微服务开发框架
Spring cloud

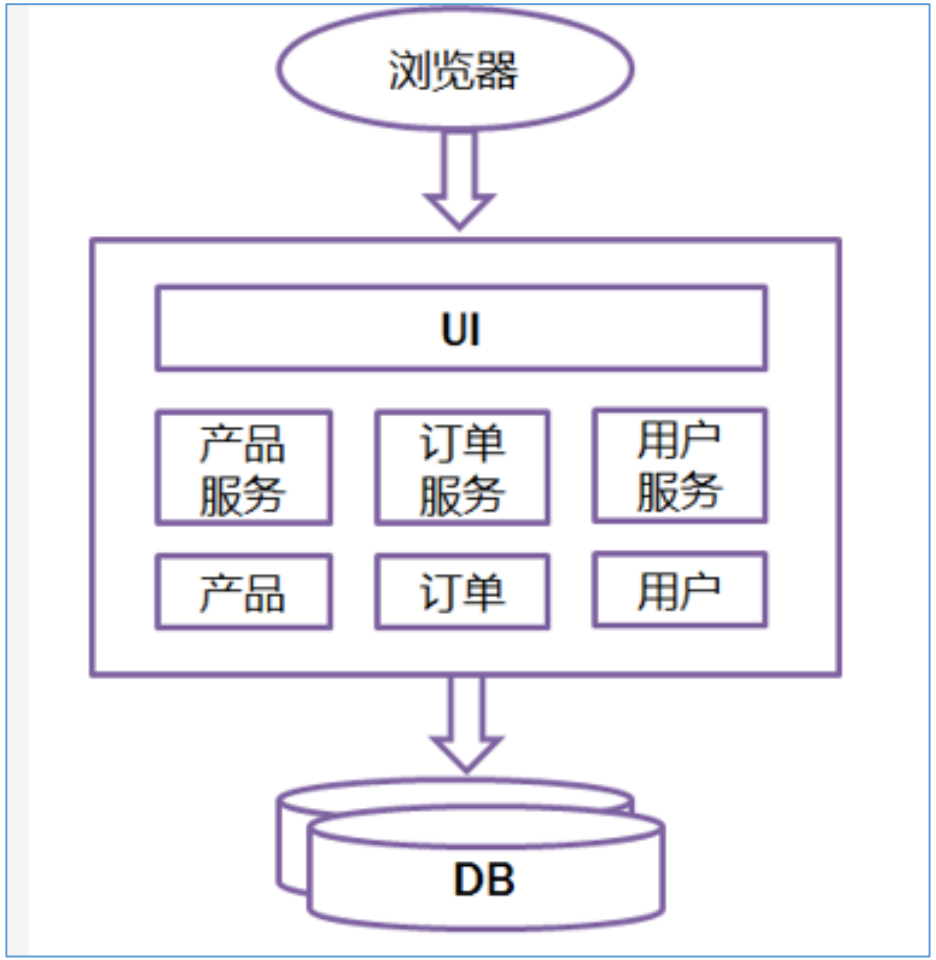
李剑

内容

- 微服务架构概述
- 微服务开发框架-spring cloud
- 微服务应用组件-基于spring cloud
- 微服务架构应用案例

单体应用

一个归档包（war）包含所有功能的应用程序。

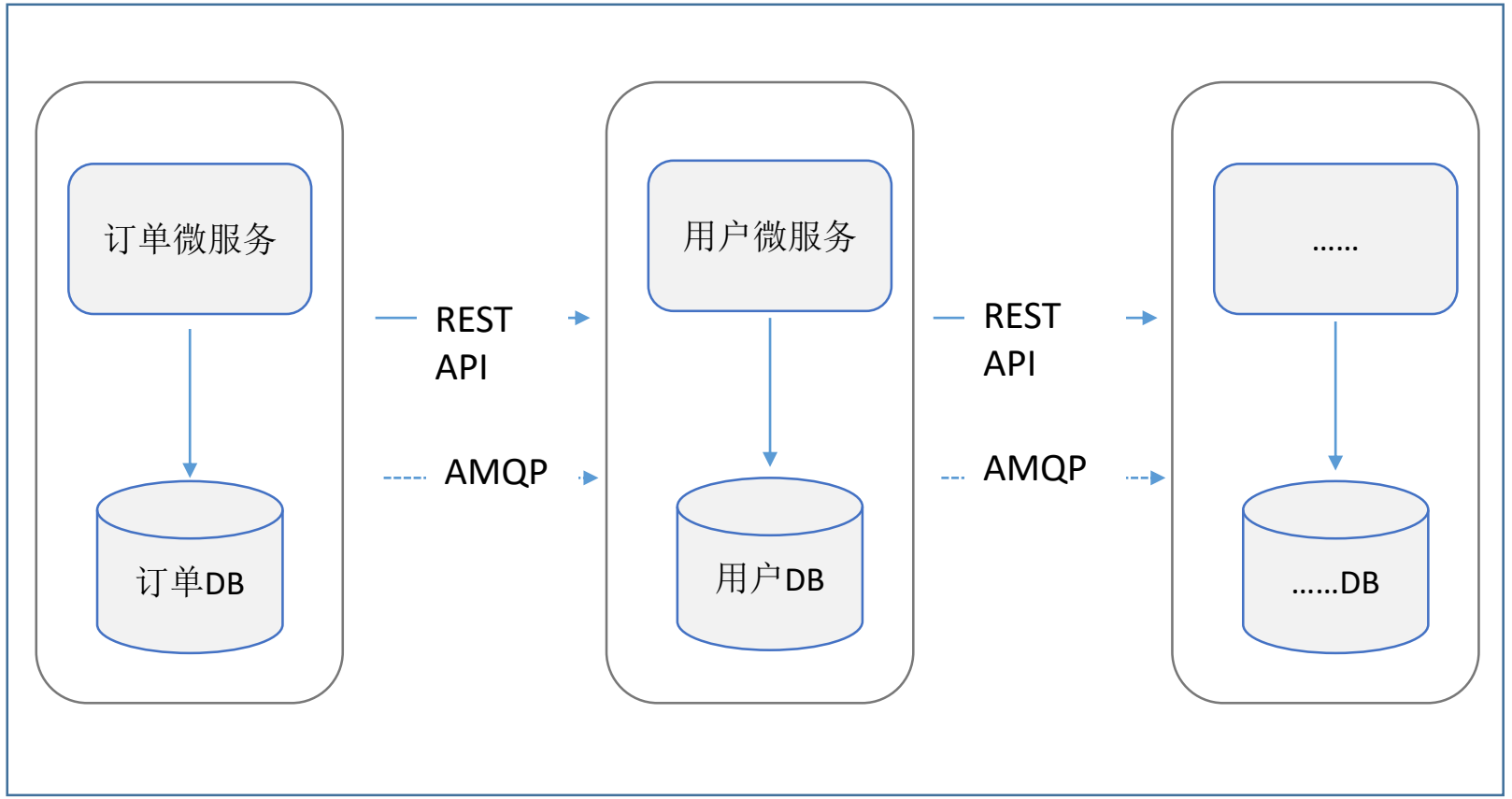


存在问题

- 代码维护难：代码功能耦合在一起，新人不知道何从下手
- 部署不灵活：构建时间长，任何小修改必须重新构建整个项目，这个过程往往很长
- 稳定性不高：一个微不足道的小问题，可以导致整个应用挂掉
- 扩展性不够：无法满足高并发情况下的业务需求
- 妨碍持续交付
- 受技术栈限制
- .....

微服务架构

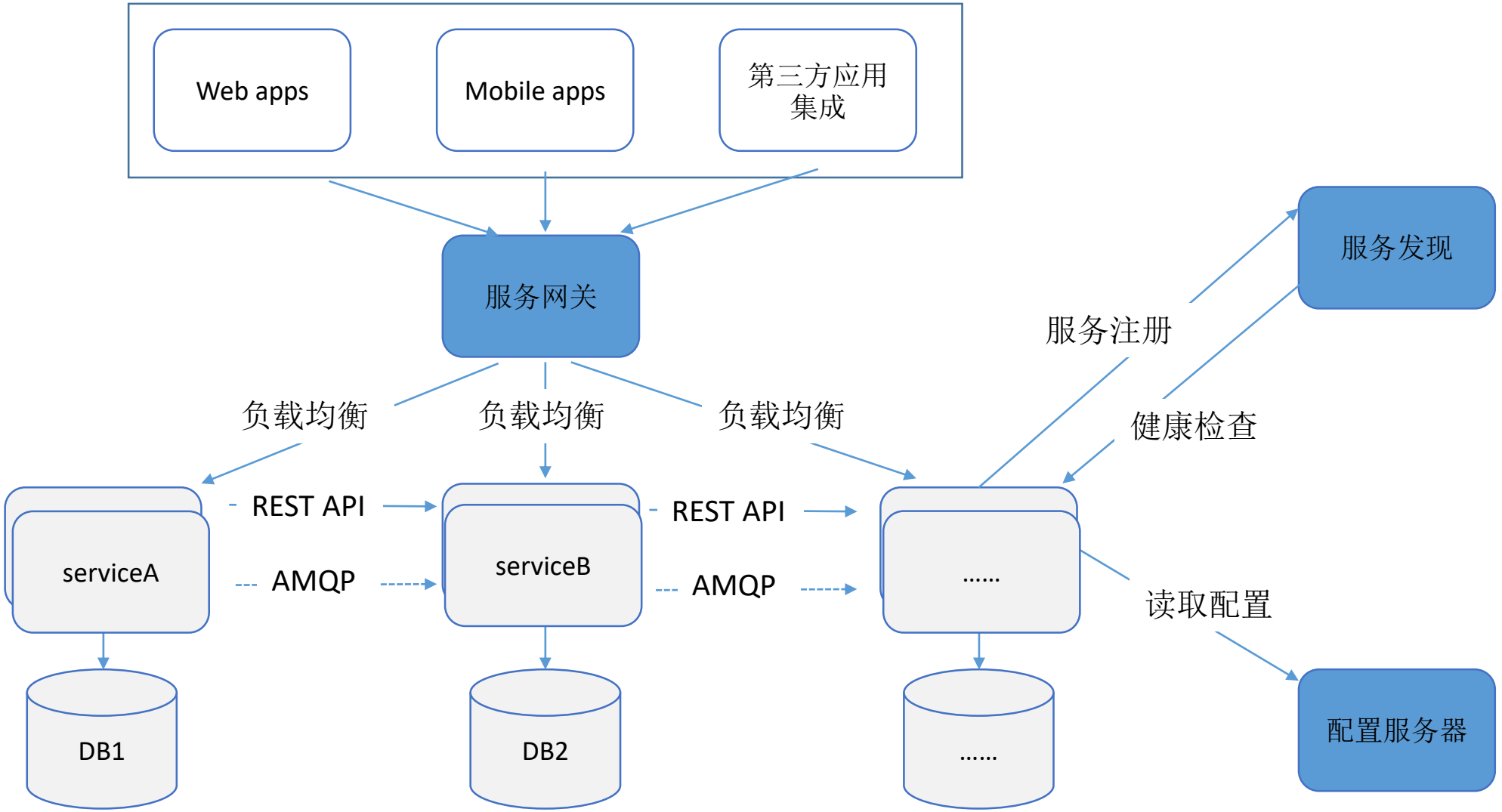
*微服务架构是将一个单体应用程序开发为**一组小型服务**的方法。*



特点

- 易于理解、开发和维护；
- 比单体应用启动快；
- 局部修改很容易部署，有利于持续集成和持续交付；
- 故障隔离，一个服务出现问题不会影响整个应用；
- 不会受限于任何技术栈
- ……

微服务架构图及常用组件



内容

- 微服务架构概述
- 微服务开发框架-spring cloud
- 微服务应用组件-基于spring cloud
- 微服务架构应用案例

spring cloud概念

基于spring boot，是微服务开发和治理框架。

Spring cloud特点

- 开箱即用、快速启动
- 提供声明式、无xml配置方式
- 轻量级组件
- 组件丰富，功能齐全
- 选型中立、丰富
- 灵活

版本简介

http://projects.spring.io/spring-cloud/#quick-start

Spring Cloud builds on Spring Boot by providing a bunch of libraries that enhance the behaviour of an application when added to the classpath. You can take advantage of the basic default behaviour to get started really quickly, and then when you need to, you can configure or extend to create a custom solution.

Quick Start

The release train label (see below) is actually only used explicitly in one artifact: "spring-cloud-dependencies" (all the others have normal numeric release labels tied to their parent project). The depednencies POM is the one you can use as a BOM for dependency management. Example using the latest version with the config client and eureka (change the artifact ids to pull in other starters):

Download | Dalston SR1 CURRENT GA | MAVEN GRADLE

The recommended way to get started using `spring-cloud` in your project is with a dependency management system – the snippet below can be copied and pasted into your build. Need help? See our getting started guides on building with Maven and Gradle.

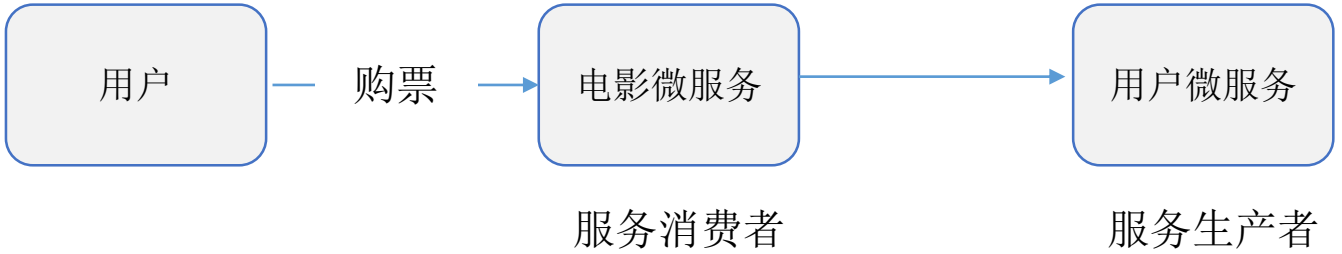
```
<parent>
    <groupId>org.springframework.boot</groupId>
```

Spring Cloud

RELEASE	DOCUMENTATION
Finchley M1 PRE	
Finchley SNAPSHOT	
Edgware SNAPSHOT	Reference
Dalston SR1 CURRENT GA	Reference
Camden SR7 GA	Reference
Brixton SR7 GA	Reference
Angel SR6 GA	Reference

- 主版本-Dalston 伦敦地铁站的名称，按照字母排序发行（Angel、Brixton、Camden）
- 以SR（Service Release）X（X为数字）命名版本号-表示bug修复

spring cloud-开发微服务



添加 spring boot 和 spring cloud 依赖

```
<!-- 引入spring boot的依赖 -->
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.4.RELEASE</version>
</parent>


<!-- 引入spring cloud的依赖 -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Dalston.SR1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Rest服务

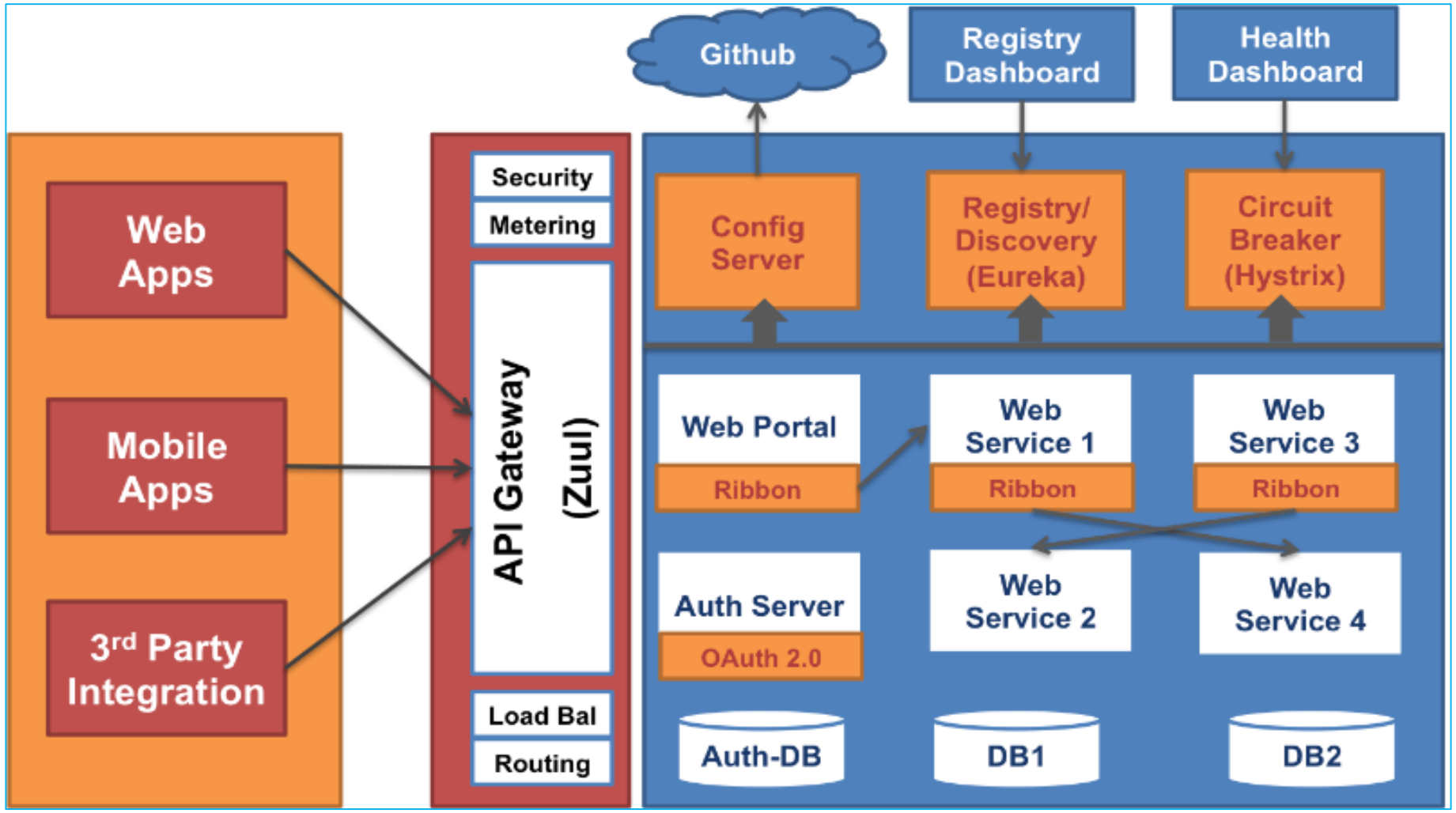
```
@RestController
public class UserController {
  @Autowired
  private UserRepository userRepository;

  @GetMapping("/{id}")
  public User findById(@PathVariable Long id) {
```

启动spring boot 应用

```
@SpringBootApplication
public class ProviderUserApplication {
  public static void main(String[] args) {
    SpringApplication.run(ProviderUserApplicatio
  }
}
```

架构图及主要组件


Github
Registry Dashboard
Health Dashboard
Web Apps
Mobile Apps
3rd Party Integration
Security
Metering
API Gateway (Zuul)
Load Bal
Routing
Config Server
Registry/ Discovery (Eureka)
Circuit Breaker (Hystrix)
Web Portal
Ribbon
Web Service 1
Ribbon
Web Service 3
Ribbon
Auth Server
OAuth 2.0
Web Service 2
Web Service 4
Auth-DB
DB1
DB2

内容

- 微服务架构概述
- 微服务开发框架-spring cloud
- 微服务应用组件-基于spring cloud
- 微服务架构应用案例

主要应用组件介绍

Eureka

Netflix

云端服务发现，一个基于 REST 的服务，用于定位服务，以实现云端中间层服务发现和故障转移。

Spring Cloud Config

Spring

配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git以及Subversion。

Zuul

Netflix

Zuul 是在云平台上提供动态路由,监控,弹性,安全等边缘服务的框架。Zuul 相当于是设备和 Netflix 流应用的 Web 网站后端所有请求的前门。

Ribbon

Netflix

提供云端负载均衡，有多种负载均衡策略可供选择，可配合服务发现和断路器使用。

Feign

OpenFeign

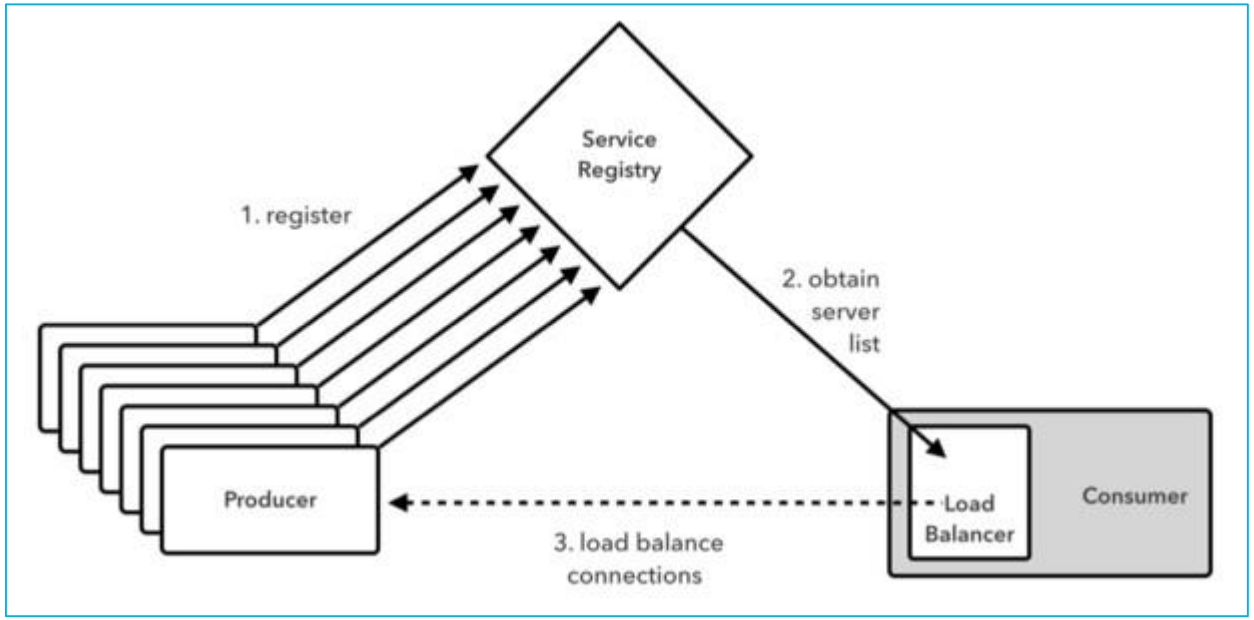
Feign是一种声明式、模板化的HTTP客户端。

Hystrix

Netflix

熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

应用组件-服务注册发现（Eureka）



- 服务注册/发现
- 服务注册表-记录微服务的信息
- 服务检查-定时检查已注册的服务，移除长时间无法访问的实例

Eureka依赖

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

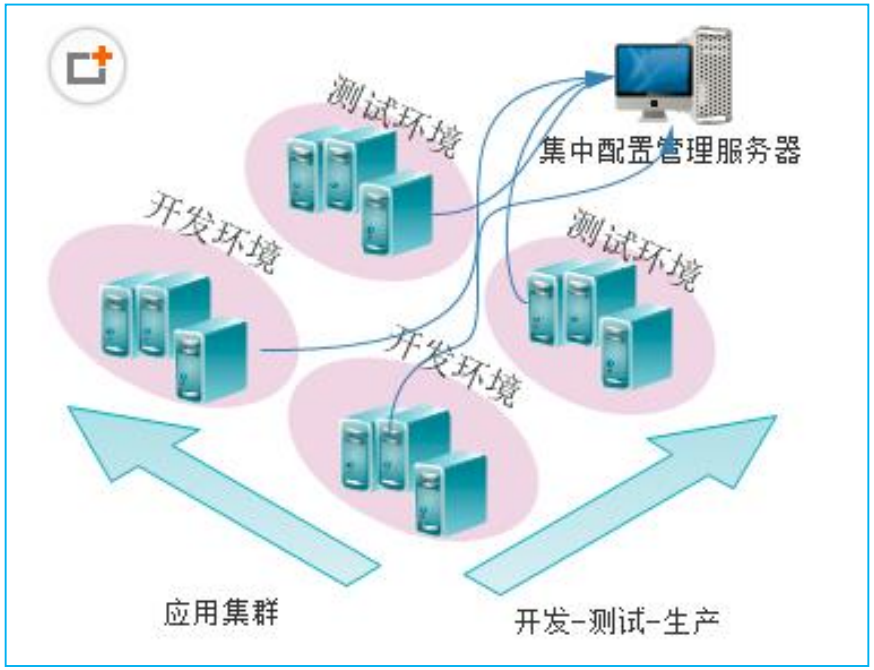
服务端声明

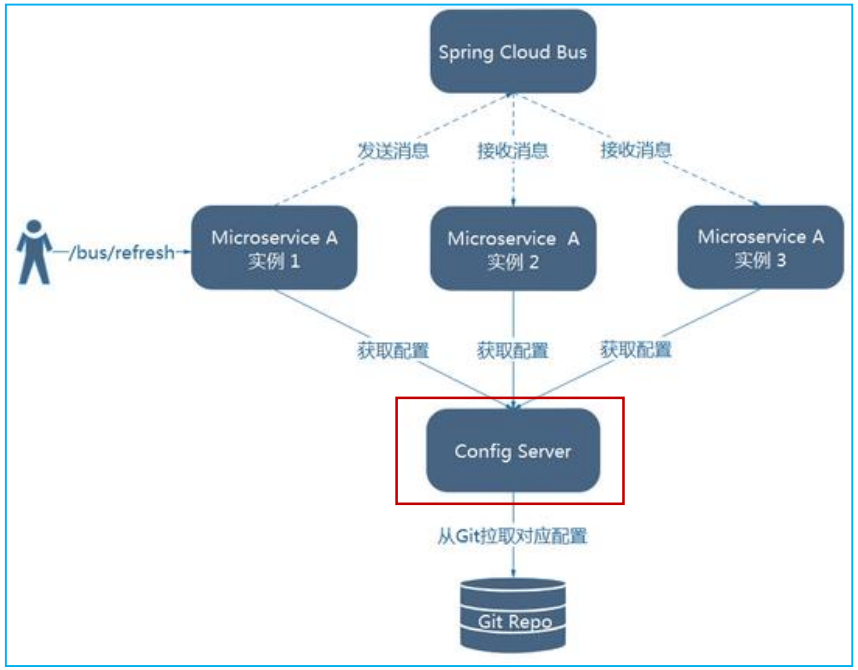
```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {
  public static void main(String[] args) {
    SpringApplication.run(EurekaApplication.class, a
  }
}
```

客户端声明

```
@EnableDiscoveryClient
@SpringBootApplication
public class ProviderUserApplication {
  public static void main(String[] args) {
    SpringApplication.run(ProviderUserApplicatio
  }
}
```

应用组件-配置中心（config）





Config server依赖

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

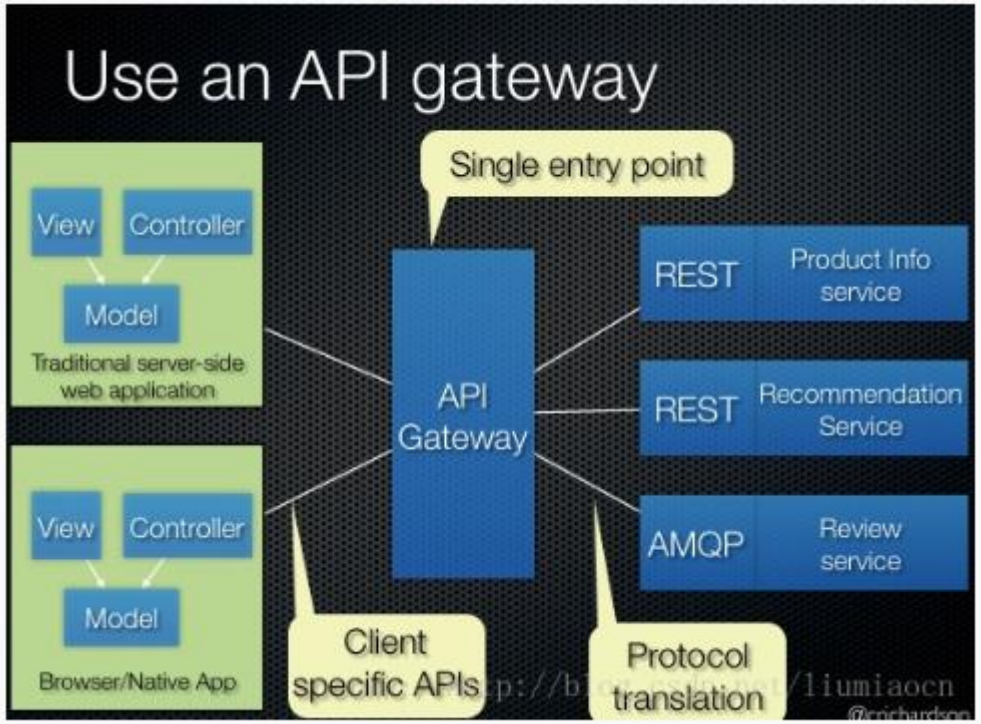
配置文件

```
server:
  port: 8080
    ng:
    plication:
    name: microservice-config-server
    oud:
    config:
      server:
        git:
          uri: https://git.oschina.net/itmuch/spring-cloud-confi
          username:
          password:
```

服务端声明

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
  public static void main(String[] args) {
    SpringApplication.run(ConfigServerApplication
  }
}
```

应用组件-服务网关（zuul）



- 身份验证
- 动态路由
- 协议转换
- 安全
- .....

zuul依赖

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

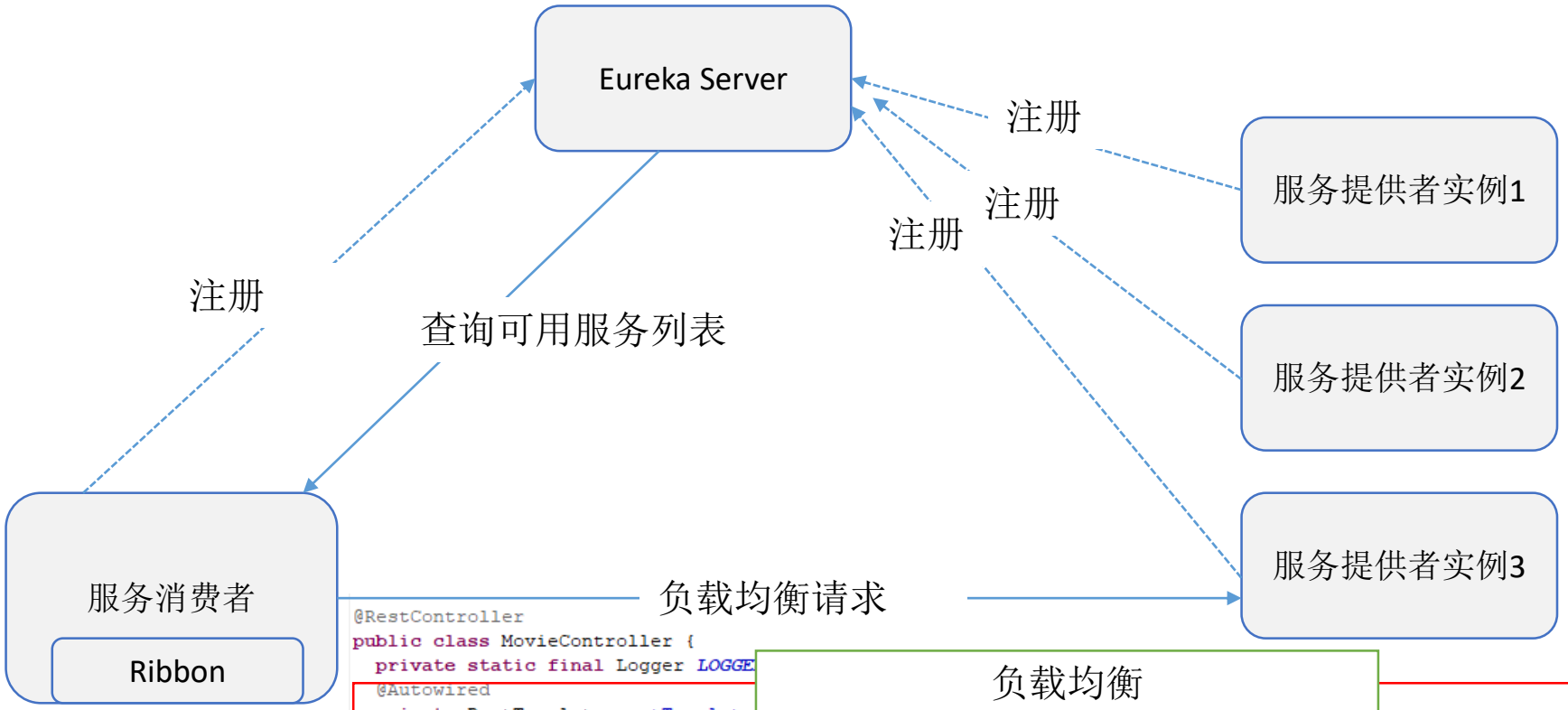
配置文件

```
zuul:
  routes:
    microservice-provider-user: /user/**
```

应用声明

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

应用组件-负载均衡（Ribbon）




```
@RestController
public class MovieController {
  private static final Logger LOGGE
  @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-user-instance")
  public void logUserInstance() {
    ServiceInstance serviceInstance = this.loadBalancerClient.choose("microservice-provider-user");
    // 打印当前选择的是哪个节点
    MovieController.LOGGER.info("{}:{}:{}", serviceInstance.getServiceId(), serviceInstance.getHost(), serviceInstance.getPort());
```

应用组件-Rest服务调用（Feign）

Feign是Netflix公司开发的声明式、模板化的HTTP客户端。

Feign依赖

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

应用声明

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

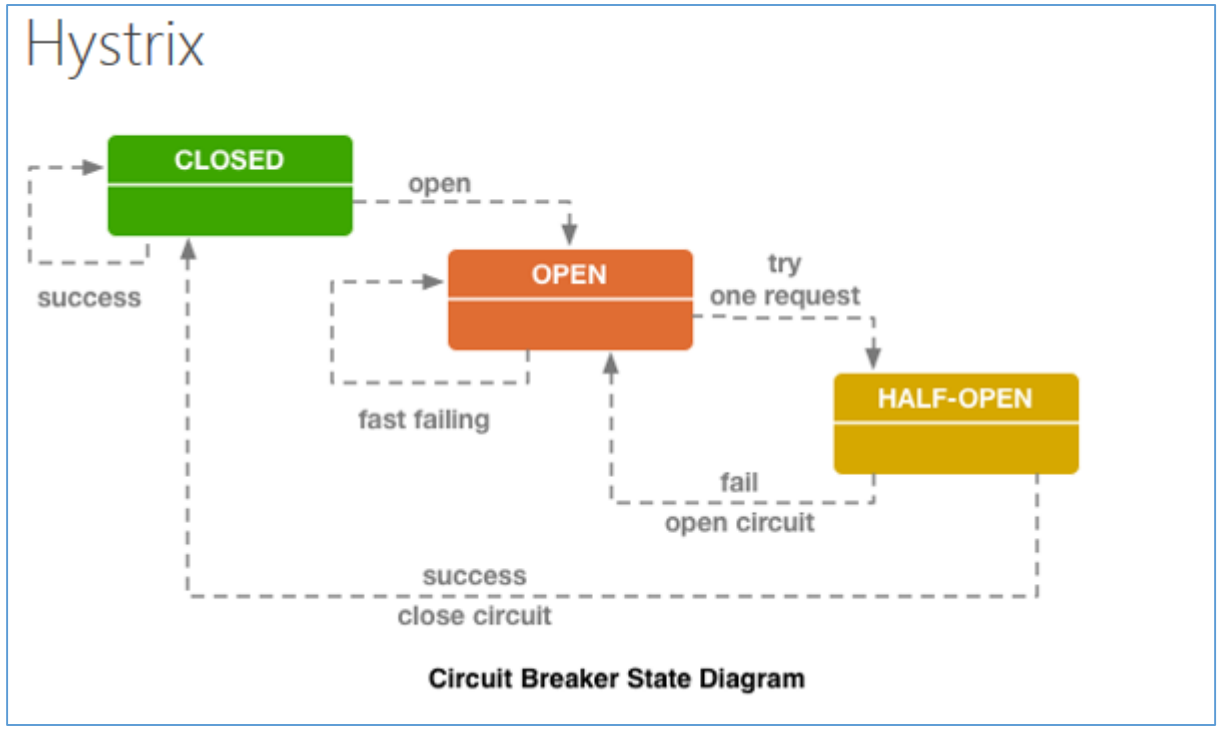
Feign应用

```java
@RestController
public class MovieController {
  @Autowired
  private UserFeignClient userFeignClient;

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

```java
@FeignClient(name = "microservice-provider-user")
public interface UserFeignClient {
  @RequestMapping(value = "/{id}", method = Reque
  public User findById(@PathVariable("id") Long i
}
```

应用组件-容错处理（Hystrix）



状态

- Closed
- Open
- Half-open

FeignClient声明定义fallback class

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

}
```

实现FeignClient

```
@Component
class FeignClientFallback implements UserFeignClient {
  @Override
  public User findById(Long id) {
    User user = new User();
    user.setId(-1L);
    user.setUsername("默认用户");
    return user;
  }
}
```

Netflix 微服务应用

Netflix是一家在全球范围内提供流视频服务的公司，截止到2016年已经拥有8300+万订阅用户，每天播放时间达到了1亿2千万小时，是北美互联网峰值下载量的1/3。

为了支撑庞大的用户访问，Netflix从2009便下决心向云原生的微服务生态系统演进，在2016年完成了整体应用迁徙到云端，目前拥有500+的微服务，在系统演进的7年内，Netflix的流量增长了1000多倍，可谓是开着火箭换发动机。坚决的转向微服务+云原生让Netflix演进为一家成功实践微服务架构的互联网公司，更重要的是Netflix几乎开源了公司内部全部的微服务架构技术栈，这套架构在Netflix公司大规模分布式微服务环境中经过数年的生产环境检验被证明是可靠的。对于打算采用微服务架构的公司来说，可以在这套事实标准架构的基础上进行符合自身业务需求的定制化。所以看看Netflix给出的菜：

Netflix OSS

- 来源-http://www.jianshu.com/p/22fefd73fc46

- 建议？