

一、环境准备

springboot+mybatis整合

1.创建父工程

New Maven Project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

▶ **Advanced**

1. POM依赖

```
1 <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">
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.haoge.cloud</groupId>
4   <artifactId>springcloud</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <packaging>pom</packaging>
7
8   <properties>
9     <project.build.sourceEncoding>UTF-
10    8</project.build.sourceEncoding>
11
12    <maven.compiler.source>1.8</maven.compiler.sourc
13    e>
14    <maven.compiler.target>1.8</maven.compiler.targe
15    t>
16
17    <junit.version>4.12</junit.version>
18    <log4j.version>1.2.17</log4j.version>
19    <lombok.version>1.16.18</lombok.version>
20  </properties>
```

```
16      <!-- 父类工程的管理 -->
17      <dependencyManagement>
18          <dependencies>
19              <dependency>
20
21                  <groupId>org.springframework.cloud</groupId>
22                  <artifactId>spring-cloud-
dependencies</artifactId>
23                  <version>Dalston.SR1</version>
24                  <type>pom</type>
25                  <scope>import</scope>
26              </dependency>
27              <dependency>
28
29                  <groupId>org.springframework.boot</groupId>
30                  <artifactId>spring-boot-
dependencies</artifactId>
31                  <version>1.5.9.RELEASE</version>
32                  <type>pom</type>
33                  <scope>import</scope>
34              </dependency>
35              <dependency>
36
37                  <groupId>mysql</groupId>
38                  <artifactId>mysql-connector-
java</artifactId>
39                  <version>5.0.4</version>
40              </dependency>
41              <dependency>
```

```
39         <groupId>com.alibaba</groupId>
40         <artifactId>druid</artifactId>
41         <version>1.0.31</version>
42     </dependency>
43     <dependency>
44
45     <groupId>org.mybatis.spring.boot</groupId>
46     <artifactId>mybatis-spring-boot-
47 starter</artifactId>
48     <version>1.3.0</version>
49     </dependency>
50     <dependency>
51     <groupId>ch.qos.logback</groupId>
52     <artifactId>logback-
53 core</artifactId>
54     <version>1.2.3</version>
55     </dependency>
56     <dependency>
57     <groupId>junit</groupId>
58     <artifactId>junit</artifactId>
59     <version>${junit.version}</version>
60     <scope>test</scope>
61     </dependency>
62     <dependency>
63     <groupId>log4j</groupId>
64     <artifactId>log4j</artifactId>
```

```
62         <version>${log4j.version}
</version>
63     </dependency>
64 </dependencies>
65 </dependencyManagement>
66
67 <build>
68     <finalName>firstMainCloud</finalName>
69     <resources>
70         <resource>
71
72         <directory>src/main/resources</directory>
73         <filtering>true</filtering>
74     </resource>
75 </resources>
76 <plugins>
77     <plugin>
78
79     <groupId>org.apache.maven.plugins</groupId>
80     <artifactId>maven-resources-
plugin</artifactId>
81     <configuration>
82         <delimiters>
83             <delimiter>$</delimiter>
84         </delimiters>
85     </configuration>
86 </plugin>
87 </plugins>
```

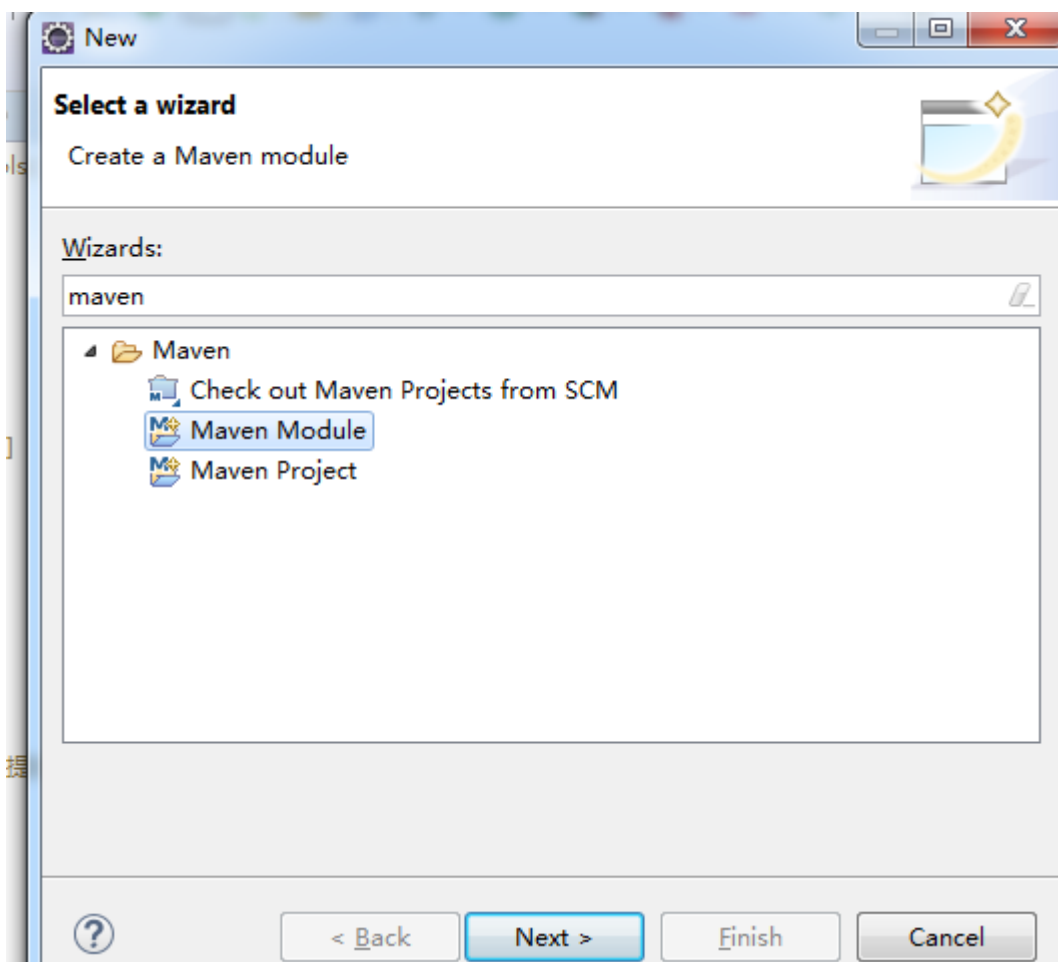
```
86     </build>
```

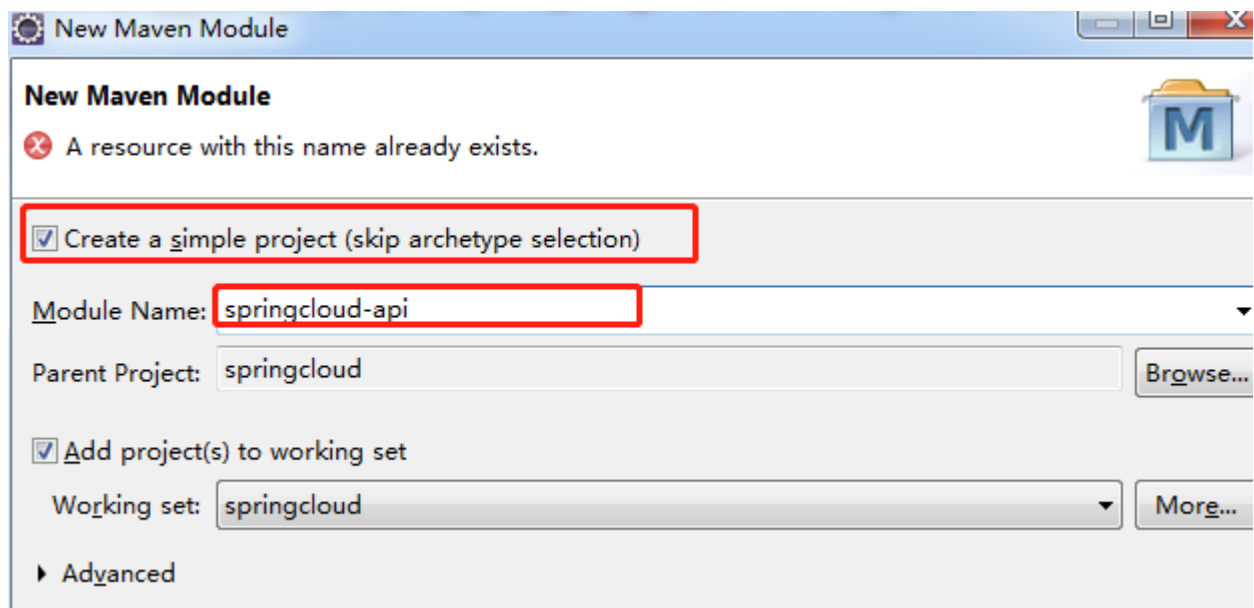
```
87 </project>
```

2. 创建子工程

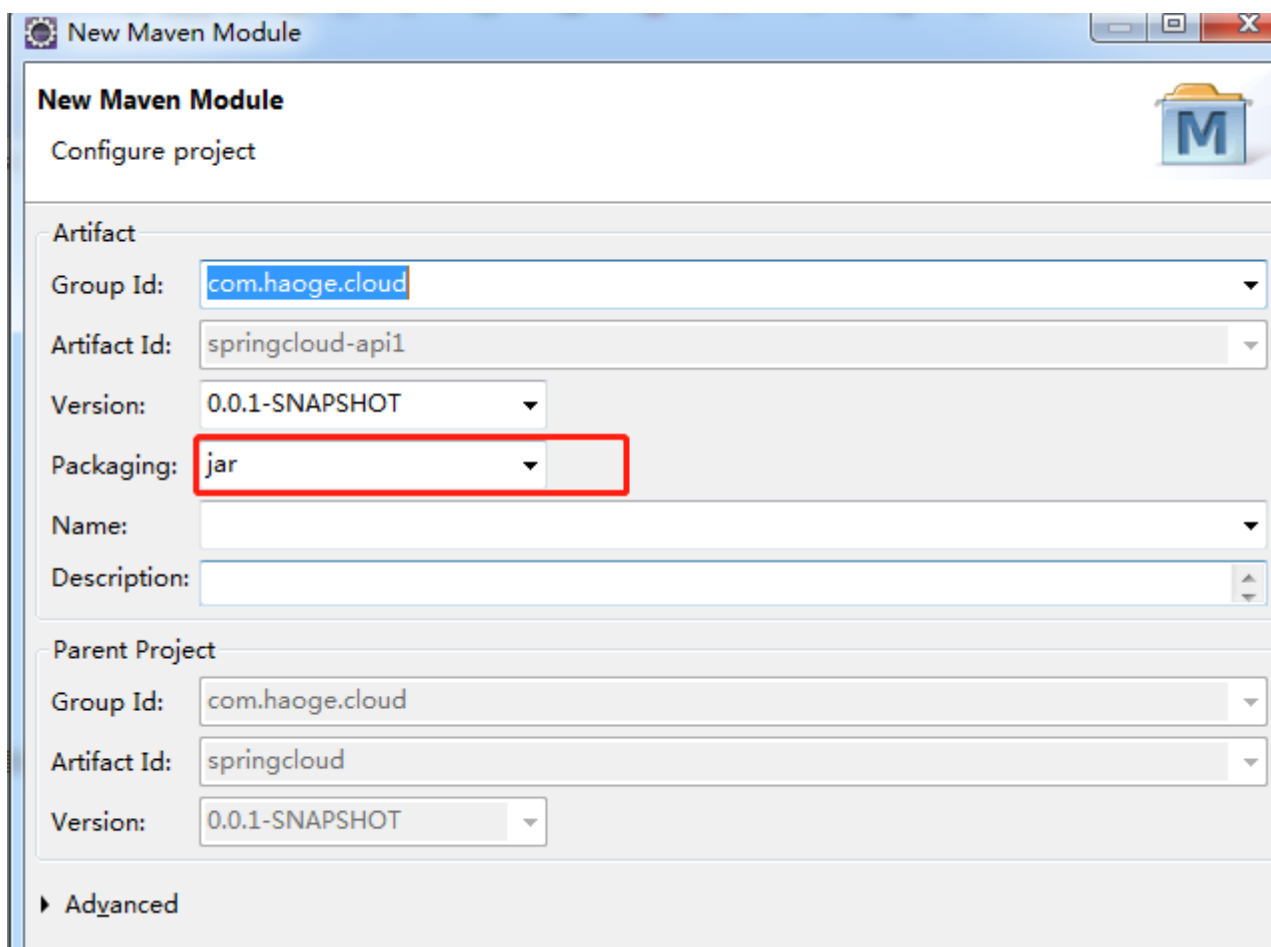
1. 创建maven子模块

在springcloud上右键创建maven子工程springcloud-api





2. 打包方式选择jar



3. POM依赖

```
1 <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">
2   <modelVersion>4.0.0</modelVersion>
3   <parent>
4     <groupId>com.haoge.cloud</groupId>
5     <artifactId>springcloud</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7   </parent>
8   <artifactId>springcloud-api</artifactId>
9 </project>
```

4. 编写公用实体类


```
1 @SuppressWarnings("serial")
2 public class Dept implements Serializable{//必须实现序列化
3
4     private Long deptno;//主键
5     private String dname;//部门名称
6     private String db_source;// 来自那个数据库，因为微服务架构可以一个服务对应一个数据库，同一个信息被存储到不同数据库
7
8 }
```

3. 创建提供者springcloud-provider-dept-8001

任然是在springcloud项目上右键选择 new maven module,打包方式选择jar springcloud-provider-dept-8001

1. POM依赖

```
1 <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">
2     <modelVersion>4.0.0</modelVersion>
3     <parent>
```

```
4     <groupId>com.haoge.cloud</groupId>
5     <artifactId>springcloud</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7 </parent>
8     <artifactId>springcloud-provider-dept-
8001</artifactId>
9     <dependencies>
10         <!-- 引入自己定义的api通用包，可以使用Dept
部门Entity -->
11         <dependency>
12             <groupId>com.haoge.cloud</groupId>
13             <artifactId>springcloud-
api</artifactId>
14             <version>${project.version}
</version>
15         </dependency>
16         <!-- actuator主管监控信息完善 -->
17         <dependency>
18
19         <groupId>org.springframework.boot</groupId>
20         <artifactId>spring-boot-starter-
actuator</artifactId>
21         </dependency>
22         <!-- 将微服务provider侧注册进eureka eureka
后面没有server表示是eureka的客户端-->
23         <dependency>
24
25         <groupId>org.springframework.cloud</groupId>
```

```
24         <artifactId>spring-cloud-starter-  
eureka</artifactId>  
25     </dependency>  
26     <dependency>  
27  
    <groupId>org.springframework.cloud</groupId>  
28         <artifactId>spring-cloud-starter-  
config</artifactId>  
29     </dependency>  
30  
31  
32     <dependency>  
33         <groupId>junit</groupId>  
34         <artifactId>junit</artifactId>  
35     </dependency>  
36     <dependency>  
37         <groupId>mysql</groupId>  
38         <artifactId>mysql-connector-  
java</artifactId>  
39     </dependency>  
40     <dependency>  
41         <groupId>com.alibaba</groupId>  
42         <artifactId>druid</artifactId>  
43     </dependency>  
44     <dependency>  
45         <groupId>ch.qos.logback</groupId>  
46         <artifactId>logback-  
core</artifactId>
```

```
47         </dependency>
48         <dependency>
49
50             <groupId>org.mybatis.spring.boot</groupId>
51             <artifactId>mybatis-spring-boot-
52 starter</artifactId>
53         </dependency>
54         <dependency>
55
56             <groupId>org.springframework.boot</groupId>
57             <artifactId>spring-boot-starter-
58 jetty</artifactId>
59         </dependency>
60         <dependency>
61
62             <groupId>org.springframework.boot</groupId>
63             <artifactId>spring-boot-starter-
64 web</artifactId>
65         </dependency>
66         <dependency>
67
68             <groupId>org.springframework.boot</groupId>
69             <artifactId>spring-boot-starter-
70 test</artifactId>
71         </dependency>
72         <!-- 修改后立即生效，热部署 -->
73         <dependency>
```

```

66     <groupId>org.springframework</groupId>
67     <artifactId>springloaded</artifactId>
68         </dependency>
69         <dependency>
70
71     <groupId>org.springframework.boot</groupId>
72         <artifactId>spring-boot-
73         devtools</artifactId>
74         </dependency>
75     </dependencies>
76 </project>

```

2. yaml文件

```

1  server:
2      port: 8001      # 项目的端口号
3
4  mybatis:
5      config-location:
6          classpath:mybatis/mybatis.cfg.xml      #
7          mybatis配置文件所在路径
8      type-aliases-package:
9          com.atguigu.springcloud.entities      # 所有Entity
10         别名类所在包
11      mapper-locations:

```

```
8   - classpath:mybatis/mapper/**/*.xml
      # mapper映射文件
9
10  spring:
11    profiles:
12      active:
13        - dev
14    application:
15      name: springcloud-dept # 当前微服务向外暴露的
      微服务名称
16    datasource:
17      type: com.alibaba.druid.pool.DruidDataSource
      # 当前数据源操作类型
18      driver-class-name: org.gjt.mm.mysql.Driver
      # mysql驱动包
19      url: jdbc:mysql://localhost:3306/cloudDB01
      # 数据库名称
20      username: root
21      password: 123456
22      dbcp2:
23        min-idle: 5
      # 数据库连接池的最小维持连接数
24        initial-size: 5
      # 初始化连接数
25        max-total: 5
      # 最大连接数
26        max-wait-millis: 200
```

3. 项目结构

```

└─ springcloud-provider-dept-8001 [boot] [devtools] [spring]
  └─ src/main/java
    └─ com.haoge.cloud
      └─ controller
        └─ DeptController.java
      └─ dao
        └─ DeptDao.java
      └─ service
        └─ impl
          └─ DeptServiceImpl.java
          └─ DeptService.java
        └─ DeptProvider8001_App.java
  └─ src/main/resources
    └─ mybatis
      └─ mapper
        └─ DeptMapper.xml
        └─ mybatis.cfg.xml
      └─ application.yml
  └─ src/test/java
  └─ src/test/resources
  └─ JRE System Library [JavaSE-1.8]
  └─ Maven Dependencies
  └─ src
  └─ target
  └─ pom.xml

```

4. 主启动程序

```
1 @SpringBootApplication
2 public class DeptProvider8001_App {
3     public static void main(String[] args) {
4
5         SpringApplication.run(DepartmentProvider8001_App.class,
6             args);
7     }
8 }
```

5. DeptController

```
1 @RestController
2 public class DeptController {
3     @Autowired
4     private DeptService service;
5
6     /**
7      * 增加部门的方法
8      * @param dept
9      * @return
10     */
11
12     @RequestMapping(value="/dept/add",method=Request
13         Method.POST)
14     public boolean add(@RequestBody Dept dept) {
15         return service.add(dept);
16     }
17 }
```



```

16      * 根据Id查询部门
17      * @param id
18      * @return
19      */
20
    @RequestMapping(value="/dept/get/{id}",method=RequestMethod.GET)
21      public Dept get(@PathVariable Long id) {
22          return service.get(id);
23      }
24      /**
25       * 查询部门列表
26       * @return
27       */
28
    @RequestMapping(value="/dept/list",method=RequestMethod.GET)
29      public List<Dept> get() {
30          System.out.println(8001);
31          return service.list();
32      }
33 }

```

6. DeptService

```
1 public interface DeptService
2 {
3     public boolean add(Dept dept);
4
5     public Dept get(Long id);
6
7     public List<Dept> list();
8 }
```

7. DeptServiceImpl

```
1 @Service
2 public class DeptServiceImpl implements
   DeptService {
3     @Autowired
4     private DeptDao dao;
5
6     public boolean add(Dept dept) {
7         // TODO Auto-generated method stub
8         return dao.addDept(dept);
9     }
10
11     @Override
12     public Dept get(Long id) {
13         // TODO Auto-generated method stub
14         return dao.findById(id);
15     }
16 }
```

```
17     @Override
18     public List<Dept> list() {
19         // TODO Auto-generated method stub
20         return dao.findAll();
21     }
22 }
23
```

8. DeptDao

```
1 @Mapper
2 public interface DeptDao
3 {
4     public boolean addDept(Dept dept);
5
6     public Dept findById(Long id);
7
8     public List<Dept> findAll();
9 }
```

9. DeptMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD
  Mapper 3.0//EN"
3 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4
5 <mapper namespace="com.haoge.cloud.dao.DeptDao">
```

```
6
7     <select id="findById"
resultType="com.haoge.cloud.entities.Dept"
parameterType="Long">
8         select deptno,dname,db_source from dept
where deptno=#{deptno};
9     </select>
10    <select id="findAll"
resultType="com.haoge.cloud.entities.Dept">
11        select deptno,dname,db_source from dept;
12    </select>
13    <insert id="addDept"
parameterType="com.haoge.cloud.entities.Dept">
14        INSERT INTO dept(dname,db_source)
VALUES("#{dname}",DATABASE());
15    </insert>
16
17 </mapper>
```

10 mybatis.cfg.xml (可以省略)

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5
6 <configuration>
7
8     <!-- <settings>
9         <setting name="cacheEnabled"
10         value="true" />二级缓存开启
11     </settings> -->
12 </configuration>
```

11. 测试

<http://localhost:8001/dept/get/2>

<http://localhost:8001/dept/list>

4. 创建消费者springcloud-consumer-dept-80

1. 创建maven子项目

任然是在springcloud项目上右键选择 new maven module, 打包方式选择jar springcloud-consumer-dept-80

2. POM依赖

```
1 <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">
2   <modelVersion>4.0.0</modelVersion>
3   <parent>
4     <groupId>com.haoge.cloud</groupId>
5     <artifactId>springcloud</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7   </parent>
8   <artifactId>springcloud-consumer-dept-
  80</artifactId>
9
10  <dependencies>
11    <dependency>
12      <groupId>com.haoge.cloud</groupId>
13      <artifactId>springcloud-
  api</artifactId>
14      <version>${project.version}
  </version>
15    </dependency>
16    <!-- Ribbon相关 , 负载均衡-->
17    <dependency>
18
  <groupId>org.springframework.cloud</groupId>
```

```
19         <artifactId>spring-cloud-starter-  
eureka</artifactId>  
20     </dependency>  
21     <dependency>  
22  
    <groupId>org.springframework.cloud</groupId>  
23         <artifactId>spring-cloud-starter-  
ribbon</artifactId>  
24     </dependency>  
25     <dependency>  
26  
    <groupId>org.springframework.cloud</groupId>  
27         <artifactId>spring-cloud-starter-  
config</artifactId>  
28     </dependency>  
29     <dependency>  
30  
    <groupId>org.springframework.boot</groupId>  
31         <artifactId>spring-boot-starter-  
web</artifactId>  
32     </dependency>  
33     <!-- 修改后立即生效，热部署 -->  
34     <dependency>  
35  
    <groupId>org.springframework</groupId>  
36  
    <artifactId>springloaded</artifactId>  
37     </dependency>
```

```
38         <dependency>
39
40         <groupId>org.springframework.boot</groupId>
41         <artifactId>spring-boot-
42         devtools</artifactId>
43     </dependency>
44 </dependencies>
45 </project>
```

3. yaml文件

```
1 server:
2   port: 80
```

4. 主程序

```
1 @SpringBootApplication
2 public class DeptConsumer80_App {
3
4     public static void main(String[] args) {
5
6         SpringApplication.run(DeptConsumer80_App.class,
7         args);
8     }
```

5. Controller

```
1 @RestController
2 public class DeptController_Consumer {
3     private static final String REST_URL_PREFIX
4     = "http://localhost:8001";
5
6     // private static final String REST_URL_PREFIX
7     = "http://firstmaincloud-dept";
8
9     @Autowired
10    private RestTemplate template;
11
12    /**
13     * 使用 使用restTemplate访问restful接口非常的
14     * 简单粗暴无脑。 (url, requestMap,
15     * ResponseBean.class)这三个参数分别代表 REST
16     * 请求地址、请求参数、HTTP响应转换被转换成的对象类型。
17     */
18    @RequestMapping(value =
19    "/consumer/dept/add")
20    public boolean add(Department dept) {
21        return
22        template.postForObject(REST_URL_PREFIX +
23        "/dept/add", dept, boolean.class);
24    }
25
26    /**
27     * 根据ID查询部门的方法
28     *
29     * @param dept
30     * @return
31     */
32 }
```

```
22     @RequestMapping(value =
23         "/consumer/dept/get/{id}")
24     public Dept get(@PathVariable("id") Long id)
25     {
26         return
27         template.getForObject(REST_URL_PREFIX +
28             "/dept/get/" + id, Dept.class);
29     }
30     /**
31     * 查询列表的方法
32     *
33     * @param dept
34     * @return
35     */
36     @SuppressWarnings("unchecked")
37     @RequestMapping(value =
38         "/consumer/dept/list")
39     public List<Dept> list() {
40
41         return
42         template.getForObject(REST_URL_PREFIX +
43             "/dept/list", List.class);
44     }
45     // 测试@EnableDiscoveryClient,消费端可以调用服务发现
46     @RequestMapping(value =
47         "/consumer/dept/discovery")
48     public Object discovery() {
```

```
41         return
    template.getForObject(REST_URL_PREFIX +
        "/dept/discovery", Object.class);
42     }
43 }
```

RestTemplate

提供了多种便捷访问远程HTTP服务的方法，是一种简单便捷的访问restful的服务模板类，是spring提供的用于访问rest服务的客户端模板工具集

6. 配置类

```
1 @Configuration //@Configuration配置    ConfigBean =
   applicationContext.xml
2 public class ConfigBean {
3     @Bean
4     public RestTemplate getRestTemplate() {
5         return new RestTemplate();
6     }
7 }
```

ConfigBean相当于我们以前写的applicationContext.xml配置文件，getRestTemplate()方法相当于我们以前在xml文件中定义的bean

7.测试

<http://localhost/consumer/dept/get/1>

<http://localhost/consumer/dept/list>

二、Spring Eureka

1.简介

1.Netflix在设计Eureka的时候遵循的是AP原则

C : Consistency (强一致性)

A : Availability (可用性)

P : Partition tolerance(分区容错性)

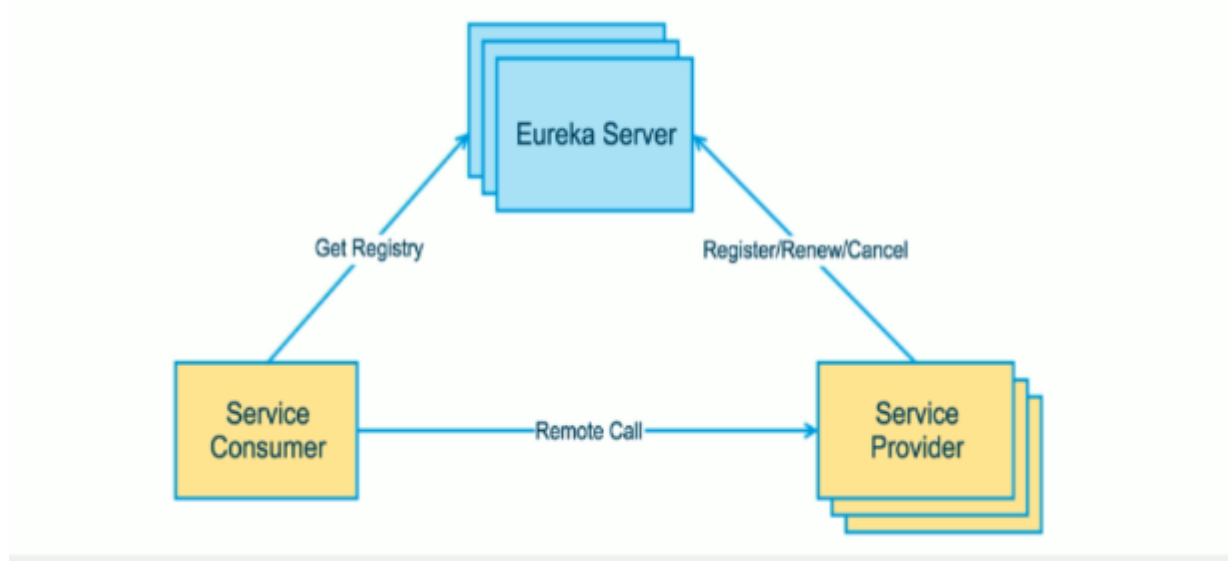
2.Eureka是一个基于rest的服务，用于定位服务，以实现云端中间层服务发现和故障转移。功能类似于dubbo的注册中心，比如zookeeper.SpringCloud主要使用Eureka来实现服务注册和发现功能。

3.Eureka采用了CS的设计架构。Eureka Server作为服务注册功能的服务器，它是服务注册中心。

而系统中的其他微服务，使用Eureka的客户端连接到Eureka的客户端连接到Eureka Server并维持心跳连接。

4.EurekaClient是一个Java客户端，用于简化Eureka Server的交互。客户端同时具备一个内置的，使用轮询负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳（默认周期为30秒），如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认为90秒）

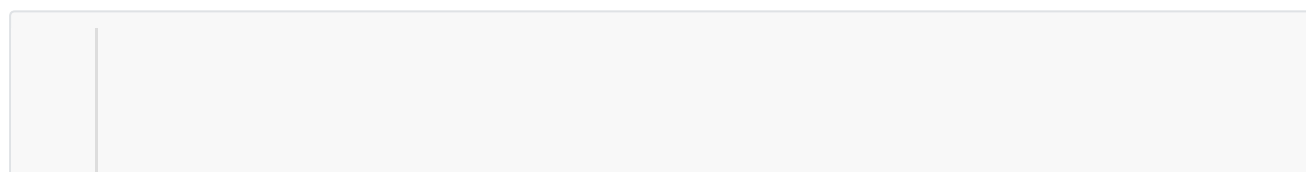
5.架构图



2 Eureka Server 搭建

1、 仍然是在springcloud项目上右键选择 new maven module,打包方式选择jar ，项目名称为 springcloud-eureka-server-7001

2、 POM依赖



```
1 <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">
2   <modelVersion>4.0.0</modelVersion>
3   <parent>
4     <groupId>com.haoge.cloud</groupId>
5     <artifactId>springcloud</artifactId>
6     <version>0.0.1-SNAPSHOT</version>
7   </parent>
8   <artifactId>springcloud-eureka-server-
  7001</artifactId>
9   <dependencies>
10     <!--eureka-server服务端 -->
11     <dependency>
12
13       <groupId>org.springframework.cloud</groupId>
14       <artifactId>spring-cloud-starter-
  eureka-server</artifactId>
15     </dependency>
16     <!-- 修改后立即生效，热部署 -->
17     <dependency>
18
19       <groupId>org.springframework</groupId>
```

```
18     <artifactId>springloaded</artifactId>
19         </dependency>
20     <dependency>
21
22     <groupId>org.springframework.boot</groupId>
23         <artifactId>spring-boot-
24 devtools</artifactId>
25         </dependency>
26     </dependencies>
27 </project>
```

3、yaml配置文件

```
1 server:
2   port: 7001
3
4 eureka:
5   instance:
6     hostname: localhost #eureka服务端的实例名称(单机版)
7   client:
8     register-with-eureka: false      #false表示不向注册中心注册自己。
9     fetch-registry: false           #false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务
10   service-url:
11     defaultZone:
http://${eureka.instance.hostname}:${server.port}/eureka/      #设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址（单机），
```

eureka.client.service-url.defaultZone 对应的属性值意义：这个地址是Eureka服务端暴露给外部的地址，外部的微服务如果想注册进这个Eureka Server中，则响应的写这个地址。

4、主启动类


```

1 @SpringBootApplication
2 @EnableEurekaServer// EurekaServer服务器端启动类,接受其它微服务注册进来
3 public class EurekaServer7001_App {
4
5     public static void main(String[] args) {
6
7         SpringApplication.run(EurekaServer7001_App.class,
8             args);
9     }
10 }

```

5、测试，还没有服务注册

<http://localhost:7001/>

The screenshot shows the Spring Eureka Server web interface. The header includes the Spring Eureka logo and navigation links for HOME and LAST 1000 SINCE START. The main content area is divided into two sections: System Status and DS Replicas.

System Status

Environment	test	Current time	2018-07-13T13:51:28 +C
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

3. 服务注册

将8001项目提供的服务注册进Eureka Server中

1.8001项目添加POM 依赖

```
1 <!-- 将微服务provider侧注册进eureka eureka后面没有
   server表示是eureka的客户端-->
2     <dependency>
3
4     <groupId>org.springframework.cloud</groupId>
5     <artifactId>spring-cloud-starter-
   eureka</artifactId>
6     </dependency>
7     <dependency>
8     <groupId>org.springframework.cloud</groupId>
9     <artifactId>spring-cloud-starter-
   config</artifactId>
10    </dependency>
```

2、yaml添加配置

```
1 eureka:
2   client: #客户端注册进eureka服务列表内
3   service-url:
4     defaultZone: http://localhost:7001/eureka
   #单机版
```

服务提供者eureka.client.service-url.defaultZone对应的属性值意义：表示这个微服务要注册进<http://localhost:7001/eureka>这个地址中。

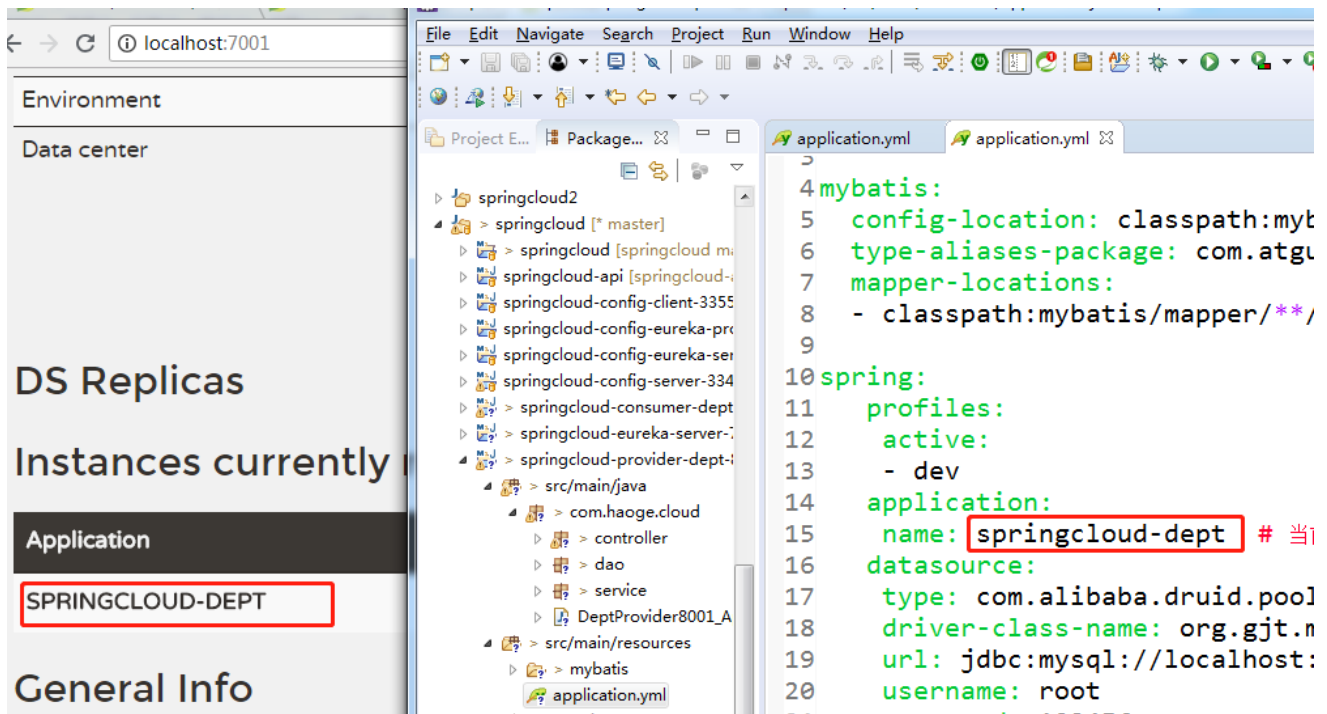
3、主启动类添加注解@EnableEurekaClient

```
1 @SpringBootApplication
2 @EnableEurekaClient//这个注解的意思是本服务启动后会
   自动注册进eureka服务端中
3 public class DeptProvider8001_App {
4     public static void main(String[] args) {
5
6         SpringApplication.run(DepProvider8001_App.class,
7             args);
8     }
9 }
```

4、测试

启动7001服务端项目，再启动8001项目。

<http://localhost:7001/>



图示为微服务对外暴露的服务名称，是我们在8001项目配置文件中写的spring.application.name对应的属性名称。Eureka自动帮我们转为大写。

5、8001服务配置补充

```

1 eureka:
2   client: #客户端注册进eureka服务列表内
3     service-url:
4       defaultZone: http://localhost:7001/eureka
#单机版
5 #     defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka7
002.com:7002/eureka/,http://eureka7003.com:7003/e
ureka/
6   instance:
7     instance-id: deptService8001      #对当前服务起
的别名
8     prefer-ip-address: true      #我们在eureka服务
端查看服务名称的时候：访问路径可以显示IP地址 （使用IP
进行服务注册）

```

效果

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-DEPT	n/a (1)	(1)	UP (1) - deptService8001

General Info	
Name	Value
total-avail-memory	366mb
environment	test
num-of-cpus	4
current-memory-usage	210mb (57%)
server-uptime	00:01
registered-replicas	
unavailable-replicas	
available-replicas	

92.168.9.19:8001/info

6、8001服务显示info信息

6.1 pom依赖

```
1 <!-- actuator主管监控信息完善 -->
2     <dependency>
3
4     <groupId>org.springframework.boot</groupId>
5     <artifactId>spring-boot-starter-actuator</artifactId>
6     </dependency>
```

6.2 主工程springcloudPOM添加依赖

```
1 <build>
2     <finalName>firstMainCloud</finalName>
3     <resources>
4         <resource>
5
6         <directory>src/main/resources</directory>
7         <filtering>true</filtering>
8     </resource>
9     </resources>
10    <plugins>
11        <plugin>
12
13        <groupId>org.apache.maven.plugins</groupId>
```

```

12         <artifactId>maven-resources-
plugin</artifactId>
13         <configuration>
14             <delimiters>
15                 <delimiter>${</delimiter>
16             </delimiters>
17         </configuration>
18     </plugin>
19 </plugins>
20 </build>

```

作用：允许yaml配置文件读取POM文件中的值。（需要以\$开头，并以\$结尾）。如下所示

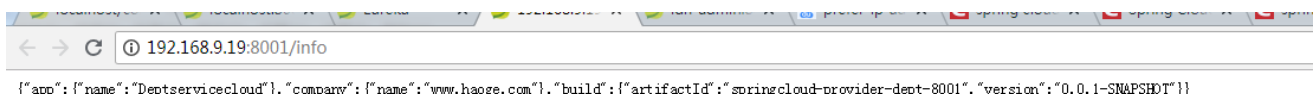
6.3 yaml配置

```

1 info:
2     app.name: Deptservicecloud
3     company.name: www.haoge.com
4     build.artifactId: ${project.artifactId}
5     build.version: ${project.version}

```

效果：

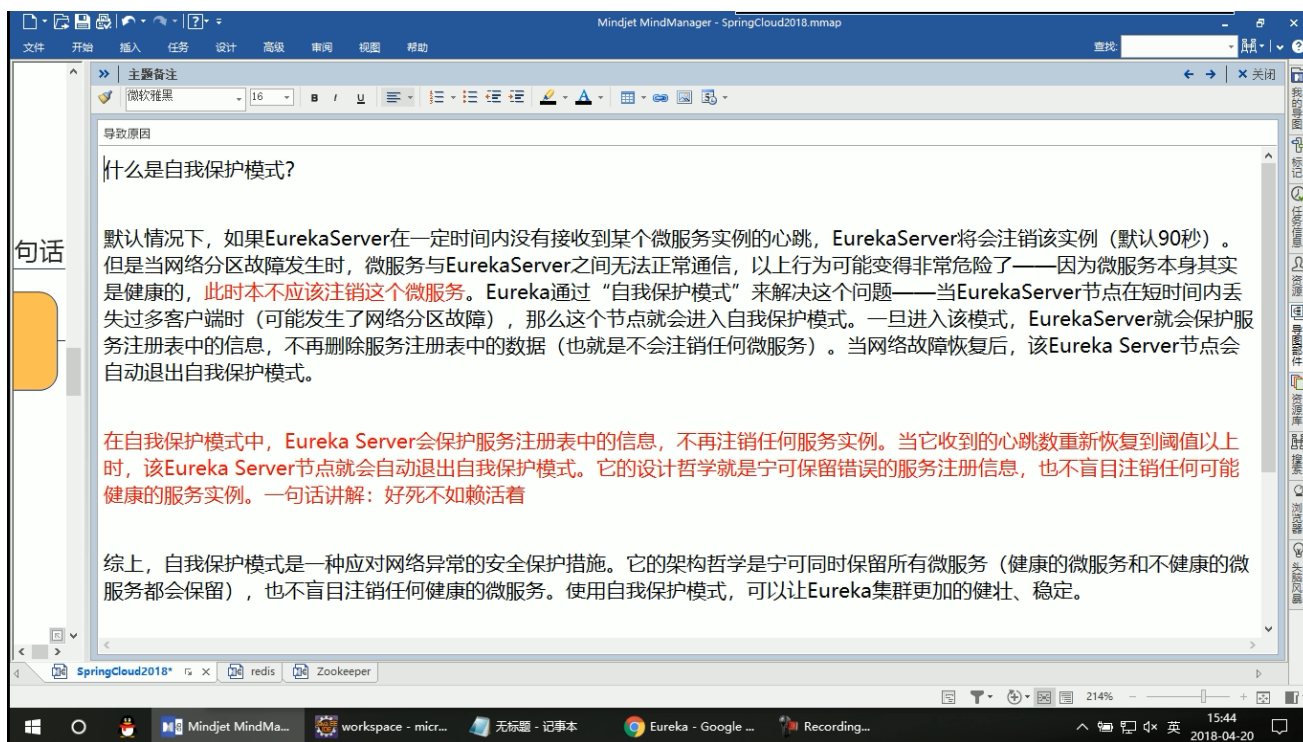


```

{"app":{"name":"Deptservicecloud"},"company":{"name":"www.haoge.com"},"build":{"artifactId":"springcloud-provider-dept-8001","version":"0.0.1-SNAPSHOT"}}

```

4 Eureka的自我保护机制



spring cloud中可以使用

`enable.server.enable-self-preservation=false` 禁用自我保护模式

5.服务发现

1、@EnableDiscoveryClient//允许服务发现的注解

8001项目主启动类加注解：@EnableDiscoveryClient


```

1 @SpringBootApplication
2 @EnableEurekaClient//这个注解的意思是本服务启动后会
   自动注册进eureka服务端中
3 @EnableDiscoveryClient//允许服务发现的注解
4 public class DeptProvider8001_App {
5     public static void main(String[] args) {
6
7         SpringApplication.run(DepartmentProvider8001_App.class,
8             args);
9     }
10 }

```

2、controller添加测试代码

```

1     //服务发现模块
2     @Autowired
3     private DiscoveryClient client;
4
5     @RequestMapping(value = "/dept/discovery",
6         method = RequestMethod.GET)
7     public Object discovery()
8     {
9         List<String> list =
10         client.getServices();//查询eureka中的服务都有哪些
11         System.out.println("*****" + list);
12
13         List<ServiceInstance> srvList =
14         client.getInstances("MICROSERVICECLOUD-DEPT");
15     }
16 }

```

```
12         for (ServiceInstance element : srvList)
13     {
14         System.out.println(element.getServiceId() + "\t"
15         + element.getHost() + "\t" + element.getPort() +
16         "\t"
17         + element.getUri());
18     }
19     return this.client;
20 }
```

3、测试

<http://localhost:8001/dept/discovery>

4、80项目controller测试代码

```
1 // 测试@EnableDiscoveryClient,消费端可以调用服务发现
2 @RequestMapping(value =
3     "/consumer/dept/discovery")
4     public Object discovery() {
5         return
6         template.getForObject(REST_URL_PREFIX +
7         "/dept/discovery", Object.class);
8     }
```

测试：<http://localhost/consumer/dept/discovery>

6. Eureka集群配置

1、仿照7001项目创建两个Eureka Server项目，端口分别为7002,7003

2、7001Server配置文件

```
1 server:
2   port: 7001
3
4 eureka:
5   instance:
6     hostname: eureka7001.com #eureka服务端的实例名称
7   #   hostname: localhost #eureka服务端的实例名称 (单机版)
8   client:
9     register-with-eureka: false      #false表示不向注册中心注册自己。
10    fetch-registry: false      #false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务
11    service-url:
12    #    defaultZone:
13      http://${eureka.instance.hostname}:${server.port}/eureka/      #设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址（单机）。
14    defaultZone:
15      http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

7001Server配置文件

```
1 server:
2   port: 7002
3
4 eureka:
5   instance:
6     hostname: eureka7002.com #eureka服务端的实例名称
7   #   hostname: localhost #eureka服务端的实例名称 (单机版)
8   client:
9     register-with-eureka: false      #false表示不向注册中心注册自己。
10    fetch-registry: false      #false表示自己端就是注册中心，我的职责就是维护服务实例，并不需要去检索服务
11    service-url:
12    #    defaultZone:
13      http://${eureka.instance.hostname}:${server.port}/eureka/      #设置与Eureka Server交互的地址查询服务和注册服务都需要依赖这个地址（单机）。
14    defaultZone:
15      http://eureka7001.com:7001/eureka/,http://eureka7003.com:7003/eureka/
```

7003Server配置文件

```
1 server:
2   port: 7003
3
4 eureka:
5   instance:
6     hostname: eureka7003.com #eureka服务端的实例名称
7   #   hostname: localhost #eureka服务端的实例名称
8   #   (单机版)
9   client:
10    register-with-eureka: false      #false表示不
11    #   向注册中心注册自己。
12    fetch-registry: false          #false表示自己端就是
13    #   注册中心，我的职责就是维护服务实例，并不需要去检索服务
14    service-url:
15    #   defaultZone:
16    #   http://${eureka.instance.hostname}:${server.port}
17    #   /eureka/      #设置与Eureka Server交互的地址查询
18    #   服务和注册服务都需要依赖这个地址（单机）。
19    #   defaultZone:
20    #   http://eureka7002.com:7002/eureka/,http://eureka
21    #   7001.com:7001/eureka/
```

8001服务提供者yaml配置文件

```
1 server:
2   port: 8001      # 项目的端口号
```

```
3
4 mybatis:
5   config-location:
6     classpath:mybatis/mybatis.cfg.xml      #
7     mybatis配置文件所在路径
8   type-aliases-package:
9     com.atguigu.springcloud.entities      # 所有Entity
10    别名类所在包
11   mapper-locations:
12     - classpath:mybatis/mapper/**/*.xml
13       # mapper映射文件
14
15 spring:
16   profiles:
17     active:
18       - dev
19   application:
20     name: springcloud-dept  # 当前微服务向外暴露的
21    微服务名称
22   datasource:
23     type: com.alibaba.druid.pool.DruidDataSource
24       # 当前数据源操作类型
25     driver-class-name: org.gjt.mm.mysql.Driver
26       # mysql驱动包
27     url: jdbc:mysql://localhost:3306/cloudDB01
28       # 数据库名称
29     username: root
30     password: 123456
```

```
22     dbcp2:
23         min-idle: 5
24             # 数据库连接池的最小维持连接数
25         initial-size: 5
26             # 初始化连接数
27         max-total: 5
28             # 最大连接数
29         max-wait-millis: 200
30             # 等待连接获取的最大超时时间
31 eureka:
32     client: #客户端注册进eureka服务列表内
33         service-url:
34             # defaultZone: http://localhost:7001/eureka
35             #单机版
36         defaultZone:
37             http://eureka7001.com:7001/eureka/,http://eureka
38             7002.com:7002/eureka/,http://eureka7003.com:7003
39             /eureka/
40     instance:
41         instance-id: deptService8001 #对当前服务
42         起的别名
43         prefer-ip-address: true #我们在eureka服务
44         端查看服务名称的时候：访问路径可以显示IP地址
45     info:
46         app.name: Deptservicecloud
47         company.name: www.haoge.com
48         build.artifactId: $project.artifactId$
49         build.version: $project.version$
```

3、配置host文件

127.0.0.1 eureka7001.com 127.0.0.1 eureka7002.com

127.0.0.1 eureka7003.com

4、测试。分别启动7001,7002,7003 , 8001项目

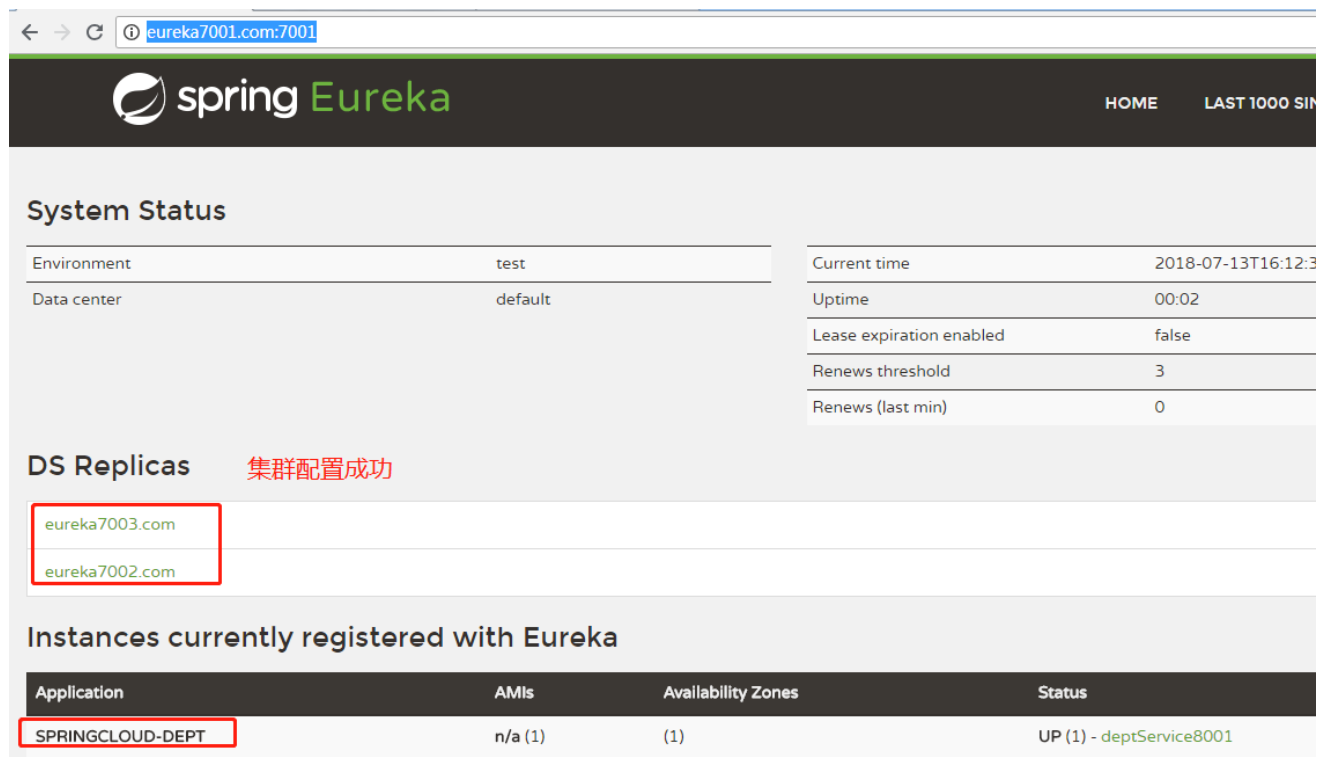
分别访问

<http://eureka7001.com:7001/>

<http://eureka7002.com:7002/>

<http://eureka7003.com:7003/>

效果



The screenshot shows the Spring Eureka web interface. The browser address bar displays `http://eureka7001.com:7001/`. The page header includes the Spring Eureka logo and navigation links for HOME and LAST 1000 SIGNED. The main content area is divided into two sections: System Status and DS Replicas.

System Status

System Status	
Environment	test
Data center	default
Current time	2018-07-13T16:12:3
Uptime	00:02
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

DS Replicas 集群配置成功

DS Replicas
eureka7003.com
eureka7002.com

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-DEPT	n/a (1)	(1)	UP (1) - deptService8001

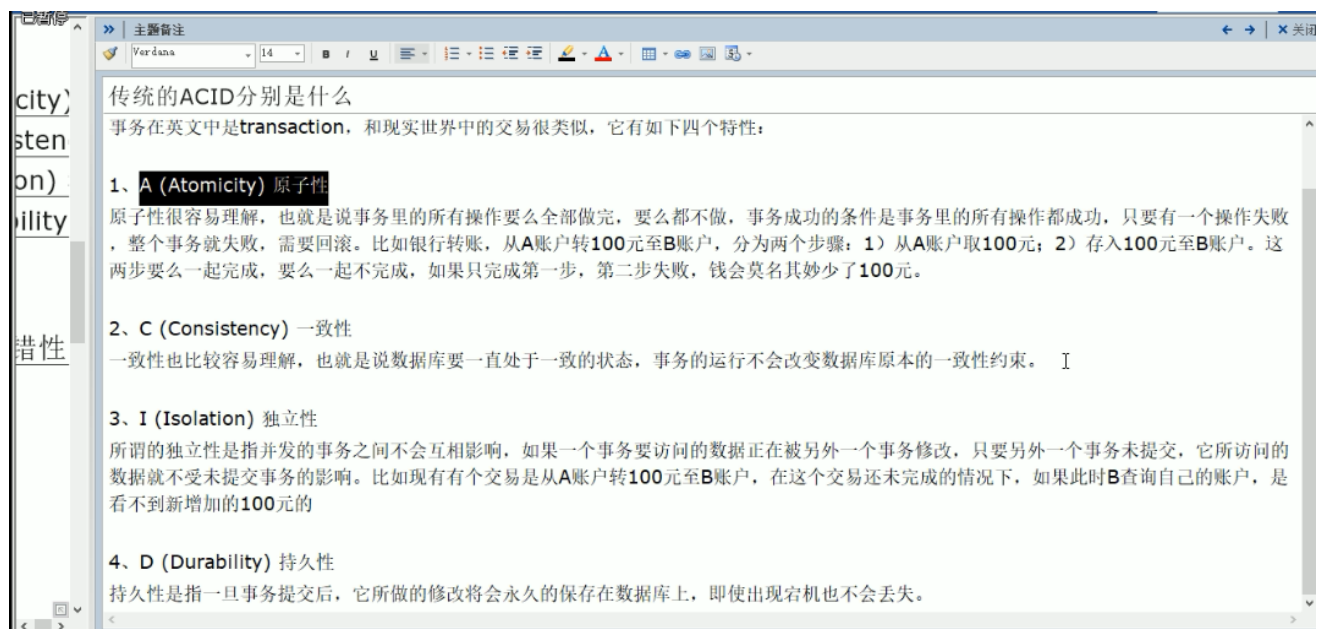
7. Eureka和zookeeper的比较

Zookeeper保证的是CP原则

Eureka保证的是AP原则

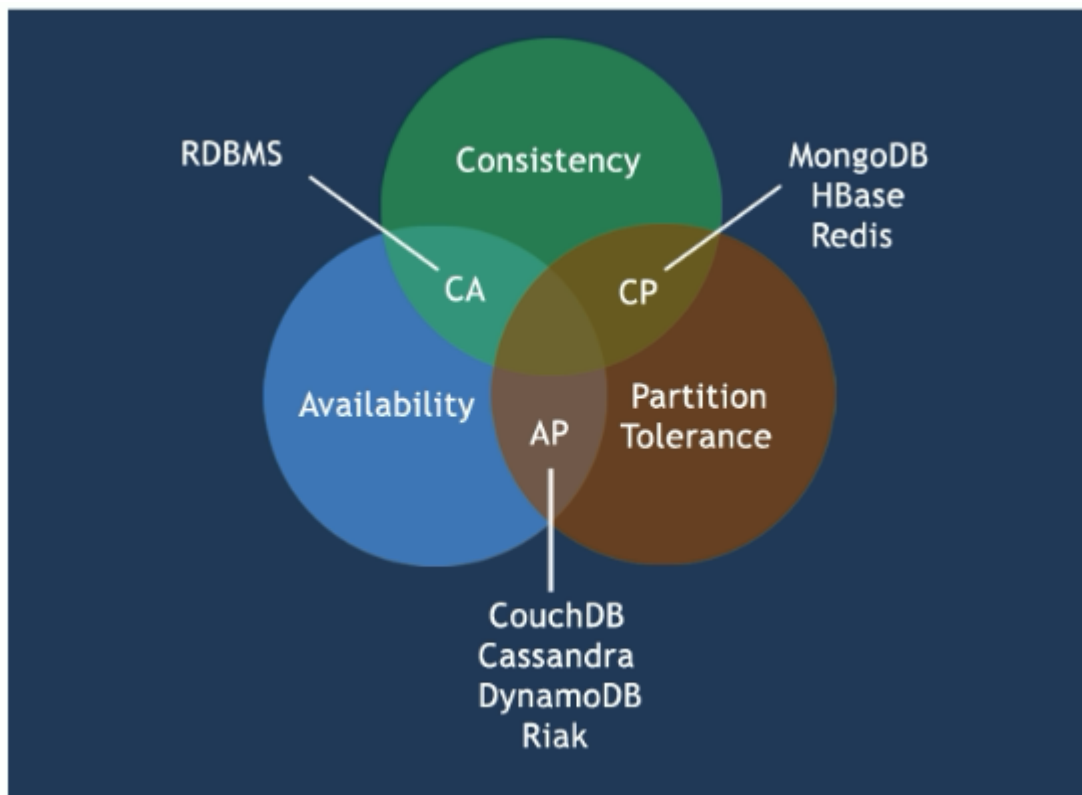
在电商网站的高并发访问情况下，比如双11当天。高可用性比强一致性更加重要。

1.数据库ACID介绍



2、CAP理论介绍

传统的数据库如mysql满足的是CA原则



最多只能同时较好的满足两个。

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。

CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。

I

AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。

由于在当前的网络硬件肯定会出现延迟丢包等问题，所以分区容错性，即P使我们必须要实现的。所以我们必须在A和C中间进行权衡。

4.1 Zookeeper保证CP

当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s，且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是本能容忍的。

4.2 Eureka保证AP

Eureka看明白了这一点，因此在设计时就优先保证可用性。**Eureka各个节点都是平等的**，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka注册或时如果发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka还有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

1. Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务
2. Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)
3. 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

因此，**Eureka**可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像**zookeeper**那样使整个注册服务瘫痪。