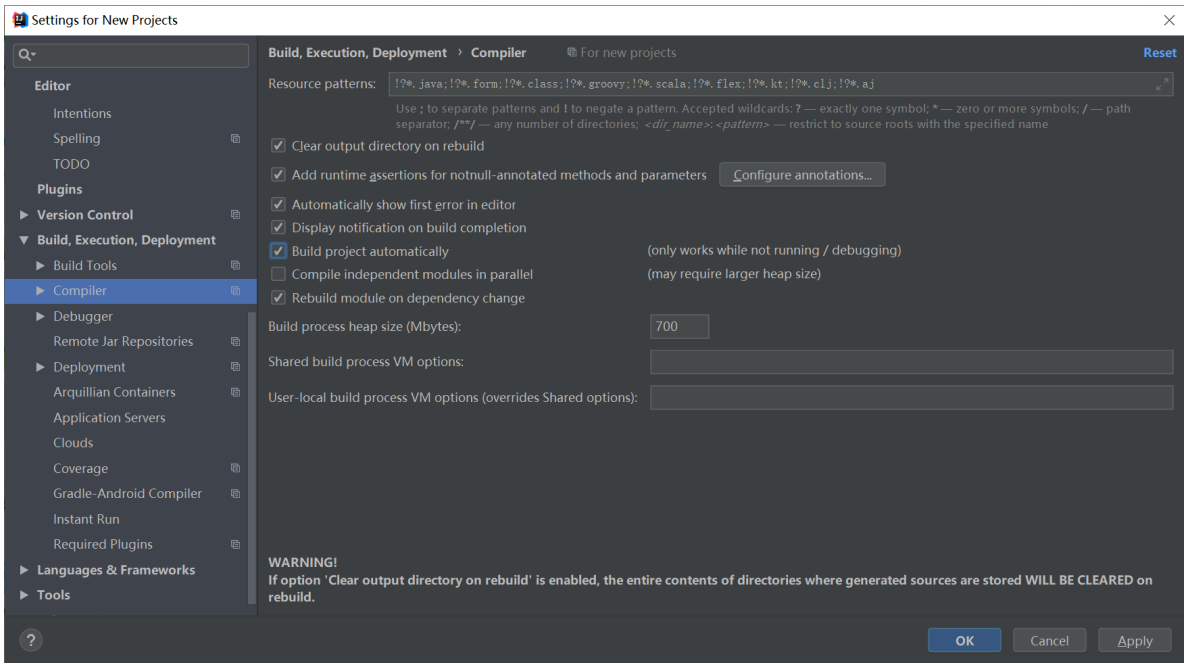
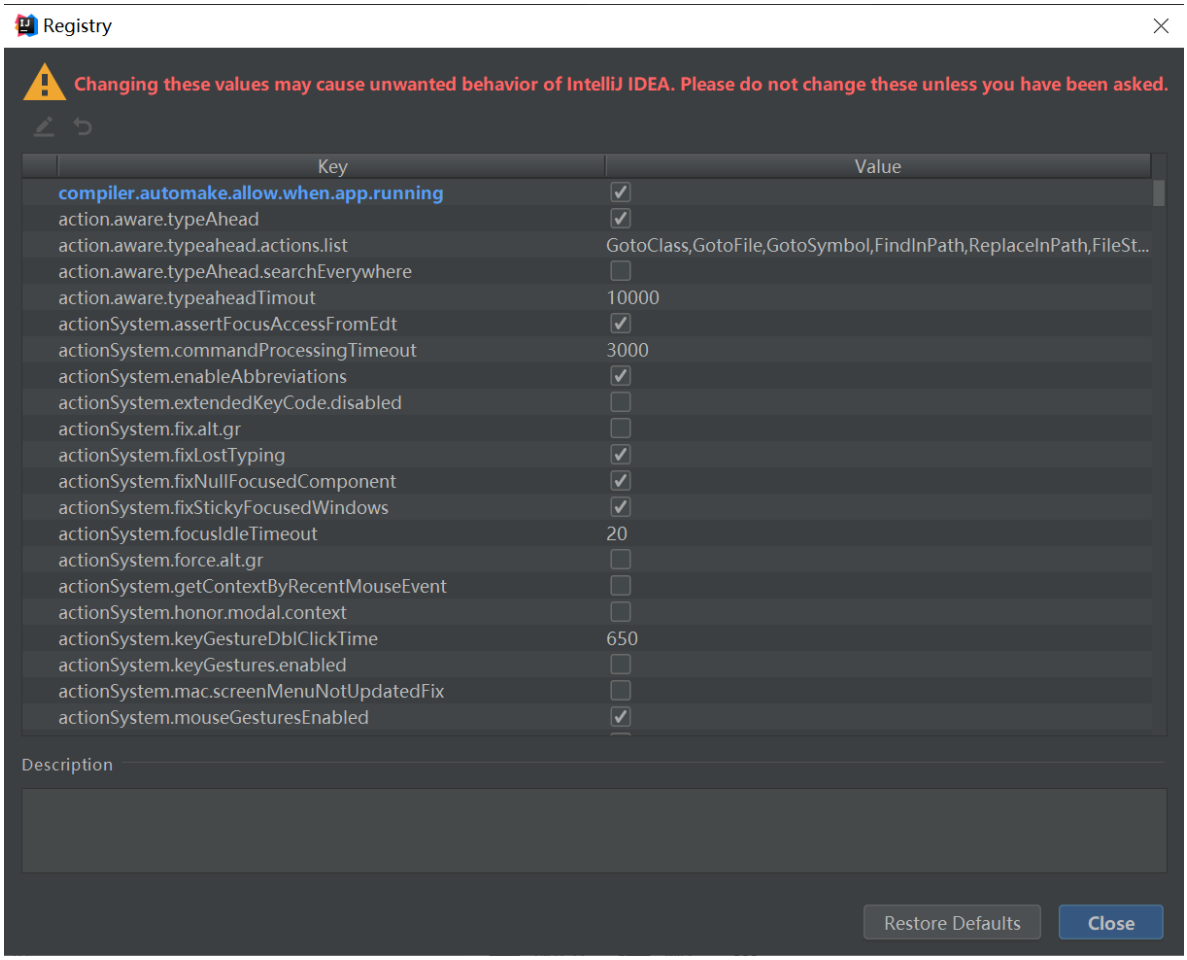
Spring Cloud

附录

热部署(IDEA)



> Ctrl+Shift+A Registry



```
 <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-devtools</artifactId>
           <optional>true</optional>
</dependency>
```

```
  <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
      <fork>true</fork>
      </configuration>
  </plugin>
```

配置好之后，还是需要重启一下IDEA的，然后修改java文件还是配置文件都会自动重启的，注意：Chrome浏览器可能有缓存，可以去按F12关了

https://www.jianshu.com/p/e40d111c7bfc?utm_source=oschina-app

Spring Cloud Eureka

搭建服务注册中心

maven依赖

```
<?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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>club.myrimer</groupId>
    <artifactId>server-provider</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>server-provider</name>
    <packaging>jar</packaging>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
    </properties>

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

```
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</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>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.2</version>
                <configuration>
                    <skipTests>true</skipTests>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

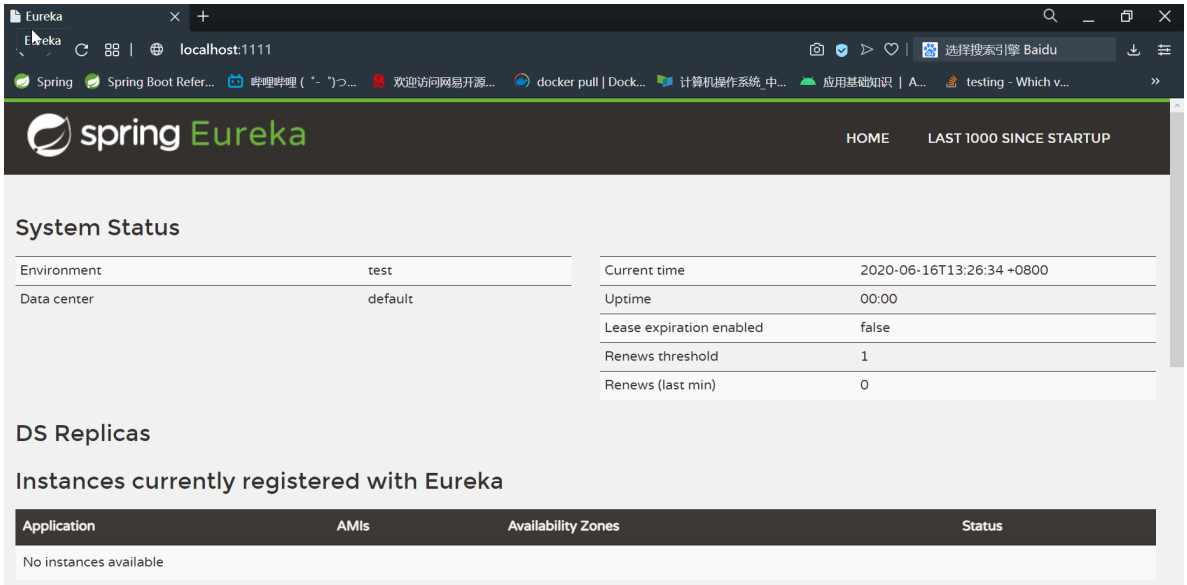
启动服务注册中心

```java
//启动服务注册中心
@EnableEurekaServer
@SpringBootApplication
public class EurekaServerApplication {

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

}
```

禁用客户端注册行为

服务注册中心会将自己作为客户端尝试注册它自己

```properties
spring.application.name=eureka-server
server.port=1111

eureka.instance.hostname=localhost

# 关闭保护机制
#eureka.server.enable-self-preservation=false

#不注册自己
eureka.client.register-with-eureka=false

#不检索服务
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/

logging.file=${spring.application.name}.log
```



注册服务提供者

maven依赖

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>club.myrimer</groupId>
    <artifactId>server-provider</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>server-provider</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</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>
        </plugins>
    </build>

</project>
```

注入DiscoverClient对象

```
@RestController
public class HelloController {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Autowired
    private EurekaClient discoveryClient;

    @GetMapping("/hello")
    public String index() {
        InstanceInfo instance =
discoveryClient.getNextServerFromEureka("STORES", false);
        logger.info("/hello,host:" + instance.getHostName() + ", service_id:" +
instance.getId());
        return "Hello Spring Cloud!";
    }
}
```

激活Eureka中的DiscoveryClient实现

```
@EnableDiscoveryClient
@SpringBootApplication
public class ServerProviderApplication {

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

}
```

指定服务注册中心的地址

```
spring.application.name=server-provider
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

```
Tomcat started on port(s): 8080 (http) with context path ''
Updating port to 8080
DiscoveryClient_SERVER-PROVIDER/localhost:server-provider - registration status: 204
```

```
Registered instance SERVER-PROVIDER/localhost:server-provider with status UP (replication=false)
```

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SERVER-PROVIDER	n/a (1)	(1)	UP (1) - localhost:server-provider

高可用注册中心

构建双节点的服务注册中心集群

- 创建application-peer1.properties

```
spring.application.name=eureka-server
server.port=1111
eureka.instance.hostname=peer1

eureka.client.serviceUrl.defaultZone=http://peer2:1112/eureka/
```

- 创建application-peer2.properties

```
spring.application.name=eureka-server
server.port=1112
eureka.instance.hostname=peer2

eureka.client.serviceUrl.defaultZone=http://peer1:1111/eureka/
```

- 修改/etc/hosts

  或者使用IP地址形式，需要增加配置参数eureka.instance.prefer-ip-address=true

```
127.0.0.1 peer1
127.0.0.1 peer2
```

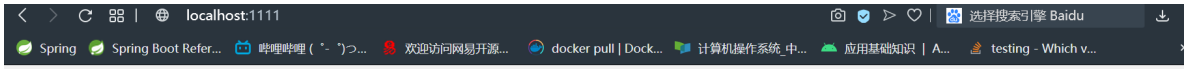
- 打包后启动

  > 或者IDEA修改active profile启动

```
java -jar eureka-server-0.0.1-SNAPSHOT.jar --spring.profiles.active=peer1
java -jar eureka-server-0.0.1-SNAPSHOT.jar --spring.profiles.active=peer2
```

- 注册服务提供者

```
spring.application.name=server-provider
eureka.client.serviceUrl.defaultZone=http://peer1:1111/eureka/,http://peer2:
1112/eureka/
```



System Status

Environment	test
Data center	default

Current time	2020-06-18T15:09:24 +0800
Uptime	00:02
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	3

DS Replicas

peer2

Instances currently registered with Eureka

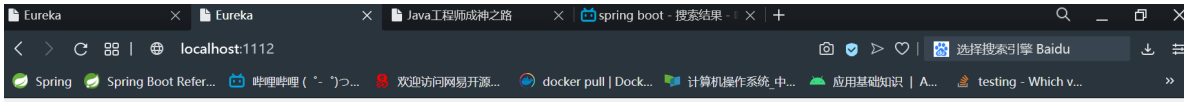
Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - Dell921421:eureka-server:1112 , Dell921421:eureka-server:1111
SERVER-PROVIDER	n/a (1)	(1)	UP (1) - Dell921421:server-provider

General Info

General Info

Name	Value
total-avail-memory	686mb
environment	test
num-of-cpus	4
current-memory-usage	502mb (73%)
server-uptime	00:02
registered-replicas	http://peer2:1112/eureka/
unavailable-replicas	
available-replicas	http://peer2:1112/eureka/,

Instance Info



System Status

Environment	test
Data center	default

Current time	2020-06-18T15:09:30 +0800
Uptime	00:01
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	0

DS Replicas

peer1

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (2)	(2)	UP (2) - Dell921421:eureka-server:1112 , Dell921421:eureka-server:1111
SERVER-PROVIDER	n/a (1)	(1)	UP (1) - Dell921421:server-provider

General Info

Name	Value
total-avail-memory	713mb
environment	test
num-of-cpus	4
current-memory-usage	488mb (68%)
server-uptime	00:01
registered-replicas	http://peer1:1111/eureka/
unavailable-replicas	
available-replicas	http://peer1:1111/eureka/,

Instance Info

服务发现与注册

启动服务注册中心

```
spring.application.name=eureka-server
server.port=1111

eureka.instance.hostname=localhost

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server
.port}/eureka/

logging.file.name=${spring.application.name}.log
```

启动服务提供者

> 或者IDEA修改override parameters启动

```
java -jar server-provider-0.0.1-SNAPSHOT.jar --server.port=8081
java -jar server-provider-0.0.1-SNAPSHOT.jar --server.port=8082
```

启动服务消费者

```
<?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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>club.myprimer</groupId>
    <artifactId>ribbon-consumer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>ribbon-consumer</name>
    <description>Demo project for Spring Boot</description>

    <properties>
```

```
        <java.version>1.8</java.version>
        <spring-cloud.version>Hoxton.SR5</spring-cloud.version>
    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</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>
        </plugins>
    </build>

</project>
```

```
//将该应用注册为Eureka客户端应用
@EnableDiscoveryClient
@SpringBootApplication
public class RibbonConsumerApplication {

    //开启客户端负载均衡
    @Bean
```

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


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

}
```

创建调用服务提供者接口的Controller类

```
@RestController
public class ConsumerController {

    @Autowired
    RestTemplate restTemplate;

    @GetMapping("/ribbon-consumer")
    public String helloConsumer() {
        //访问的地址是服务器名
        return restTemplate.getForEntity("http://SERVER-PROVIDER/hello",
String.class).getBody();
    }

}
```

配置Eureka服务注册中心地址及消费者端口

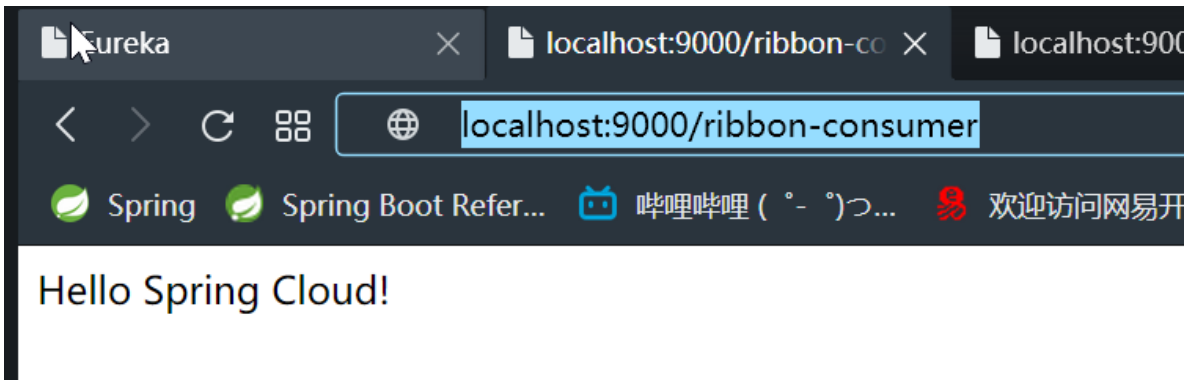
```
spring.application.name=ribbon-consumer
server.port=9000

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
RIBBON-CONSUMER	n/a (1)	(1)	UP (1) - localhost:ribbon-consumer:9000
SERVER-PROVIDER	n/a (2)	(2)	UP (2) - localhost:server-provider:8082 , localhost:server-provider:8081

General Info



服务治理体系

启动注册的参数

```
#默认启动
eureka.client.register-with-eureka=true
```

服务续约的参数

```
#定义服务续约任务的调用间隔时间,默认30s
eureka.instance.lease-renewal-interval-in-seconds=30
#定义服务失效的时间，默认90s
eureka.instance.lease-expiration-duration-in-seconds=90
```

获取服务的参数

```
#默认获取
eureka.client.fetch-registry=true
#修改缓存清单的更新时间,默认30s
eureka.client.registry-fetch-interval-seconds=30
```

自我保护的参数

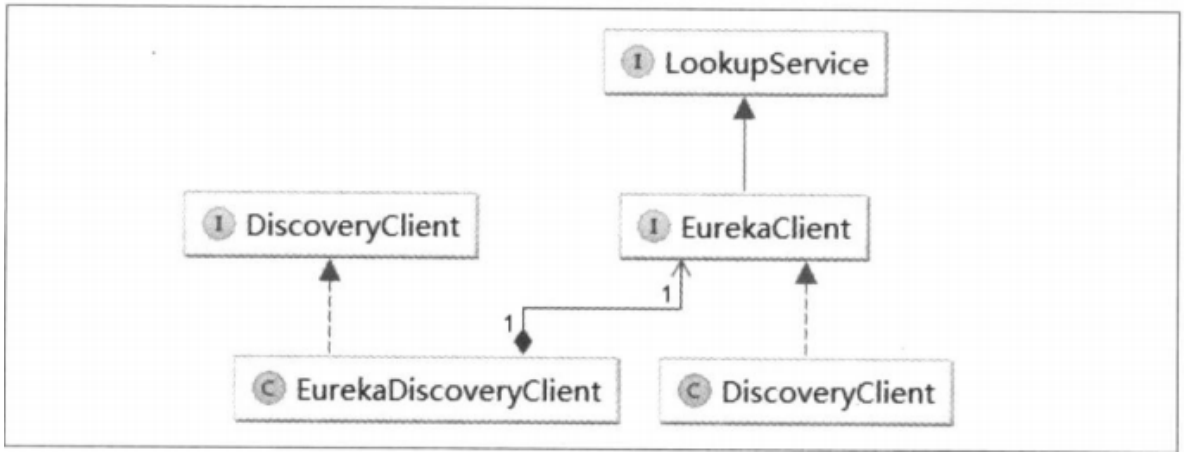
```
    EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT.
RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST
TO BE SAFE.
```

```
#默认开启自我保护机制
eureka.server.enable-self-preservation=true
```

源码分析

客户端主动通信行为的源码线索

- 在应用主类中配置了@EnableDiscoveryClient注解
- 在application.properties中用eureka.client.serviceUrl.defaultZone参数指定了服务注册中心的位置



- 左DiscoveryClient接口定义了发现服务的抽象方法(适用于不同服务治理框架)
- LookupService接口定义了针对Eureka的发现服务的抽象方法

配置详解

- 服务注册相关的配置信息，包括注册中心的地址,服务获取的间隔时间可用区域

- 服务实例相关的配置信息，包括服务实例的名称,IP地址，端口号，健康服务路径

跨平台支持

- Eureka使用Jersey和XStream配合JSON作为Server与Client之间的通信协议
- 可选择实现自己的协议来替代默认协议

Spring Cloud Ribbon

RestTemplate详解

GET请求

```
ResponseEntity<String> responseEntity = restTemplate.getForEntity("http://USER-
SERVICE/user?name={1}", String.class, "didi");
        String body = responseEntity.getBody();
```

```
Map<String,String> params = new HashMap<>();
        params.put("name","Tom" );
        ResponseEntity<String> responseEntity =
restTemplate.getForEntity("http://USER-SERVICE/user?name={name}", String.class,
params);
        String body = responseEntity.getBody();
```

```
UriComponents uriComponents = UriComponentsBuilder.fromUriString("http://USER-
SERVICE/user?name={name}")
                .build()
                .expand("Jim")
                .encode();
        URI uri = uriComponents.toUri();
        ResponseEntity<String> responseEntity = restTemplate.getForEntity(uri,
String.class);
        String body = responseEntity.getBody();
```

```
String s = restTemplate.getForObject(uri,String.class );
User user = restTemplate.getForObject(uri,User.class ) ;
```

POST请求

```
User user = new User();
        ResponseEntity<String> responseEntity =
restTemplate.postForEntity("http://USER-SERVICE/user",user, String.class);
        String body = responseEntity.getBody();
```

```
String s = restTemplate.postForObject("http://USER-SERVICE/user",
user,String.class );
```

PUT请求

```
Long id = 10001L;
User user = new User();
restTemplate.put("http://USER-SERVICE/user/{1}",user,id );
```

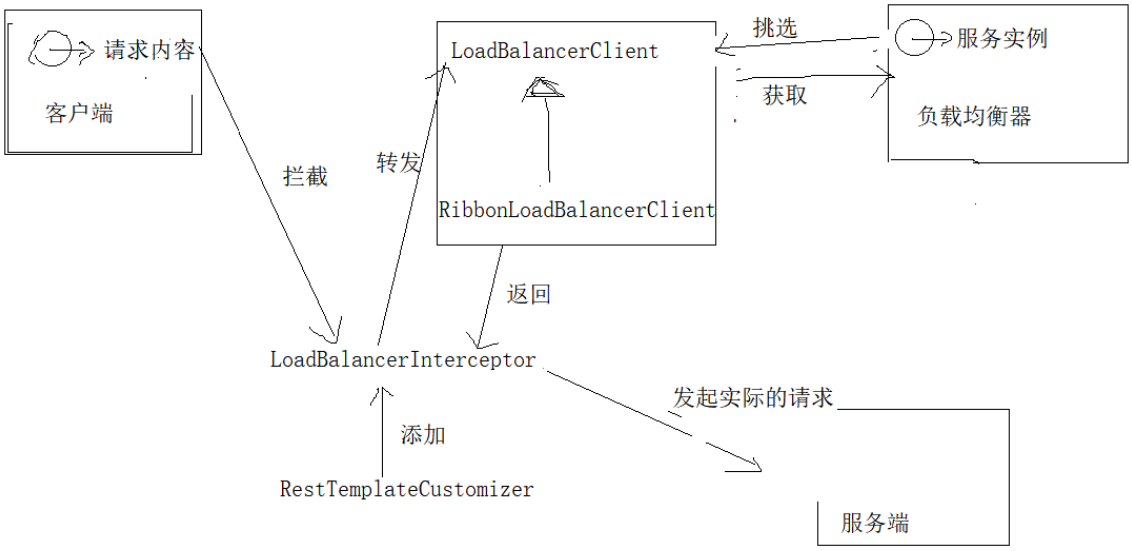
DELETE请求·

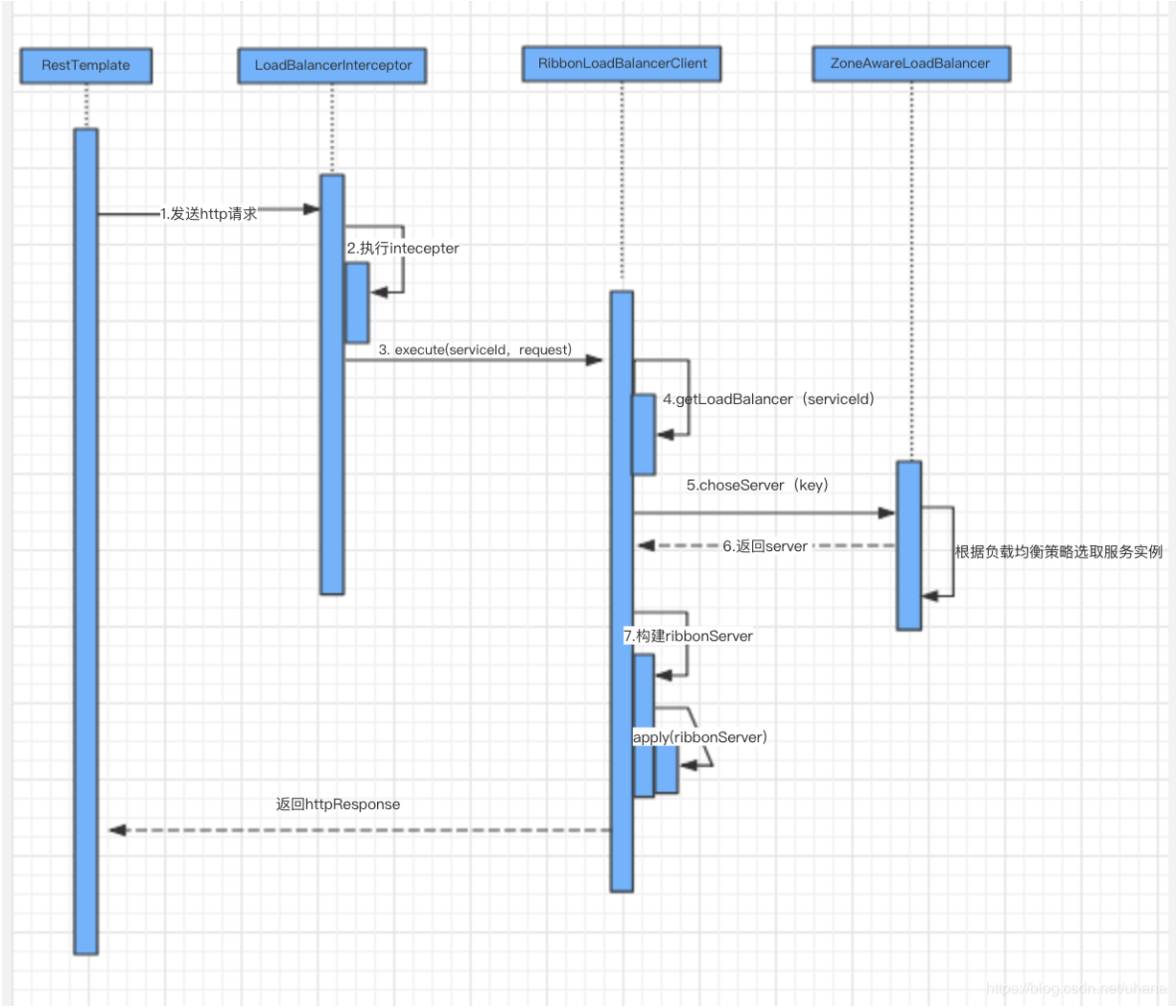
```
Long id = 10001L;
restTemplate.delete("http://USER-SERVICE/user/{1}",id);
```

源码分析

客户端负载均衡的源码线索

LoadBalancerInterceptor对客户端发起的请求进行拦截，获取负载均衡策略分配到的服务实例对象Server之后，将其内容包装成RibbonServer对象，使用该对象再回调LoadBalancerInterceptor请求拦截器中的LoadBalancerRequest对象的apply方法，向一个实际的具体服务实例发送请求，从而实现以服务名host的URI请求到host：port形式的实际访问地址的转换





Spring Cloud Hystrix

实现断路器

maven依赖

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

开启断路器功能

```
@EnableCircuitBreaker
@EnableDiscoveryClient
@SpringBootApplication
public class RibbonConsumerApplication {

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


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

```
}
```

```
//另一种方式
@SpringCloudApplication
@SpringBootApplication
public class RibbonConsumerApplication {

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


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

}
```

```
//@SpringCloudApplication源码
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker
public @interface SpringCloudApplication {
}
```

改造服务消费方式

```
@Service
public class HelloService {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Autowired
    RestTemplate restTemplate;

    //指定服务降级的实现方法
    @HystrixCommand(fallbackMethod = "helloFallback")
    public String helloService() {
        long start = System.currentTimeMillis();
        String result = restTemplate.getForEntity("http://SERVER-
PROVIDER/hello", String.class).getBody();
        long end = System.currentTimeMillis();

        logger.info("Spend time : " + (end - start) );
        return result;
    }

    public String helloFallback() {
        return "error";
```

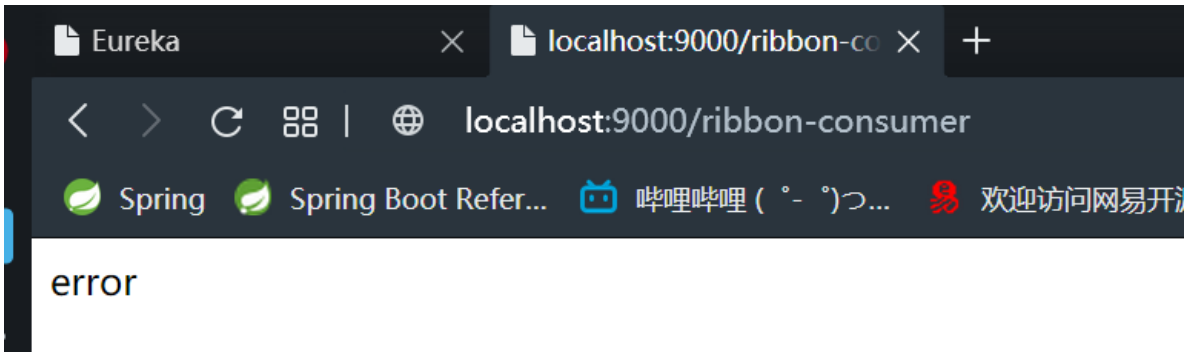
```
    }

}
```

注入服务消费方式的实例

```
@RestController
public class ConsumerController {

    @Autowired
    HelloService helloService;

    @GetMapping("/ribbon-consumer")
    public String helloConsumer() {
        return helloService.helloService();
    }

}
```

模拟服务阻塞

```
//Hystrix的默认超时时间为2000ms
@RestController
public class HelloController {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @GetMapping("/hello")
    public String index() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(3000);
        return "Hello Spring Cloud!";
    }
}
```



创建请求命令

继承的方式实现(HystrixCommand)

```
public class UserCommand extends HystrixCommand<User> {

    private RestTemplate restTemplate;
    private Long id;

    public UserCommand(Setter setter, RestTemplate restTemplate, Long id) {
        super(setter);
        this.restTemplate = restTemplate;
```

```
        this.id = id;
    }

    //封装具体的依赖服务逻辑
    @Override
    protected User run() {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
User.class, id);
    }
}

public static void main(String[] args) {

        RestTemplate restTemplate = new RestTemplate();
    //同步执行:
        User u = new UserCommand(null,restTemplate, 1L).execute();
    //异步执行:
        Future<User> futureUser = new UserCommand(null, restTemplate,
1L).queue();
    //响应式执行方式
        //Hot Observable
        Observable<User> hot = new UserCommand(null, restTemplate,
1L).observe();
        //Cold Observable
        Observable<User> cold = new UserCommand(null, restTemplate,
1L).toObservable();

    }
```

继承的方式实现(HystrixObservableCommand)

> HystrixCommand返回的Observable只能发射一次数据 HystrixObservableCommand返回的Observable可以发射多次数据

```
public class UserObservableCommand  extends HystrixObservableCommand<User> {

    private RestTemplate restTemplate;
    private Long id;

    public UserObservableCommand(Setter setter, RestTemplate restTemplate, Long
id) {
        super(setter);
        this.restTemplate = restTemplate;
        this.id = id;
    }

    @Override
    protected Observable<User> construct() {
        return Observable.unsafeCreate(new Observable.OnSubscribe<User>() {
            @Override
            public void call(Subscriber<? super User> subscriber) {
                try {
                    if (!subscriber.isUnsubscribed()) {
                        User user = restTemplate.getForObject("http://USER-
SERVICE/users/{1}", User.class, id);
                        subscriber.onNext(user);
                        subscriber.onCompleted();
```

```
                    }
                } catch (Exception e) {

                }
            }
        });
    }


}
```

注解的方式实现

```
public class UserService {

    @Autowired
    private RestTemplate restTemplate;

    //同步执行
    @HystrixCommand
    public User getUserById(Long id) {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
User.class,id);
    }

    //异步执行
    @HystrixCommand
    public Future<User> getUserByIdAsync(final String id) {
        return new AsyncResult<User>() {
            @Override
            public User invoke() {
                return restTemplate.getForObject("http://USER-
SERVICE/users/{1}", User.class,id);
            }
        };
    }

    //响应式执行
    //Hot Observable
    //@HystrixCommand(observableExecutionMode = ObservableExecutionMode.EAGER)
    //Cold Observable
    @HystrixCommand(observableExecutionMode = ObservableExecutionMode.LAZY)
    public Observable<User> getUserById(final String id) {
        return Observable.unsafeCreate(new Observable.OnSubscribe<User>() {
            @Override
            public void call(Subscriber<? super User> subscriber) {
                try {
                    if (!subscriber.isUnsubscribed()) {
                        User user = restTemplate.getForObject("http://USER-
SERVICE/users/{1}", User.class, id);
                        subscriber.onNext(user);
                        subscriber.onCompleted();
                    }
                } catch (Exception e) {

                }
            }
```

```
        });
    }
}
```

定义服务降级

```
//HystrixCommand的重载方法
@Override
protected User getFallback() {
    return new User();
}
```

```
//HystrixObservableCommand的重载方法
@Override
protected Observable<User> resumeWithFallback() {
    return super.resumeWithFallback();
}
```

```
//注解方式
@HystrixCommand(fallbackMethod = "helloFallback")
    public String helloService() {
        return "";
    }

    public String helloFallback() {
        return "error";
    }
```

异常处理

异常传播

```
//忽略指定异常类型
    @HystrixCommand(ignoreExceptions = HystrixBadRequestException.class)
    public User getUserById(Long id) {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
User.class,id);
    }
```

异常获取

- `getExecutionException();//方法1`
- `xxx(Throwable e)`

命令名称，分组以及线程池划分

Setter静态类设置命令名称

```
//Hystrix命令默认的线程划分是根据命令分组实现的
//默认同组名使用相同线程池
public UserCommand(RestTemplate restTemplate, Long id) {

 super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("GroupName"))
            .andCommandKey(HystrixCommandKey.Factory.asKey("CommandName")));
        this.restTemplate = restTemplate;
        this.id = id;
    }
```

```
//特别指定线程池
super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("GroupName"))
            .andCommandKey(HystrixCommandKey.Factory.asKey("CommandName"))

.andThreadPoolKey(HystrixThreadPoolKey.Factory.asKey("ThreadPoolName")));
```

```
//注解方式设置
@HystrixCommand(commandKey = "getUserByIdAsync",groupKey =
"UserGroup",threadPoolKey = "getUserByIdAsyncThread")
    public Future<User> getUserByIdAsync(final String id) {
        return new AsyncResult<User>() {
            ...
            }
        };
    }
```

请求缓存

开启请求缓存功能

```
//重载getCacheKey方法
@Override
    protected String getCacheKey() {
        return String.valueOf(id);
    }
```

清理失效缓存功能

```
 public static void flushCache(long id) {
        HystrixCommandKey commandKey =
HystrixCommandKey.Factory.asKey("CommandName");
        HystrixRequestCache.getInstance(commandKey,
HystrixConcurrencyStrategyDefault.getInstance()).clear(String.valueOf(id));
    }
```

使用注解实现请求缓存

```
@CacheResult(cacheKeyMethod = "getUserByIdCacheKey")
    @HystrixCommand(ignoreExceptions = HystrixBadRequestException.class)
    public User getUserById(Long id) {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
User.class,id);
    }

    private Long getUserByIdCacheKey(Long id) {
        return id;
    }
```

```
@CacheResult
    @HystrixCommand(ignoreExceptions = HystrixBadRequestException.class)
    public User getUserById(@CacheKey("id") Long id) {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
User.class,id);
    }
//@CacheKey("id") User user
```

```
 @CacheRemove(commandKey = "getUserById")
    @HystrixCommand
    public void update(@CacheKey("id") User user) {
        return restTemplate.postForObject("http://USER-SERVICE/users",
user,User.class );
    }
```

请求合并

继承方式实现

```
@Service
public class UserService {

    @Autowired
    private RestTemplate restTemplate;

    public User find(Long id) {
        return restTemplate.getForObject("http://USER-SERVICE/users/{1}",
                User.class,id );

    }

    @SuppressWarnings("unchecked")
    public List<User> findAll(List<Long> ids) {
        return restTemplate.getForObject("http://USER-SERVICE/users?ids=
{1}",List.class, StringUtils.join(ids,","));
    }
}
```

```
public class UserBatchCommand extends HystrixCommand<List<User>> {

    UserService userService;
```

```
    List<Long> userIds;

    public UserBatchCommand(UserService userService, List<Long> userIds) {

 super(Setter.withGroupKey(HystrixCommandGroupKey.Factory.asKey("userServiceComm
and")));
        this.userService = userService;
        this.userIds = userIds;
    }

    @Override
    protected List<User> run() throws Exception {
        return userService.findAll(userIds);
    }
}
```

```
public class UserCollapseCommand extends HystrixCollapser<List<User>,User,Long>
{

    private UserService userService;
    private Long userId;

    public UserCollapseCommand(UserService userService, Long userId) {

 super(Setter.withCollapserKey(HystrixCollapserKey.Factory.asKey("userCollapseCo
mmand"))
            .andCollapserPropertiesDefaults(HystrixCollapserProperties.Setter().
                    withTimerDelayInMilliseconds(100)));
        this.userService = userService;
        this.userId = userId;
    }

    //获取请求参数类型
    @Override
    public Long getRequestArgument() {
        return userId;
    }

    //合并请求产生批量命令
    @Override
    protected HystrixCommand<List<User>>
createCommand(Collection<CollapsedRequest<User, Long>> collection) {
        List<Long> userIds = new ArrayList<>(collection.size());
        for (CollapsedRequest<User,Long> collapsedRequest : collection) {
            userIds.add(collapsedRequest.getArgument());
        }
        return new UserBatchCommand(userService,userIds );
    }

    //批量命令结果返回后的处理，需要实现将批量结果拆分并传
    // 递给合并前的各个原子请求命令的逻辑
    @Override
    protected void mapResponseToRequests(List<User> users,
Collection<CollapsedRequest<User, Long>> collection) {
        int count = 0;
        for (CollapsedRequest<User,Long> collapsedRequest : collection) {
            User user = users.get(count++);
```

```
            collapsedRequest.setResponse(user);
        }
    }
}
```

注解方式实现

```
 @HystrixCollapser(batchMethod = "findAll",collapserProperties
            = {@HystrixProperty(name = "timerDelayInMilliseconds",value =
"100")})
    public User find(Long id) {
        return null;
    }


    @SuppressWarnings("unchecked")
    @HystrixCommand
    public List<User> findAll(List<Long> ids) {
        return restTemplate.getForObject("http://USER-SERVICE/users?ids=
{1}",List.class, StringUtils.join(ids,","));
    }
```

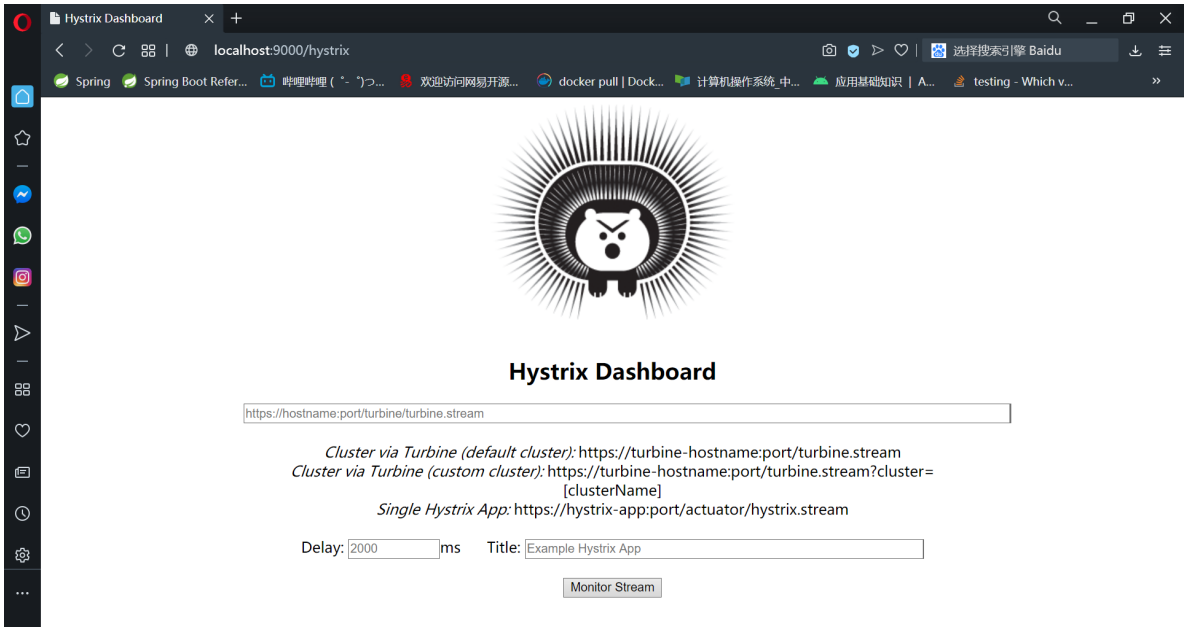
仪表盘

添加依赖

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

启动仪表盘功能

```
@SpringCloudApplication
//启动仪表盘功能
@EnableHystrixDashboard
@SpringBootApplication
public class RibbonConsumerApplication {
    ...
}
```



集群监控

添加依赖

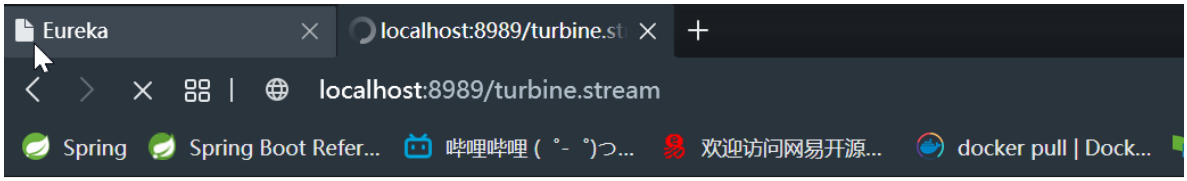
```
 <dependency>
       <groupId>org.springframework.cloud</groupId>
       <artifactId>spring-cloud-starter-netflix-turbine</artifactId>
</dependency>
```

开启集群监控功能

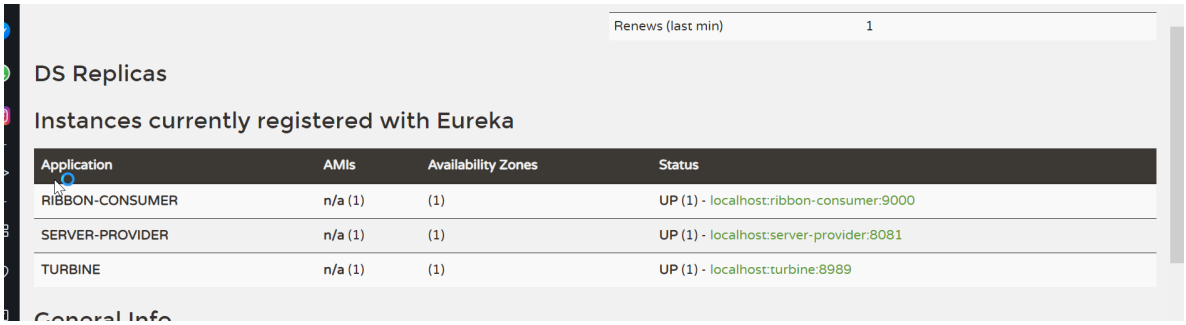
```
@EnableTurbine
@EnableDiscoveryClient
@SpringBootApplication
public class TurbineApplication {

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

}
```

配置集群监控属性

```
spring.application.name=turbine

server.port=8989

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/

turbine.app-config=RIBBON-CONSUMER
turbine.cluster-name-expression="default"
turbine.combine-host-port=true
```




```
: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054554038}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054557040}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054560042}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054563044}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054566046}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054569048}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054572050}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054575052}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054578084}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054582085}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054585087}

: ping
data: {"reportingHostsLast10Seconds":0,"name":"meta","type":"meta","timestamp":1593054588088}

: ping
```



与消息代理结合

使用RabbitMQ收集监控消息，Turbine服务再从RabbitMQ中异步获取监控消息

最后监控信息聚合到Hystrix Dashboard

Spring Cloud Feign

实现服务接口的调用

添加依赖

```
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

开启Spring Cloud Feign的支持功能

```
@EnableFeignClients
@EnableDiscoveryClient
@SpringBootApplication
public class FeignConsumerApplication {

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

}
```

指定服务名并绑定服务

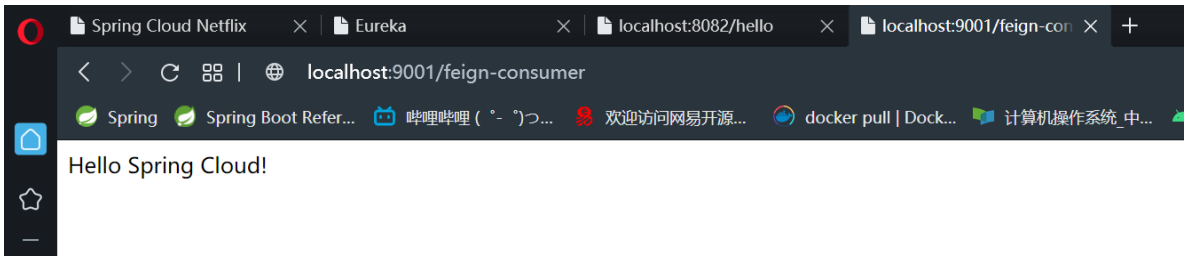
```
@FeignClient("SERVER-PROVIDER")
public interface IHelloService {

    @RequestMapping("/hello")
    String hello();

}
```

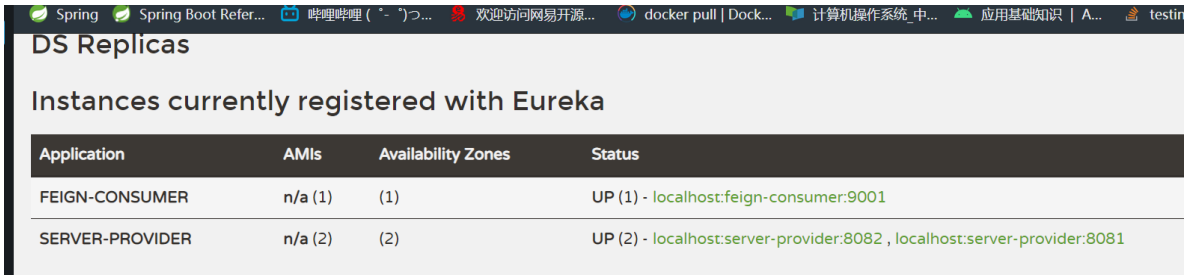
实现对Feign客户端的调用

```
@RestController
public class ConsumerController {

    @Autowired
    IHelloService helloService;

    @GetMapping(value = "/feign-consumer")
    public String helloConsumer() {
        return helloService.hello();
    }

}
```

指定服务注册中心

```
spring.application.name=feign-consumer
server.port=9001

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```





参数绑定

定义实体类

```
public class User {

    private String name;
    private Integer age;

    public User() {
    }

    public User(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "name=" + name + ", age=" +age;
```

```
    }

}
```

增加带有参数的请求

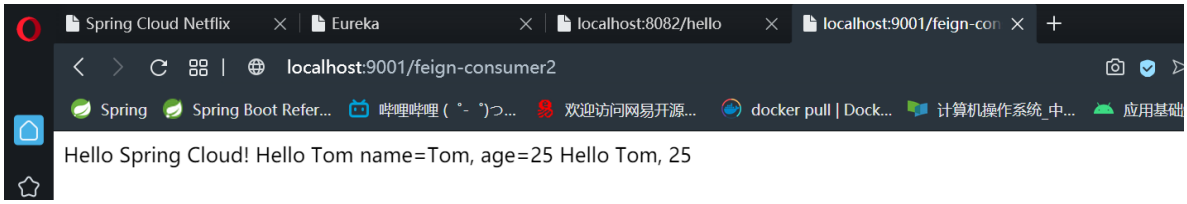
```
    @GetMapping("/hello1")
    public String hello(@RequestParam String name) {
        logger.info("hello1");
        return "Hello " + name;
    }

    @GetMapping("/hello2")
    public User hello(@RequestHeader String name, @RequestHeader Integer age) {
        logger.info("hello2");
        return new User(name, age);
    }

    @PostMapping("/hello3")
    public String hello(@RequestBody User user) {
        logger.info("hello3");
        return "Hello "+ user.getName() + ", " + user.getAge();
    }
```

增加新增接口的绑定声明

```
    @GetMapping("/hello1")
    String hello(@RequestParam("name") String name) ;

    @GetMapping("/hello2")
    User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer
age);

    @PostMapping("/hello3")
    String hello(@RequestBody User user);
```

对新增接口进行调用

```
    @GetMapping(value = "/feign-consumer2")
    public String helloConsumer2() {
        StringBuilder sb = new StringBuilder();
        sb.append(helloService.hello()).append("\n");
        sb.append(helloService.hello("Tom")).append("\n");
        sb.append(helloService.hello("Tom",25)).append("\n");
        sb.append(helloService.hello(new User("Tom",25))).append("\n");
        return sb.toString();
    }
```



```
: Resolving eureka endpoints via configuration
: hello1
: hello3
: Resolving eureka endpoints via configuration
```

```
: hello2
: Resolving eureka endpoints via configuration
```

继承特性

复用DTO与接口定义

- 添加DTO类
- 添加接口

```
@RequestMapping("/refactor")
public interface IHelloService {

    @RequestMapping(value = "/hello4", method = RequestMethod.GET)
    String hello(@RequestParam("name") String name) ;

    @RequestMapping(value = "/hello5", method = RequestMethod.GET)
    User hello(@RequestHeader("name") String name, @RequestHeader("age")
Integer age);

    @RequestMapping(value = "/hello6", method = RequestMethod.POST)
    String hello(@RequestBody User user);

}
```

- 添加依赖

```
<packaging>jar</packaging>

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

<build>
        <plugins>
            <!--据说是跳过单元测试？？？-->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <skip>true</skip>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
```

```
                <version>2.22.2</version>
                <configuration>
                    <skipTests>true</skipTests>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

- mvn clean install

引用复用模块

```
<dependency>
            <groupId>club.myprimer</groupId>
            <artifactId>server-provider-api</artifactId>
            <version>0.0.1-SNAPSHOT</version>
 </dependency>
```

继承并实现复用模块接口

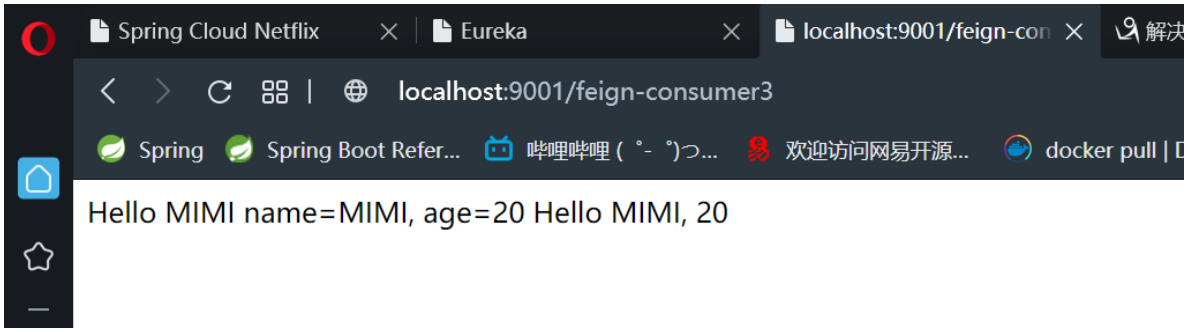
```
@RestController
public class RefactorController implements IHelloService {
    @Override
    public String hello(String name) {
        return "Hello " + name;
    }

    @Override
    public User hello(String name, Integer age) {
        return new User(name, age);
    }

    @Override
    public String hello(User user) {
        return "Hello "+ user.getName() + ", " + user.getAge();
    }
}
```

为复用模块的接口绑定服务

```
@FeignClient("SERVER-PROVIDER")
public interface IRefactorHelloService extends IHelloService {

}
```

新增接口进行调用

```
@Autowired
IRefactorHelloService refactorHelloService;

@GetMapping(value = "/feign-consumer3")
    public String helloConsumer3() {
        StringBuilder sb = new StringBuilder();
        sb.append(refactorHelloService.hello("MIMI")).append("\n");
        sb.append(refactorHelloService.hello("MIMI", 20)).append("\n");
        sb.append(refactorHelloService.hello(new
club.myprimer.serverproviderapi.dto.User("MIMI", 20))).append("\n");
        return sb.toString();
    }
```



Ribbon配置

全局配置

指定服务配置

重试机制

Hystrix配置

全局配置

禁用Hystrix

指定命令配置

服务降级配置

```
public class HelloServiceFallback implements IHelloService {
    @Override
    public String hello() {
        return "error";
    }

    @Override
    public String hello(String name) {
        return "error";
    }

    @Override
    public User hello(String name, Integer age) {
        return null;
    }

    @Override
    public String hello(User user) {
```

```
        return "error";
    }
}
```

```
@FeignClient(name = "SERVER-PROVIDER2",fallback = HelloServiceFallback.class)
public interface IHelloService {

    @GetMapping("/hello")
    String hello();


    @GetMapping("/hello1")
    String hello(@RequestParam("name") String name) ;

    @GetMapping("/hello2")
    User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer
age);

    @PostMapping("/hello3")
    String hello(@RequestBody User user);
}
```

其它配置

请求压缩

日志配置

Spring Cloud Zuul

构建网关

添加依赖

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

开启API网关服务功能

```
@EnableZuulProxy
@SpringBootApplication
public class ApiGatewayApplication {

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

}
```

配置信息

```
spring.application.name=api-gateway
server.port=5555
```

请求路由

传统路由方式

```
# routes to url
zuul.routes.api-a-url.path=/api-a-url/**
zuul.routes.api-a-url.url=http://localhost:8082/
```

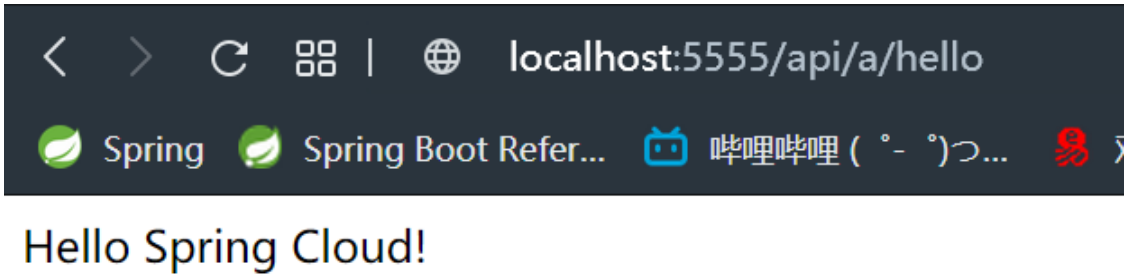
面向服务的路由

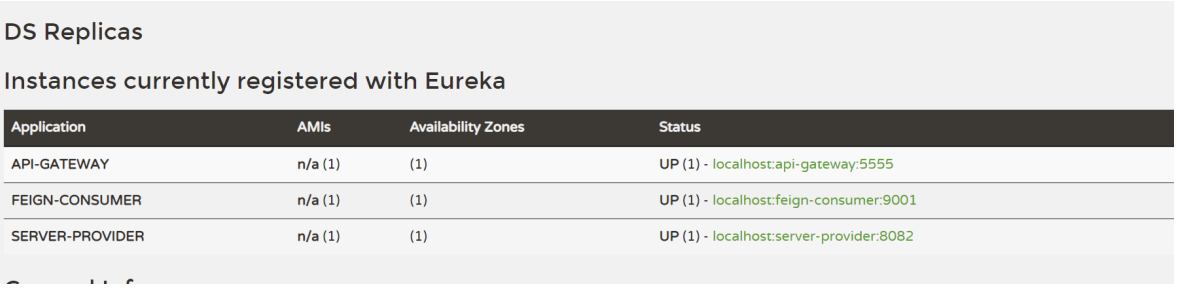
- 添加依赖

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

- 指定服务中心并配置服务路由

```
# routes to serviceId
zuul.routes.api-a.path=/api-a/**
zuul.routes.api-a.serviceId=server-provider

zuul.routes.api-b.path=/api-b/**
zuul.routes.api-b.serviceId=feign-consumer

# eureka
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```



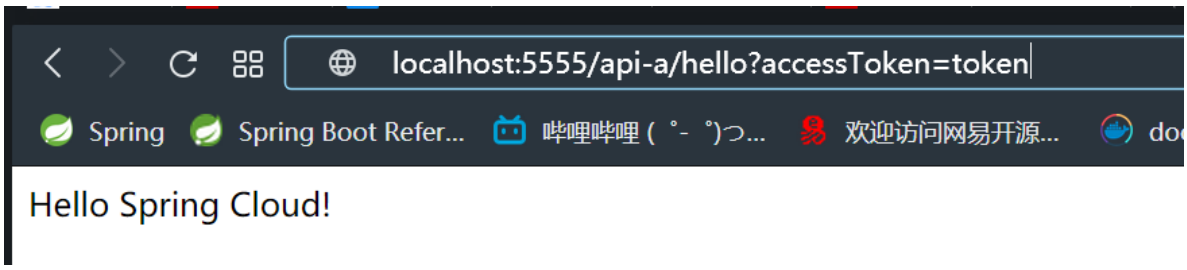


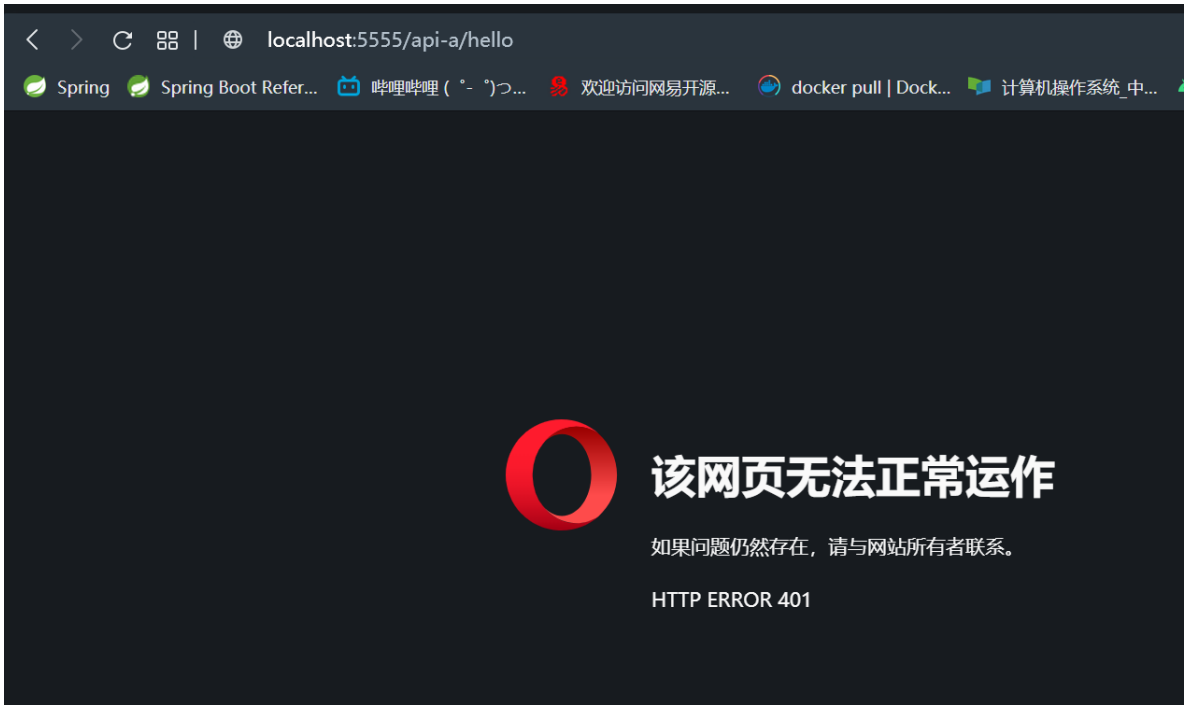
请求过滤

定义过滤器

```
@Component
public class AccessFilter extends ZuulFilter {

    private static Logger log = LoggerFactory.getLogger(AccessFilter.class);
```

```
    //pre代表在请求被路由之前执行过滤器
    @Override
    public String filterType() {
        return "pre";
    }

    //过滤器的执行顺序
    @Override
    public int filterOrder() {
        return 0;
    }

    //该过滤器是否需要被执行
    @Override
    public boolean shouldFilter() {
        return true;
    }

    //过滤器的具体逻辑
    @Override
    public Object run() {
        RequestContext ctx = RequestContext.getCurrentContext();
        HttpServletRequest request = ctx.getRequest();

        log.info("send {} request to {}", request.getMethod(),
request.getRequestURL().toString());

        Object accessToken = request.getParameter("accessToken");
        if(accessToken == null) {
            log.warn("access token is empty");
            //另zuul过滤该请求
            ctx.setSendZuulResponse(false);
            //返回错误码
            ctx.setResponseStatusCode(401);
            return null;
        }
        log.info("access token ok");
        return null;
    }
}
```





动态加载

Spring Cloud Config

构建配置中心

添加依赖

```
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

开启pring Cloud Config服务端功能

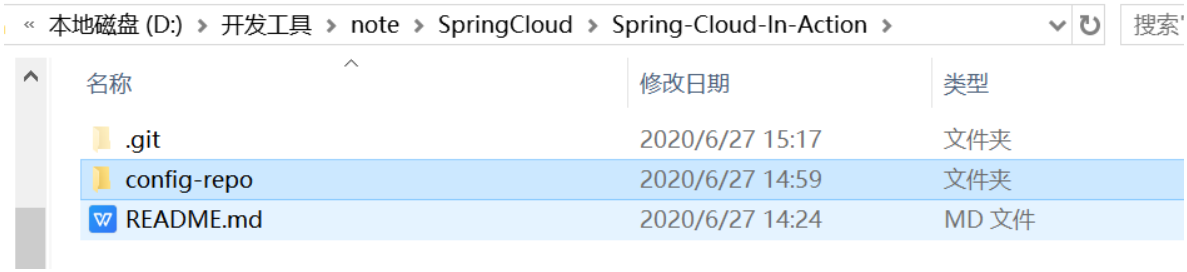
```
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {

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

}
```

添加配置服务基本信息和git信息

```
spring.application.name=config-server
server.port=7001

# git管理配置
spring.cloud.config.server.git.uri=https://gitee.com/myprimer/Spring-Cloud-Learning
spring.cloud.config.server.git.searchPaths=/config-repo
spring.cloud.config.server.git.username=myprimer
spring.cloud.config.server.git.password=yjh921421
```

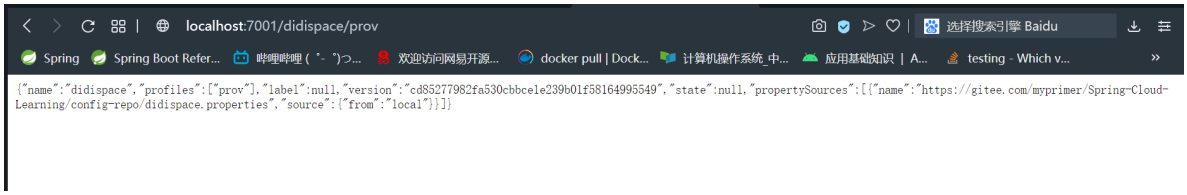
新建配置文件

```
#didispace-prod.properties
from=local-prod
```



来访问我们的配置内容了。访问配置信息的 URL 与配置文件的映射关系如下所示：

- /{application}/{profile}[/{label}]
- /{application}-{profile}.yml
- /{label}/{application}-{profile}.yml
- /{application}-{profile}.properties
- /{label}/{application}-{profile}.properties



客户端配置映射

添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

创建配置文件并指定配置中心的位置

```
spring.application.name=didispace

spring.cloud.config.profile=dev
spring.cloud.config.uri=http://localhost:7001/
spring.cloud.config.label=master

server.port=7002
```

创建返回配置中心属性的接口

```
@RefreshScope
@RestController
public class TestController {

    @Value("${from}")
```

```
    private String from;

    //另一个绑定方式
//    @Autowired
//    private Environment env;



    @RequestMapping("/from")
    public String from() {
        //return env.getProperty("from","undefine" );
        return this.from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getFrom() {
        return from;
    }

}
```

服务化配置中心

服务端配置

- 添加依赖

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

- 指定服务注册中心

```
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

- 开启注册功能

```
@EnableDiscoveryClient
```

客户端配置

- 添加依赖

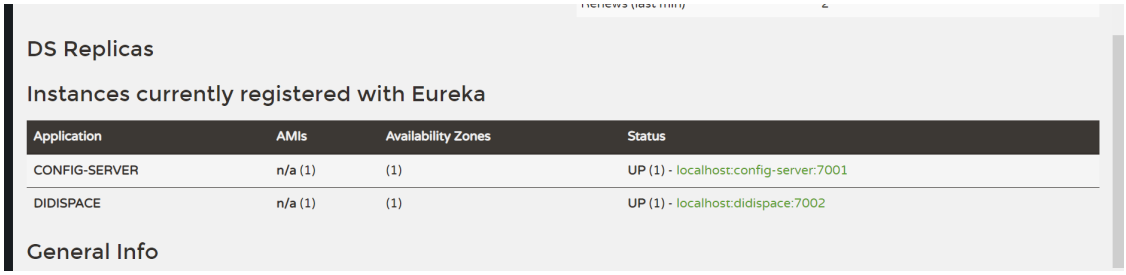
```
 <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
</dependency>
```

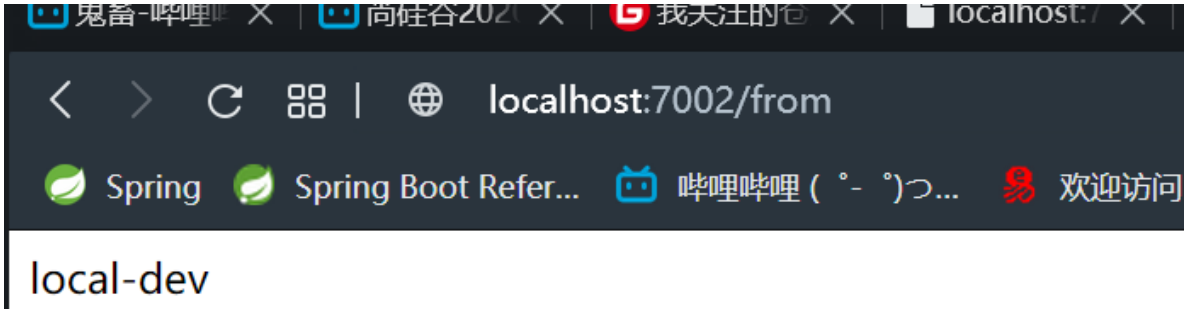
- 指定服务注册中心和服务

```
spring.application.name=didispace

spring.cloud.config.discovery.enabled=true
spring.cloud.config.discovery.service-id=config-server
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
spring.cloud.config.profile=dev

server.port=7002
```

- 开启注册功能

```
@EnableDiscoveryClient
```





Spring Cloud Bus

Spring Cloud Bus实时更新总线

> 服务端和客户端

添加依赖

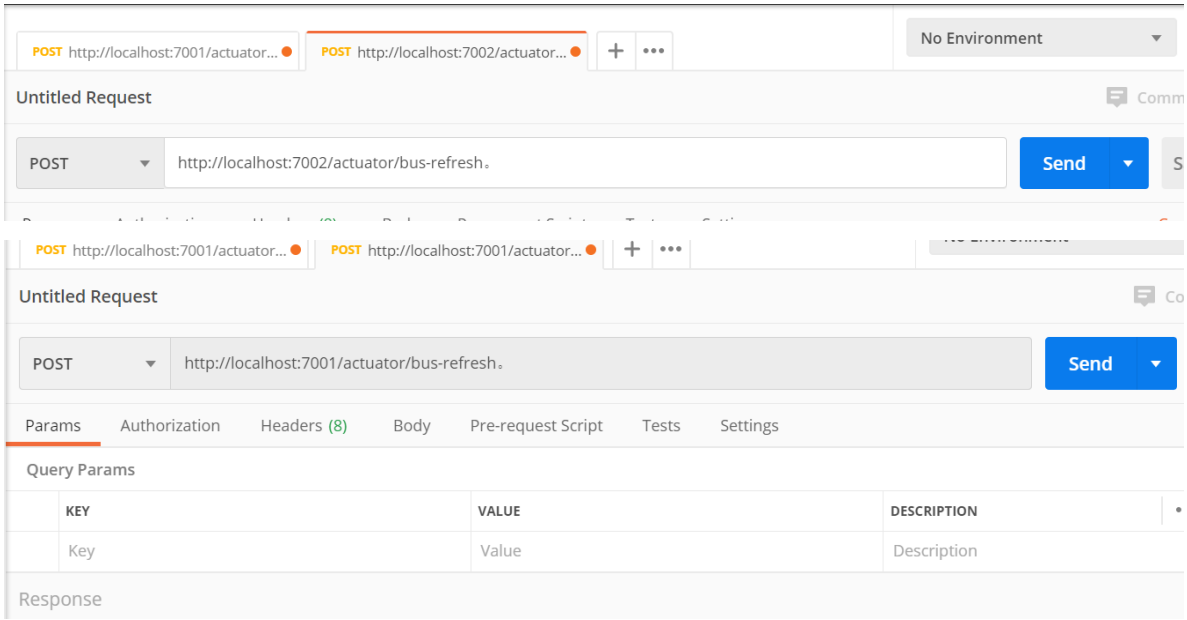
```
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

配置连接信息

```
# RabbitMQ
spring.rabbitmq.host=192.168.1.106
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

配置动态刷新端口

```
management.endpoints.web.exposure.include= bus-refresh
```



Spring Cloud Stream

构建RabbitMQ支持的消息驱动

添加依赖

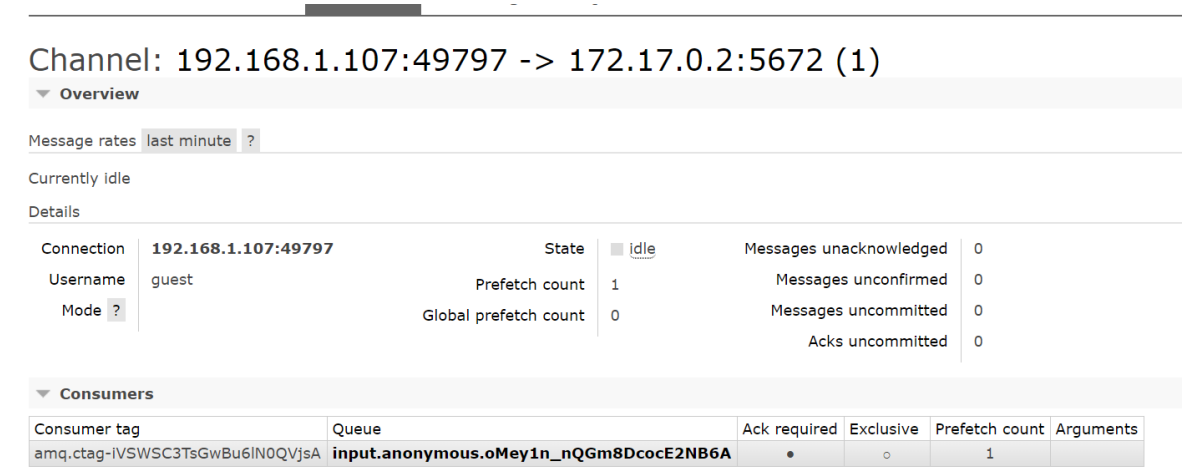
```
 <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>

<!--Alternatively -->
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

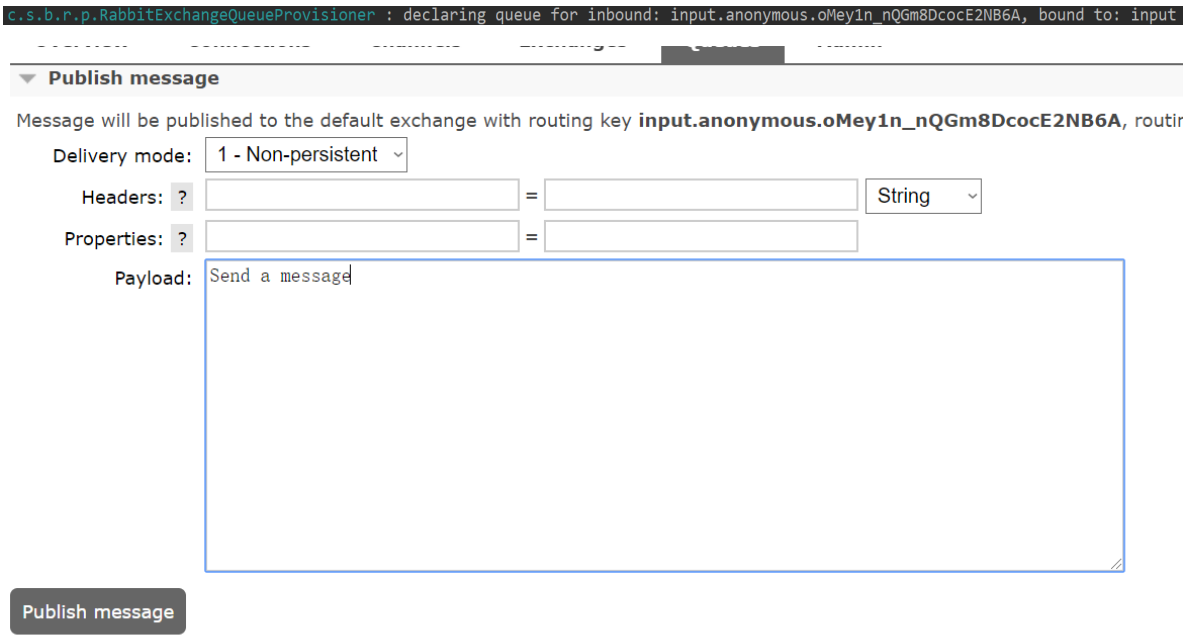
配置信息

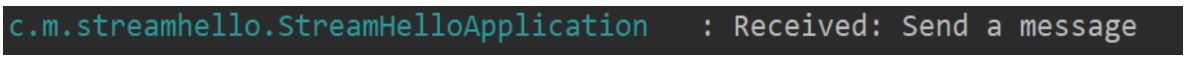
```
# RabbitMQ
spring.rabbitmq.host=192.168.1.106
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

创建消费者

```
//@EnableBinding用于实现对消息通道的绑定
//Sink是对输入消息通道绑定的定义
@EnableBinding(value = {Sink.class})
public class SinkReceiver1 {

    private static Logger logger =
LoggerFactory.getLogger(StreamHelloApplication.class);
//将被修饰的方法注册为消息中间件上数据流的事件监听器
    @StreamListener(Sink.INPUT)
    public void receive(Object payload) {
        logger.info("Received: " + payload);
    }

}
```







####

消费组

消费者应用

- 设置主题和消费组

```
#设置消费组
spring.cloud.stream.bindings.input.group=Service-A

#输入通道绑定目标指向greetings主题
spring.cloud.stream.bindings.input.destination=greetings
```

- 绑定输入通道

```
@EnableBinding(value = {Sink.class})
public class SinkReceiver4 {

    private static Logger logger =
LoggerFactory.getLogger(StreamConsumerApplication .class);


    @StreamListener(Sink.INPUT)
    public void receive(User user) {
        logger.info("Received: " + user);
    }


}
```

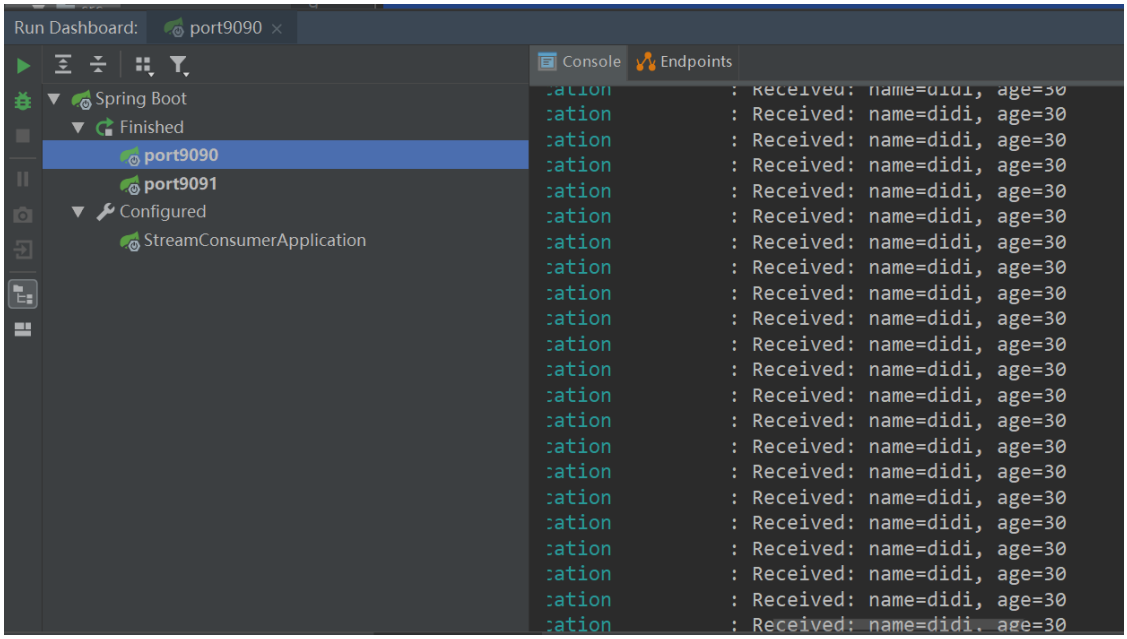
生产者应用

- 设置主题

```
#输出通道绑定目标指向greetings主题
spring.cloud.stream.bindings.output.destination=greetings
```

- 绑定输出通道

```
@EnableBinding(value = {Source.class})
public class SinkSender4 {

    private static Logger logger =
LoggerFactory.getLogger(StreamProducerApplication.class);

    @Bean
    @InboundChannelAdapter(value = Source.OUTPUT, poller =
@Poller(fixedDelay = "2000"))
    public MessageSource<String> timerMessageSource() {
        return () -> new GenericMessage<>("{\"name\":\"didi\",
\"age\":30}");
    }

}
```



消费分区

Spring Cloud Sleuth

实现服务跟踪功能

> 两个客户端应用

添加依赖

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-sleuth</artifactId>
        </dependency>
```

创建应用主类

```
@RestController
@EnableDiscoveryClient
@SpringBootApplication
public class Trace1Application {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Bean
    @LoadBalanced
    RestTemplate restTemplate() {
```

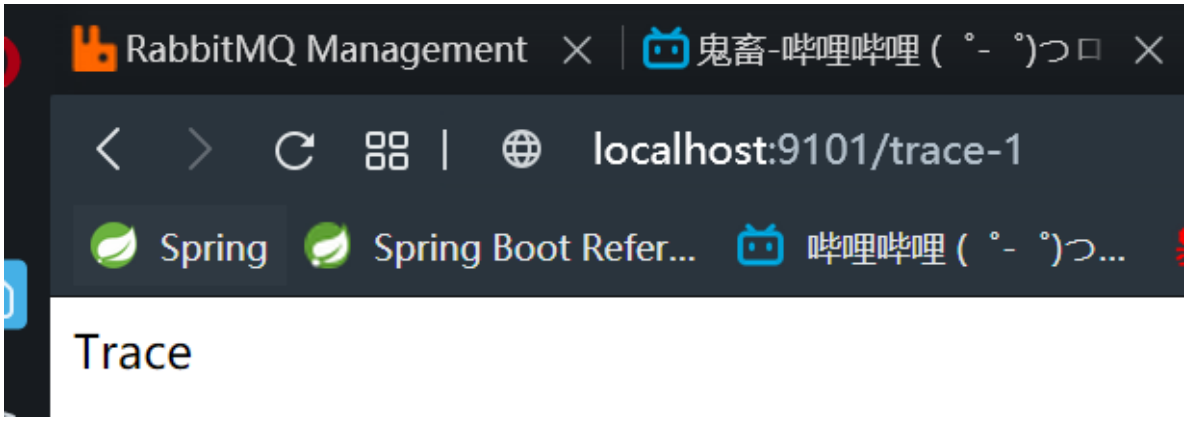
```
        return new RestTemplate();
    }


    @GetMapping("/trace-1")
    public String trace() {
        logger.info("===<call trace-1>===");
        return restTemplate().getForEntity("http://trace-2/trace-2",
String.class).getBody();
    }

    public static void main(String[] args) {
        SpringApplication.run(Trace1Application.class, args);
    }

}
```

```
@RestController
@EnableDiscoveryClient
@SpringBootApplication
public class Trace2Application {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    @GetMapping("/trace-2")
    public String trace(HttpServletRequest request) {
        logger.info("===<call trace-2, TraceId={}, SpanId={}>===",
                request.getHeader("X-B3-TraceId"), request.getHeader("X-B3-
SpanId"));
        return "Trace";
    }

    public static void main(String[] args) {
        SpringApplication.run(Trace2Application.class, args);
    }

}
```

指定服务中心

```
spring.application.name=trace-1
server.port=9101

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

```
spring.application.name=trace-2
server.port=9102

eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```



DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
TRACE-1	n/a (1)	(1)	UP (1) - localhost:trace-1:9101
TRACE-2	n/a (1)	(1)	UP (1) - localhost:trace-2:9102

```
s
6 : ===<call trace-1>===
  : Flipping property: TRACE-2.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.acti
  : Shutdown hook installed for: NFLoadBalancer-PingTimer-TRACE-2
  : Client: TRACE-2 instantiated a LoadBalancer: DynamicServerListLoadBalancer:{NFLoadBalancer:name=TRACE-2,current list of Servers
  : Using serverListUpdater PollingServerListUpdater
  : Flipping property: TRACE-2.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.acti
  : DynamicServerListLoadBalancer for client TRACE-2 initialized: DynamicServerListLoadBalancer:{NFLoadBalancer:name=TRACE-2,curren
ast connection made:Thu Jan 01 08:00:00 CST 1970; First connection made: Thu Jan 01 08:00:00 CST 1970;   Active Connections:0;   to

n.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647
```