1、为什么使用微服务?

- 单一服务存在的问题：1、复杂性扩散（当某个模块出问题以后会影响整个系统）；2、库的复用与耦合；3、可替代成本太高（但某个模块的请求量提升需要加机器时连带着其他服务也需要跟着部署）。
- 微服务架构的优缺点：
  - 优点：1、每个模块职责功能比较单一，每个服务都很简单；2、易于规模化开发，单独团队维护、工作分期、职责清晰；3、改善故障隔离，一个服务出问题不会影响其他服务；4、架构上更加清晰。
  - 缺点：1、随着服务增加，运维压力增大；2、系统部署互相依赖（facade包）；3、服务间的通信成本；4、分布式事务问题。

2、SpringCloud基于SpringBoot开发（各组件Eureka等都是基于springboot搭建），可以实现项目的打包发布以及单独运行（jar包运行）。

3、RestTemplate：RestTemplate类的设计原则与许多其他Spring模板类(例如JdbcTemplate、JmsTemplate)相同，可用于在应用中调用rest服务，它简化了与http服务的通信方式，统一了RESTful的标准，封装了http链接， 我们只需要传入url及返回值类型即可。相较于之前常用的HttpClient，RestTemplate是一种更优雅的调用RESTful服务的方式。

4、Security安全验证，引入spring-boot-starter-security依赖。详细配置看word。

Eureka（https://www.cnblogs.com/zyon/p/11023750.html）----------

一、Eureka服务搭建

1. 引入spring-cloud-starter-netflix-eureka-server依赖：

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

2. 在application.yml中配置port等：

```yaml
server:
  port: 7001
eureka:
  instance:
    hostname: 127.0.0.1
    prefer-ip-address: true
```

实例应用主机名

3. 在启动类加@EnableEurekaServer注解：

```java
@SpringBootApplication
@EnableEurekaServer
public class EurekaApp1 {
    public static void main(String[] args) {
        SpringApplication.run(EurekaApp1.class,args);
    }
```

4. 启动Eureka，启动以后Eureka可用但是会抛异常（Connection refused），抛异常是因为以下两个配置默认为true，将它门设置为false即可：

- eureka.client.fetch-registry：是否从Eureka获取注册信息，缺省：true，一般情况下Eureka服务端是不需要的。
- eureka.client.register-with-eureka：是否向注册中心注册自己，缺省：true，一般情况下，Eureka服务端是不需要再注册自己的。

二、客户端注册到Eureka

1. 客户端引入spring-cloud-starter-netflix-eureka-client依赖：

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

2. 在application.yml中加配置：

```yaml
eureka:
  client:
    service-url:
      defaultZone: http://ljq:ljq@127.0.0.1:7001/eureka
  instance:
    instance-id: cloudystudy-provider
```

Eureka服务地址

修改Status栏的名称

3. 在启动类加@EnableEurekaClient注解：

```
@SpringBootApplication
@MapperScan(value = "cn.ljq.mapper")
@EnableEurekaClient
public class ProductApp {
```

三、当客户端服务下线以后Eureka注册中心不会移除该实例（自我保护机制）

1. 自我保护机制：Eureka Server在运行期间会去统计心跳失败比例在 15 分钟之内是否低于 85%，如果低于 85%，Eureka Server会将这些实例保护起来，让这些实例不会过期，但是在保护期内如果服务刚好这个服务提供者非正常下线了，此时服务消费者就会拿到一个无效的服务实例，此时会调用失败，对于这个问题需要服务消费者端要有一些容错机制，如重试，断路器等。Eureka 的自我保护模式是有意义的，该模式被激活后，它不会从注册列表中剔除因长时间没收到心跳导致租期过期的服务，而是等待修复，直到心跳恢复正常之后，它自动退出自我保护模式。这种模式旨在避免因网络分区故障导致服务不可用的问题。例如，两个客户端实例 C1和C2的连通性是良好的，但是由于网络故障，C2未能及时向 Eureka发送心跳续约，但是C1与Eureka之间通信正常，这时候 Eureka 不能简单的将 C2 从注册表中剔除。因为如果剔除了，C1 就无法从 Eureka 服务器中获取 C2 注册的服务，但是这时候 C2 服务是可用的。（保护机制的意义：客户端之间的网络通信是正常的，即服务之间调用正常，但是某个服务与Eureka Server之间的通信异常了，此时Eureka不应该下线这个服务，因为服务之间调用正常）。

2. 当服务下线以后想要Eureka立马移除服务实例改如何做?

a. 在Eureka服务端增加以下配置：

i. 关闭保护模式：eureka.server.enable-self-preservation=false

ii. 将服务清理间隔时间设置为1秒（单位为毫秒，默认60秒）：eureka.server.eviction-interval-timer-in-ms=1000

b. 在客户端增加以下配置：

i. 将心跳时间间隔设置为2秒（默认30秒）：erueka.instance.lease-renewal-interval-in-seconds=2

ii. 将服务失效时间设置为5秒，失效的服务将被剔除（默认90秒）：eureka.instance.lease-expiration-duration-in-seconds=5

四、Eureka安全机制

1. 设置背景：一般情况下Eureka 和服务的提供注册者都会在一个内网环境中，但免不了在某些项目中需要让其他外网的服务注册到Eureka，这个时候就有必要让Eureka增加一套安全认证机制了，让所有服务提供者通过安全认证后才能注册进来。

2. 在Eureka服务端引入spring-boot-starter-security依赖：

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

3. 在Eureka服务端application.yml文件加配置：

```
spring:
  application:
    name: eureka-server1
  security:
    user:
      roles: user
      password: ljq
      name: ljq
```

4. 对于Edgware之后的版本还需要将csrf劫持关闭：

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception{
        http.csrf().disable();
        super.configure(http);
    }
}
```

四、Eureka高可用（HA），保证AP，集群搭建

1. Eureka高可用是通过自注册方式实现的，所以需要在Eureka服务端需要配置其他Eureka服务的地址（例如Eureka1的application.yml文件中需要配置Eureka2、Eureka3的地址，备注：截图中使用eureka1、eureka2、erueka3是因为我本机部署集群，在hosts文件中加了映射，在实际生产中配置直接配置IP地址就可以了）：

```yaml
client:
  fetch-registry: false
  register-with-eureka: false
  service-url:
    defaultZone: http://ljq:ljq@eureka2:7002/eureka,http://ljq:ljq@eureka3:7003/eureka
```

在eureka1中配置eureka2、eureka3的地址

2. 客户端也需要修改service-url，因为Eureka不像Zookeeper那样会进行数据同步（写到主以后，主会将数据同步到从），所以在客户端我们需要配置所有Eureka服务的地址，即将服务显示注册到每个注册中心：

```yaml
eureka:
  client:
    service-url:
      defaultZone: http://ljq:ljq@eureka1:7001/eureka,http://ljq:ljq@eureka2:7002/eureka,http://ljq:ljq@eureka3:7003/eureka
```

需要配置每个注册中心的地址，以逗号分隔

五、Eureka与Zookeeper的区别

1. 著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。在此Zookeeper保证的是CP，而Eureka则是AP。

https://www.cnblogs.com/chihirotan/p/11366394.html

六、客户端查看状态信息（需要actuator模块，和springboot使用方式一样）

1. actuator默认只加载info、health信息，加载全部需要配置如下：

```yaml
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

@Feign完全集成了springMvc的注解

springcloud服务的名字不需要区分大小写，大写小写都可以。

Ribbon脱离Eureka使用的时候不需要加@LoadBalance注解

客户端负载和服务端负载的区别。Ribbon为客户端负载

@RibbonClients的用法

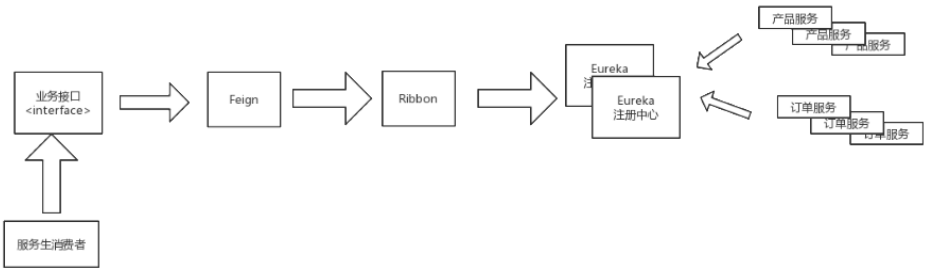
@FeignClient注解是在接口上边使用，而@RibbonClient、RibbonClients注解是在启动类上边使用

服务调用者根据业务情况来决定是否注册到Eureka：即是服务调用方也是服务提供方就需要注册，如果只是调用其他服务则不需要调用

eureka集群通过自注册实现，

yml文件中使用${}达到复用

feign调用流程要理解并记住



当feign不结合ribbon单独使用的时候可以通过@FeignClient注解的url属性指定服务地址（这种方式一般在测试的时候用），具体查看seata-samples中的使用方式

同一个服务的spring.application.name一定要相同（在腾讯云部署的时候是统一部署的没咋注意）。使用同一个服务名的多个服务里边内容不一定要保证相同，只要服务名相同注册到注册中心以后它们就是同一个服务。在例子中三个服务使用的数据库地址不一样。单功能一样也是可以的

对于服务的熔断机制，其实需要考虑两种情况：
1. 服务提供方存活，但调用接口报错（此时调用服务提供方的fallback方法）
2. 服务提供方不可用了，例如宕机（此时需要在服务消费方提供fallback方法）

Hystrix是通过弱化服务之间的强依赖性来防止服务雪崩的。（当没有hystrix的时候A服务调用B服务一直失败则会造成A服务上的请求累积最终奔溃，而引入hystrix后当A调用B失败达到一定阈值以后A就会调用本地方法从而保证整个服务可用）

PPT中的Hystrix熔断机制的两个关注点需要注意，对这两个关注点的个人理解：

服务提供者降级：假如B服务能被A服务正常调用到，但是B服务的某些业务方法有异常，此时可以加一些备胎方法来替换这些业务方法（即调用业务方法失败时去调用备胎方法，保证B服务的可用性）

服务调用者熔断：假如B服务宕机了，此时A去调用B服务肯定失败，这种情况下当失败次数达到一定阈值后A服务会进行熔断（即不在去调用B服务，而是调用A服务中的本地方法）

```
// 定义回退方法的名称, 此方法必须和hystrix的执行方法在相同类中
  String fallbackMethod() default "";
```

Feign和Ribbon是在服务调用方启动类加注解开启。Hystrix是在服务提供方启动类加注解开启。

Hystrix服务降级处理时我们需要为接口的每个方法都提供备胎（因为请求参数和返回参数不同），这样就需要写很多重复的代码，处理方式显得很笨拙（在测试的时候为get和add方法都需要添加一个备胎，因为备胎方法和正牌方法的参数是必须要一致的）。为了避免这种情况，我们项目中需要统一请求报文和返回报文，就像我们目前的代码中一样所有服务的请求报文和返回报文继承于同一个基类，这样我们只需要写一个备胎方法就ok了。

HystrixDashboard和eureka的访问路径需要加后缀，一个加/hystrix，一个加/eureka

Hystrix断路器：断路器很好理解, 当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN). 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN). Hystrix的断路器就像我们家庭电路中的保险丝, 一旦后端服务不可用, 断路器会直接切断请求链, 避免发送大量无效请求影响系统吞吐量, 并且断路器有自我检测并恢复的能力.

HystrixDashBoard监控的方法必须加上@HystrixCommand注解（老师的word中在举例时新加的user模块中的get方法上边加了这个注解）。

HystrixDashboard和Turbine结合监控多个服务的原理：Turbine从注册中心获取所有服务信息，然后在HystrixDashBoard的主界面填写Turbine的地址进行监控。Turbine相当于一个中间人，HystrixDashBoard通过这个中间人去监控所有服务。

Turbine配置集群名称时一定要注意（配置default时是小括号，不是大括号）
https://www.jianshu.com/p/590bad4c8947:

```
turbine:
  app-config: CLOUDSTUDY-USER-PROVIDER,CLOUDYSTUDY-PROVIDER
  cluster-name-expression: new String("default") #集群名称
```

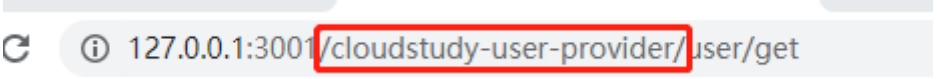
思考：zuul的高可用（http://www.itmuch.com/spring-cloud/zuul/zuul-ha/）。如图8-8，Zuul客户端将请求发送到负载均衡器，负载均衡器将请求转发到其代理的其中一个Zuul节点。这样，就可以实现Zuul的高可用。

springcloud搭建配置中心时，可以通过label属性指定获取git那个分支的配置文件。

配置中心的高可用需要另外一套eureka注册中心，而且这个eureka注册中心的配置不是通过配置中心获取的，而是在项目中配置的。因为要将配置中心注册到这个eureka集群，如果这个eureka集群的配置是从配置中心拉取的那就意味着先要将配置中心启动起来，而配置中心是要注册到这个注册中心的，那意味着配置中心要求eureka集群先要起来自己才能注册，这样就会产生矛盾，这也是为什么单独要为配置中心搭建一套eureka集群的原因（因为业务应用所用的eureka集群是要从配置中心获取配置的）

zuul其实是一个API网关，类似于设计模式里面的门面模式，它的存在就像是整个微服务集群的门面，所有的外部客户端访问都需要经过它来进行调度与过滤

通过zuul拼接url访问服务提供者时，需要将application.name小写：



疑问：关于同一个服务部署多台机器，假如有一台机器挂了，则有请求的时候还会发到这台机器吗？看eureka源码之后在解答。

当消费者通过zuul去访问其它服务提供者时，可以在zuul的接口中统一做服务降级处理（因为它整合了所有服务提供者的接口），而不需要为每个服务提供者分开写降级处理（在例子中只需要给CloudStudyZuulService接口写降级实现）。而如果消费者有单独调某个服务提供者的需求，则要为这个服务提供者接口写降级实现（例子中为ProductFeignService写降级实现）