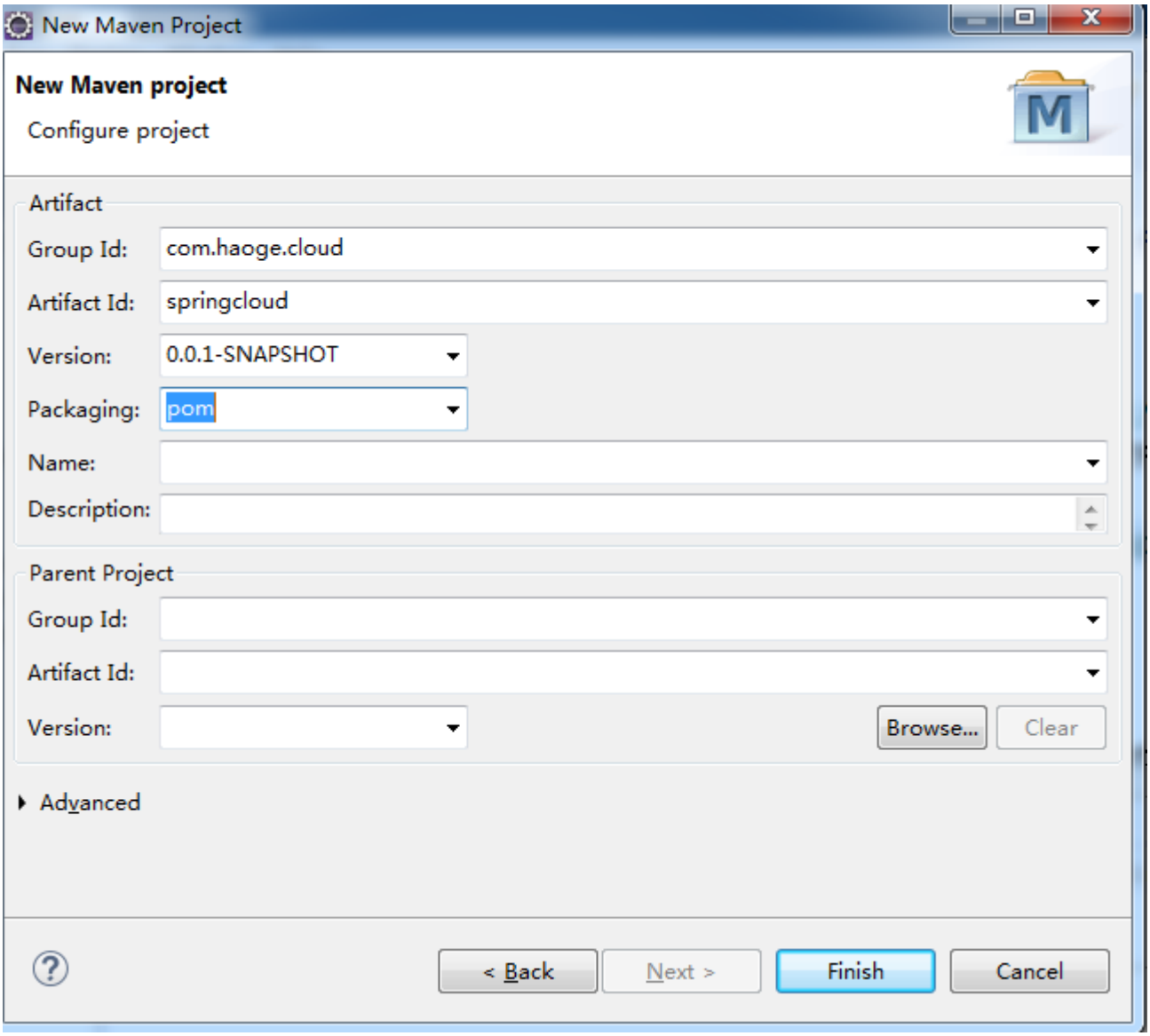
一、环境准备

springboot+mybatis整合

1.创建父工程



1. POM依赖

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/
4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.haoge.cloud</groupId>
  <artifactId>springcloud</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.sourc
e>

<maven.compiler.target>1.8</maven.compiler.targe
t>
        <junit.version>4.12</junit.version>
        <log4j.version>1.2.17</log4j.version>
        <lombok.version>1.16.18</lombok.version>
    </properties>
```

```
    <!-- 父类工程的管理 -->
    <dependencyManagement>
        <dependencies>
            <dependency>
<groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-
dependencies</artifactId>
                <version>Dalston.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
<groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-
dependencies</artifactId>
                <version>1.5.9.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-
java</artifactId>
                <version>5.0.4</version>
            </dependency>
            <dependency>
```

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

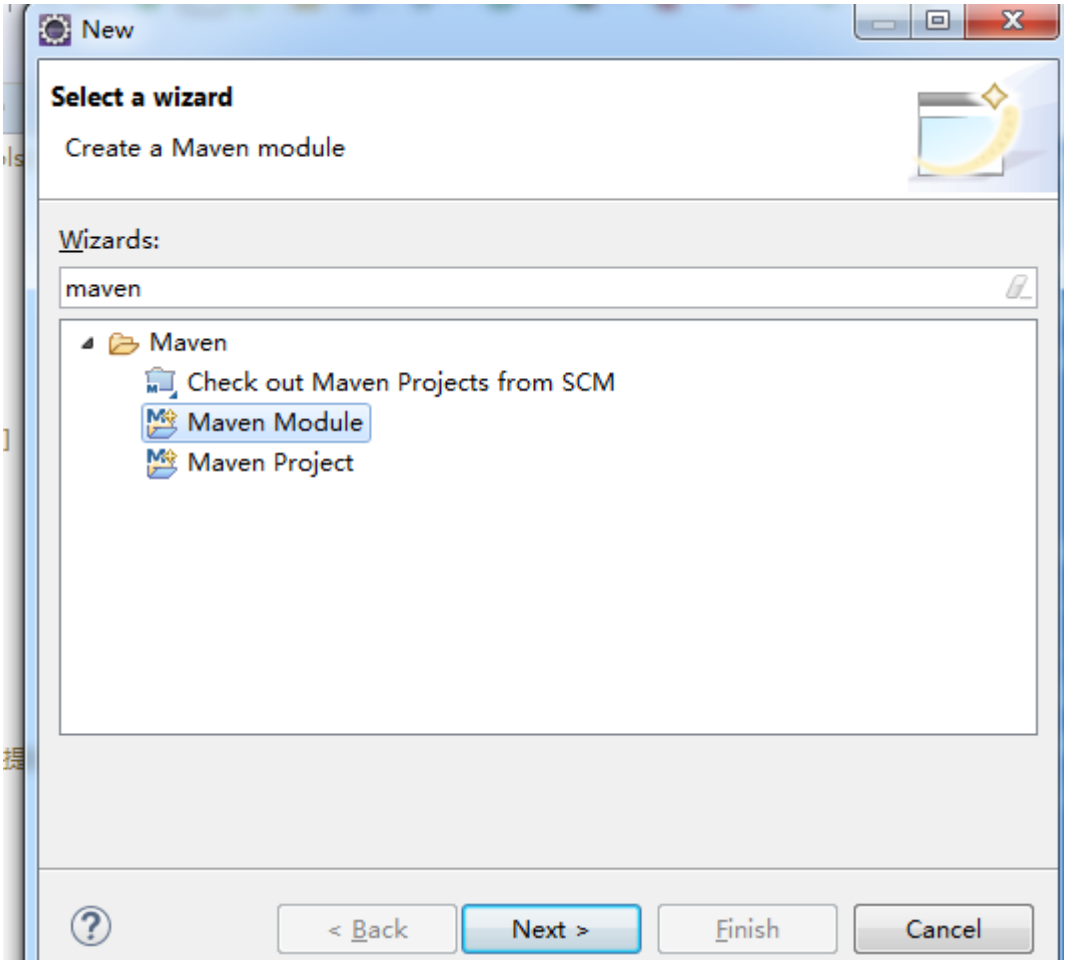
```
                <version>${log4j.version}
</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <finalName>firstMainCloud</finalName>
        <resources>
            <resource>

<directory>src/main/resources</directory>
                <filtering>true</filtering>
            </resource>
        </resources>
        <plugins>
            <plugin>

<groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-resources-
plugin</artifactId>
                <configuration>
                    <delimiters>
                        <delimit>$</delimit>
                    </delimiters>
                </configuration>
            </plugin>
        </plugins>
```

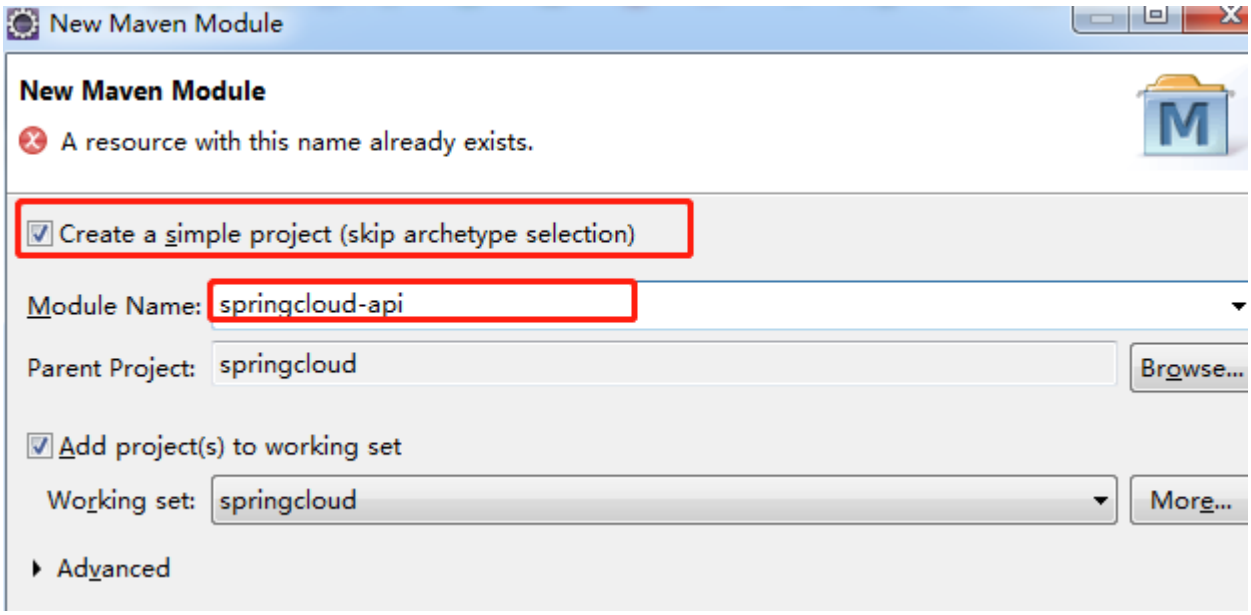
```
86      </build>
87  </project>
```

2. 创建子工程

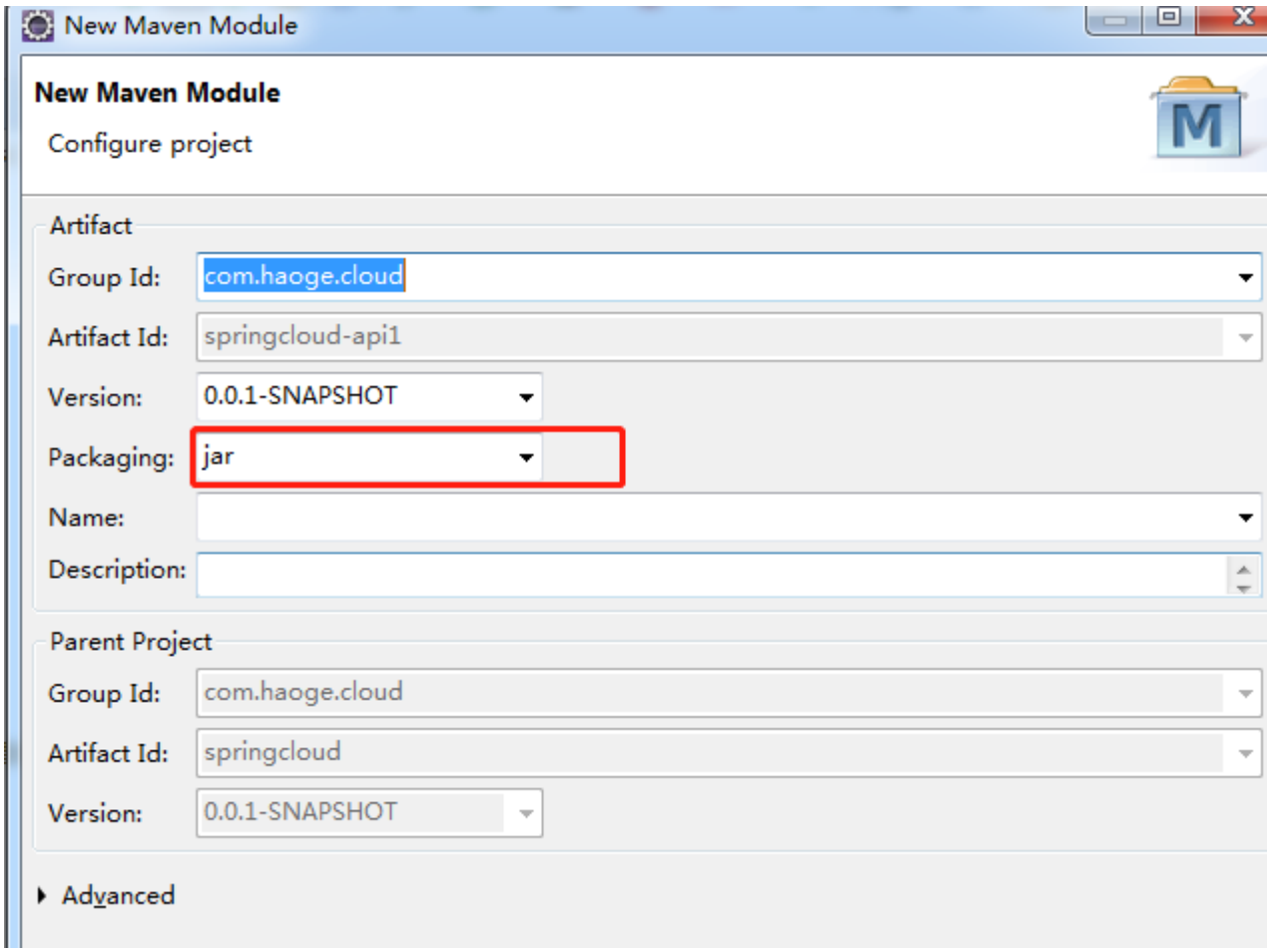
1.创建maven子模块

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

```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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.haoge.cloud</groupId>
    <artifactId>springcloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>springcloud-api</artifactId>
</project>
```

4. 编写公用实体类

```java
@SuppressWarnings("serial")
public class Dept implements Serializable{//必须实现序列化

    private Long deptno;//主键
    private String dname;//部门名称
    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

}
```

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

```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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
```

```
    <groupId>com.haoge.cloud</groupId>
    <artifactId>springcloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>springcloud-provider-dept-8001</artifactId>
   <dependencies>
        <!-- 引入自己定义的api通用包，可以使用Dept部门Entity -->
        <dependency>
            <groupId>com.haoge.cloud</groupId>
            <artifactId>springcloud-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <!-- actuator主管监控信息完善 -->
        <dependency>

<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <!-- 将微服务provider侧注册进eureka eureka后面没有server表示是eureka的客户端-->
        <dependency>

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

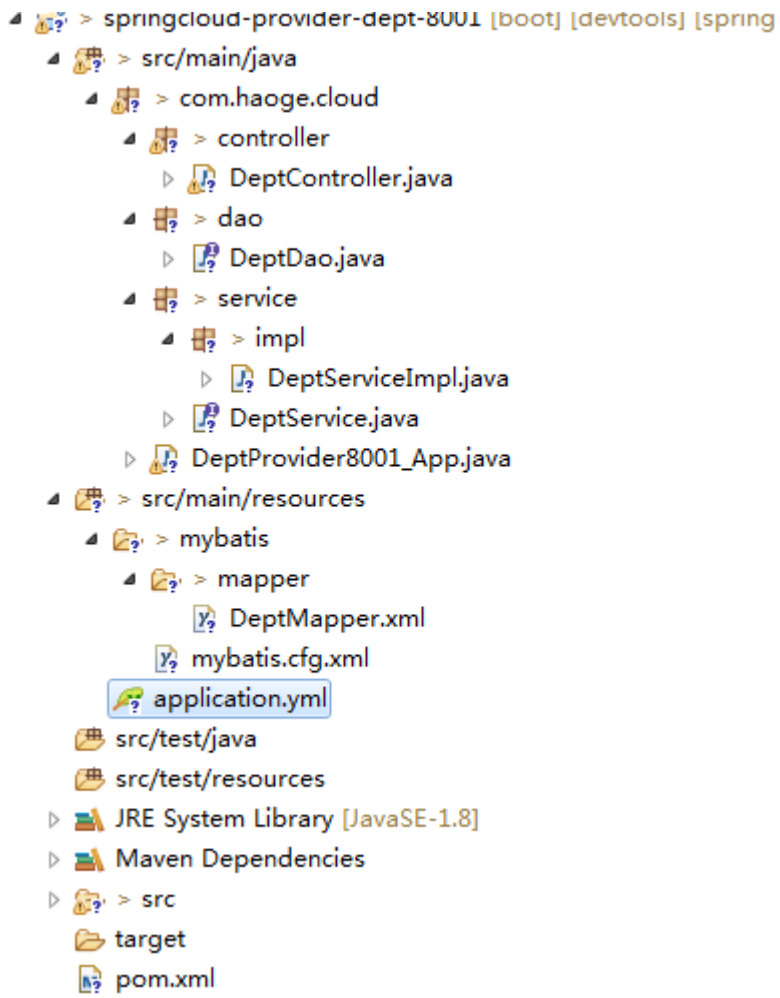
```
66
   <groupId>org.springframework</groupId>
67
   <artifactId>springloaded</artifactId>
68          </dependency>
69          <dependency>
70
   <groupId>org.springframework.boot</groupId>
71              <artifactId>spring-boot-
   devtools</artifactId>
72          </dependency>
73      </dependencies>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

2. yaml文件

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

```yaml
  - classpath:mybatis/mapper/**/*.xml
            # mapper映射文件

spring:
   profiles:
    active:
    - dev
   application:
    name: springcloud-dept  # 当前微服务向外暴露的微服务名称
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource
          # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver
          # mysql驱动包
    url: jdbc:mysql://localhost:3306/cloudDB01
          # 数据库名称
    username: root
    password: 123456
    dbcp2:
      min-idle: 5
            # 数据库连接池的最小维持连接数
      initial-size: 5
            # 初始化连接数
      max-total: 5
            # 最大连接数
      max-wait-millis: 200
```

3. 项目结构



4. 主启动程序

```
@SpringBootApplication
public class DeptProvider8001_App {
    public static void main(String[] args) {

SpringApplication.run(DeptProvider8001_App.class,
 args);
    }
}
```

5. DeptController

```
@RestController
public class DeptController {
    @Autowired
    private DeptService service;

    /**
     * 增加部门的方法
     * @param dept
     * @return
     */

@RequestMapping(value="/dept/add",method=Request
Method.POST)
    public boolean add(@RequestBody Dept dept) {
        return service.add(dept);
    }
    /**
```

```
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

```java
public interface DeptService
{
    public boolean add(Dept dept);

    public Dept get(Long id);

    public List<Dept> list();
}
```

7. DeptServiceImpl

```java
@Service
public class DeptServiceImpl implements
DeptService {
    @Autowired
    private DeptDao dao;

    public boolean add(Dept dept) {
        // TODO Auto-generated method stub
        return dao.addDept(dept);
    }

    @Override
    public Dept get(Long id) {
        // TODO Auto-generated method stub
        return dao.findById(id);
    }

```

```
    @Override
    public List<Dept> list() {
        // TODO Auto-generated method stub
        return dao.findAll();
    }
}

```

8. DeptDao

```
@Mapper
public interface DeptDao
{
    public boolean addDept(Dept dept);

    public Dept findById(Long id);

    public List<Dept> findAll();
}
```

9. DeptMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<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)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <!-- <settings>
        <setting name="cacheEnabled"
value="true" />二级缓存开启
    </settings> -->

</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

```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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.haoge.cloud</groupId>
    <artifactId>springcloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>springcloud-consumer-dept-80</artifactId>

   <dependencies>
        <dependency>
            <groupId>com.haoge.cloud</groupId>
            <artifactId>springcloud-api</artifactId>
            <version>${project.version}</version>
        </dependency>
        <!-- Ribbon相关 ，负载均衡-->
        <dependency>

<groupId>org.springframework.cloud</groupId>
```

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

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

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

<groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- 修改后立即生效，热部署 -->
        <dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
    </dependencies>
</project>
```

3. yaml文件

```
server:
  port: 80
```

4. 主程序

```
@SpringBootApplication
public class DeptConsumer80_App {

    public static void main(String[] args) {

SpringApplication.run(DeptConsumer80_App.class, args);
    }

}
```

5. Controller

```java
@RestController
public class DeptController_Consumer {
    private static final String REST_URL_PREFIX = "http://localhost:8001";
//  private static final String REST_URL_PREFIX = "http://firstmaincloud-dept";

    @Autowired
    private RestTemplate template;
    /**
     * 使用 使用restTemplate访问restful接口非常的简单粗暴无脑。 (url, requestMap,
     * ResponseBean.class)这三个参数分别代表 REST请求地址、请求参数、HTTP响应转换被转换成的对象类型。
     */
    @RequestMapping(value = "/consumer/dept/add")
    public boolean add(Dept dept) {
        return template.postForObject(REST_URL_PREFIX + "/dept/add", dept, boolean.class);
    }
    /**
     * 根据ID查询部门的方法
     *
     * @param dept
     * @return
     */
```

```
22      @RequestMapping(value =
"/consumer/dept/get/{id}")
23      public Dept get(@PathVariable("id") Long id)
{
24          return
template.getForObject(REST_URL_PREFIX +
"/dept/get/" + id, Dept.class);
25      }
26      /**
27       * 查询列表的方法
28       *
29       * @param dept
30       * @return
31       */
32      @SuppressWarnings("unchecked")
33      @RequestMapping(value =
"/consumer/dept/list")
34      public List<Dept> list() {
35
36          return
template.getForObject(REST_URL_PREFIX +
"/dept/list", List.class);
37      }
38      // 测试@EnableDiscoveryClient,消费端可以调用服
务发现
39      @RequestMapping(value =
"/consumer/dept/discovery")
40      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restfl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 getRestTemplete() {
5          return new RestTemplate();
6      }
7  }
```

ConfigBean相当于我们以前写的applicationContext.xml配置文件，getRestTemplete()方法相当于我们我们以前在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)

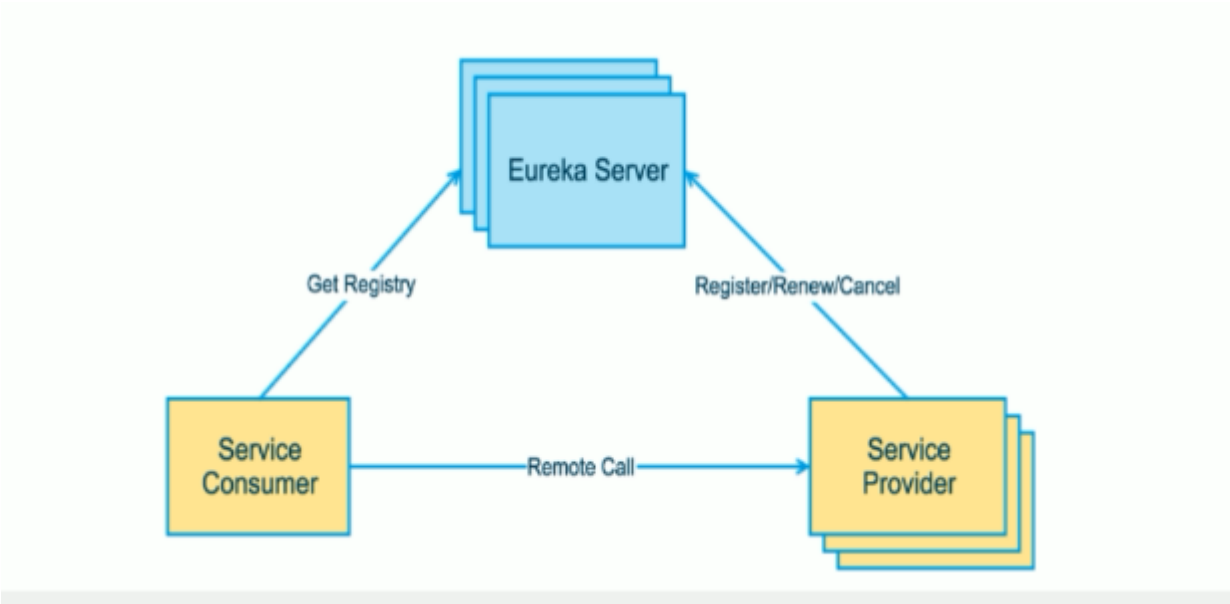
2.Eureka是一个基于rest的服务，用于定位服务，以实现云端中间层服务发现和故障转移。功能类似于dubbo的注册中心，比如zookeeper.SpringCloud主要使用Eureka来实现服务注册和发现功能。

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

```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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.haoge.cloud</groupId>
    <artifactId>springcloud</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>springcloud-eureka-server-7001</artifactId>
  <dependencies>
        <!--eureka-server服务端 -->
        <dependency>

<groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka-server</artifactId>
        </dependency>
        <!-- 修改后立即生效，热部署 -->
        <dependency>

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

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

3、yaml配置文件

```yaml
server:
  port: 7001

eureka:
  instance:
    hostname: localhost #eureka服务端的实例名称(单机版)
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    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
    service-url:
      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），
```

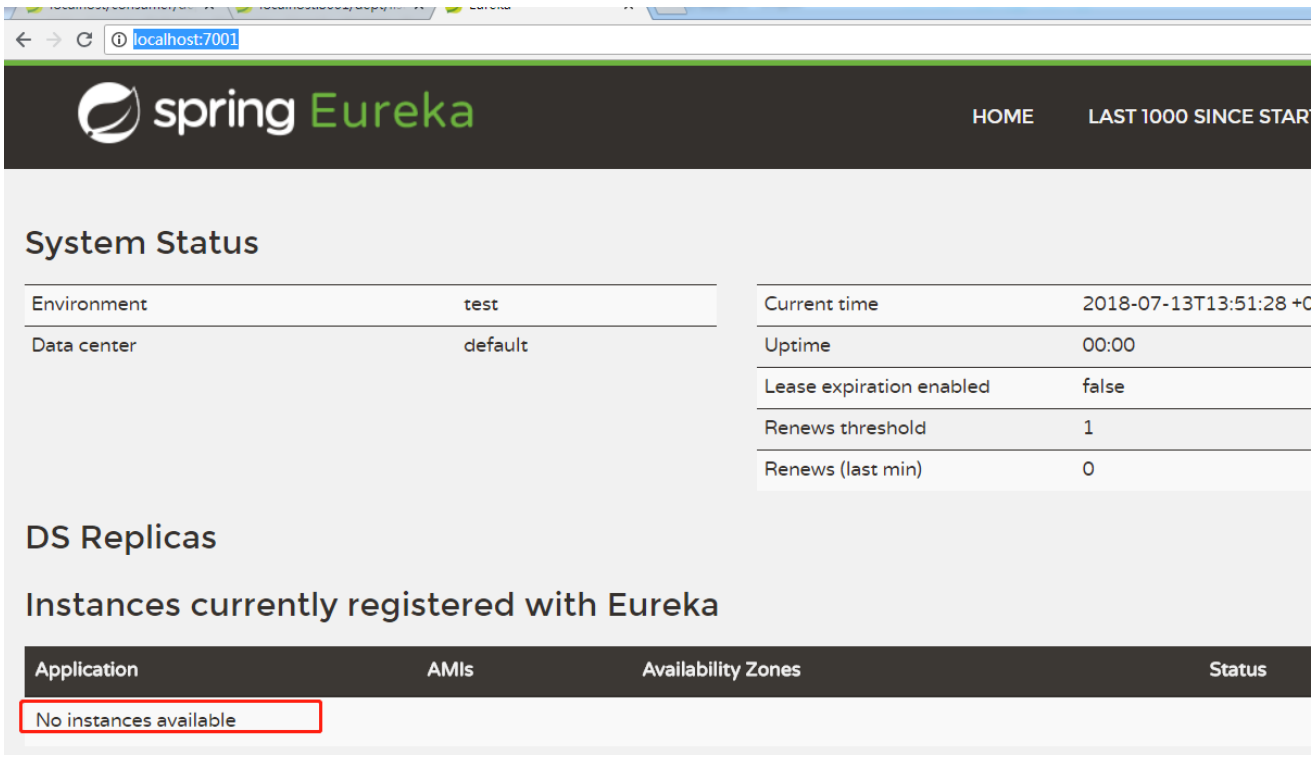
eureka.client.service-url.defaultZone 对应的属性值意义：这个地址是Eureka服务端暴露给外部的地址，外部的微服务如果想注册进这个Eureka Server中，则响应的写这个地址。

4、主启动类

```java
@SpringBootApplication
@EnableEurekaServer// EurekaServer服务器端启动类,接受其它微服务注册进来
public class EurekaServer7001_App {

    public static void main(String[] args) {

SpringApplication.run(EurekaServer7001_App.class, args);
    }
}
```

5、测试，还没有服务注册

http://localhost:7001/



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

```
<!-- 将微服务provider侧注册进eureka eureka后面没有server表示是eureka的客户端-->
        <dependency>

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

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

2、yaml添加配置

```
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
      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。

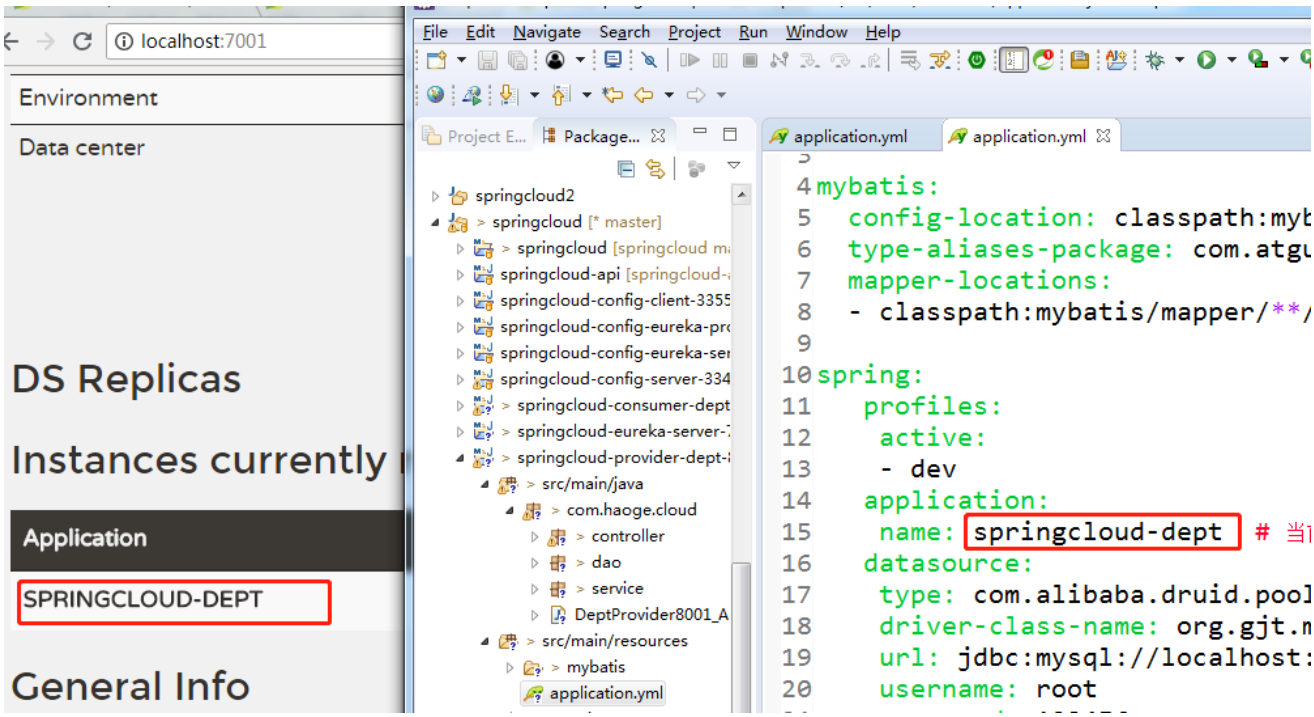
3、主启动类添加注解@EnableEurekaClient

```
@SpringBootApplication
@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
public class DeptProvider8001_App {
    public static void main(String[] args) {

SpringApplication.run(DeptProvider8001_App.class, args);
    }
}
```

4、测试

启动7001服务端项目，再启动8001项目。

http://localhost:7001/

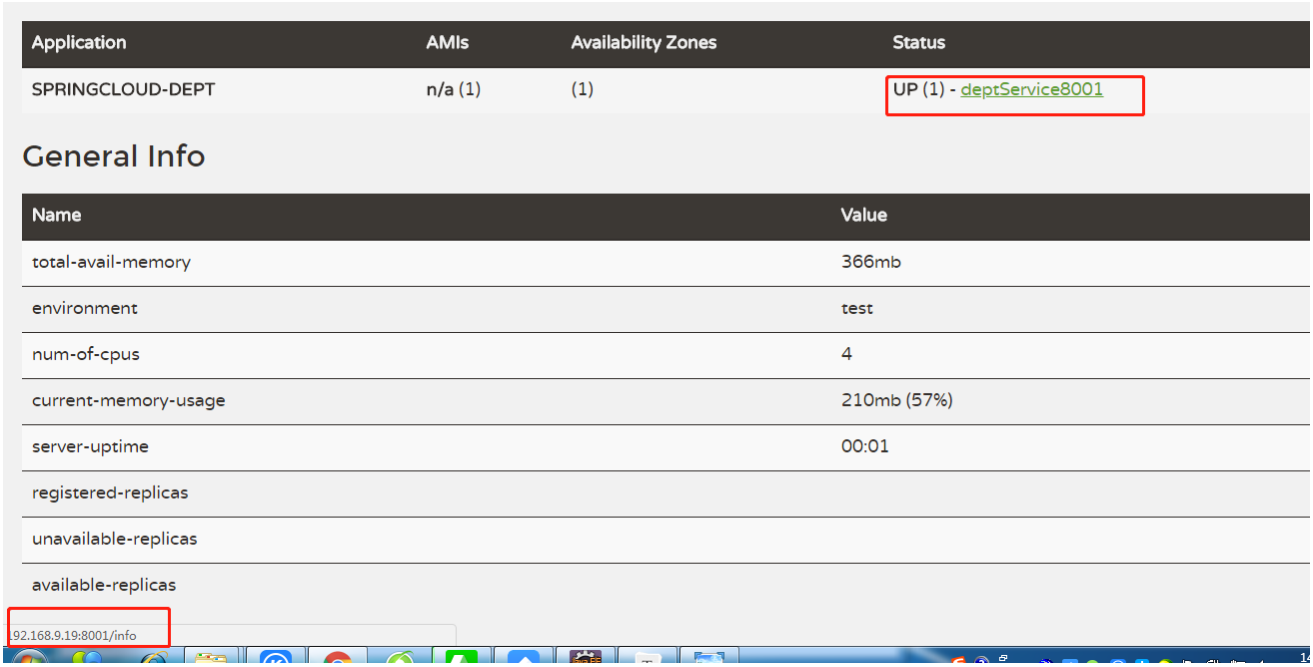


图示为微服务对外暴露的服务名称，是我们在8001项目配置文件中写的spring.application.name对应的属性名称。Eureka自动帮我们转为大写。

5、8001服务配置补充

```yaml
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
      defaultZone: http://localhost:7001/eureka
#单机版
#      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
  instance:
    instance-id: deptService8001      #对当前服务起的别名
    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



6、8001服务显示info信息

6.1 pom依赖

```
<!-- actuator主管监控信息完善 -->
        <dependency>

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

6.2 主工程springcloudPOM添加依赖

```
<build>
        <finalName>firstMainCloud</finalName>
        <resources>
            <resource>

<directory>src/main/resources</directory>
                <filtering>true</filtering>
            </resource>
        </resources>
        <plugins>
            <plugin>

<groupId>org.apache.maven.plugins</groupId>
```

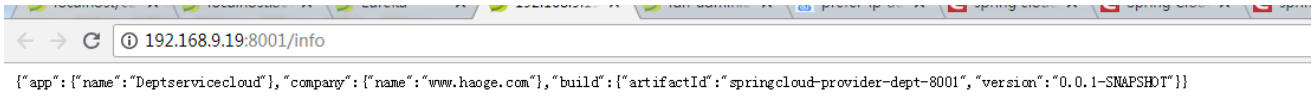
```
                <artifactId>maven-resources-plugin</artifactId>
                <configuration>
                    <delimiters>
                        <delimit>$</delimit>
                    </delimiters>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

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

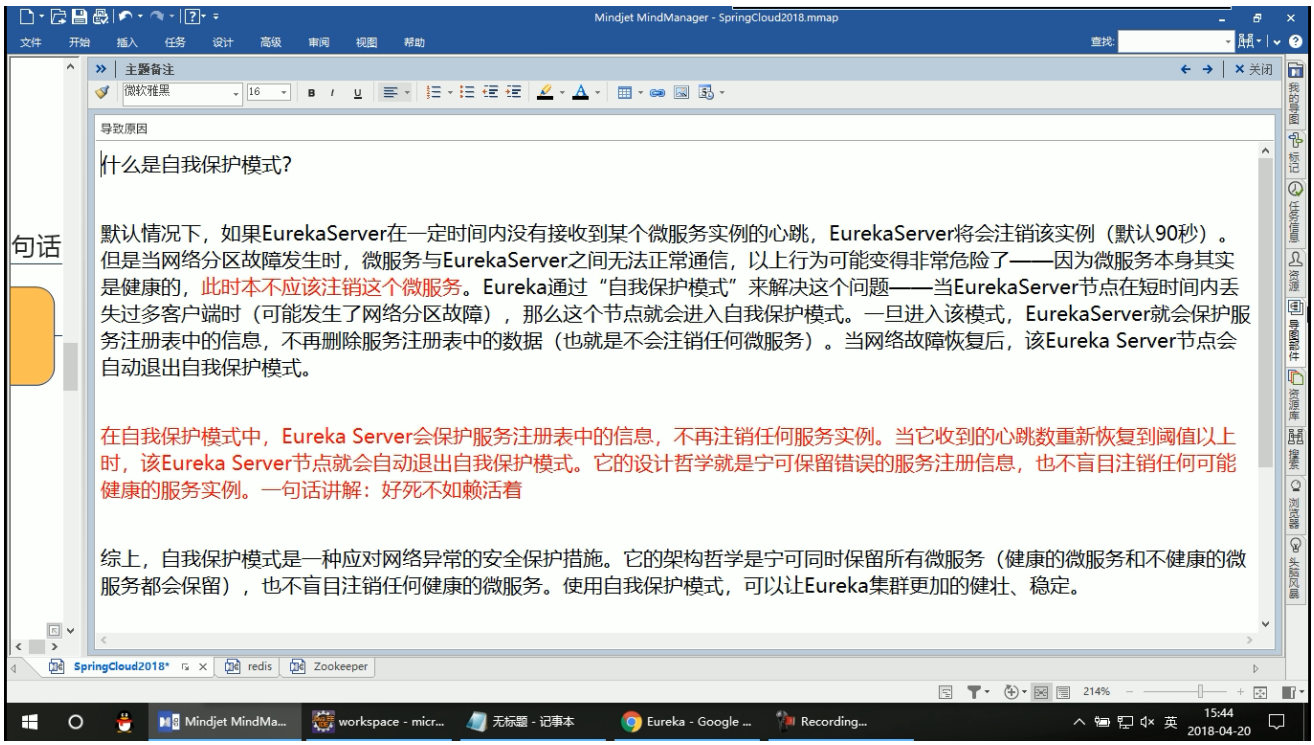
6.3 yaml配置

```
info:
  app.name: Deptservicecloud
  company.name: www.haoge.com
  build.artifactId: $project.artifactId$
  build.version: $project.version$
```

效果：



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

```
@SpringBootApplication
@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
@EnableDiscoveryClient//允许服务发现的注解
public class DeptProvider8001_App {
    public static void main(String[] args) {

SpringApplication.run(DeptProvider8001_App.class, args);
    }
}
```

2、controller添加测试代码

```
    //服务发现模块
    @Autowired
    private DiscoveryClient client;

    @RequestMapping(value = "/dept/discovery", method = RequestMethod.GET)
    public Object discovery()
    {
        List<String> list = client.getServices();//查询eureka中的服务都有哪些
        System.out.println("**********" + list);

        List<ServiceInstance> srvList = client.getInstances("MICROSERVICECLOUD-DEPT");
```

```
        for (ServiceInstance element : srvList) {
System.out.println(element.getServiceId() + "\t" + element.getHost() + "\t" + element.getPort() + "\t"
                    + element.getUri());
        }
        return this.client;
    }
```

3、测试

http://localhost:8001/dept/discovery

4、80项目controller测试代码

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

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

```
server:
  port: 7001

eureka:
  instance:
    hostname: eureka7001.com #eureka服务端的实例名称
#    hostname: localhost #eureka服务端的实例名称(单机版)
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    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
    service-url:
#      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）。
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7003.com:7003/eureka/
```

7001Server配置文件

```yaml
server:
  port: 7002

eureka:
  instance:
    hostname: eureka7002.com #eureka服务端的实例名称
#    hostname: localhost #eureka服务端的实例名称(单机版)
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    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
    service-url:
#      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）。
      defaultZone: http://eureka7001.com:7001/eureka/,http://eureka7003.com:7003/eureka/
```

7003Server配置文件

```yaml
server:
  port: 7003

eureka:
  instance:
    hostname: eureka7003.com #eureka服务端的实例名称
#    hostname: localhost #eureka服务端的实例名称(单机版)
  client:
    register-with-eureka: false     #false表示不向注册中心注册自己。
    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
    service-url:
#      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）。
      defaultZone: http://eureka7002.com:7002/eureka/,http://eureka7001.com:7001/eureka/
```

8001服务提供者yaml配置文件

```yaml
server:
  port: 8001    # 项目的端口号
```

```yaml

mybatis:
  config-location: classpath:mybatis/mybatis.cfg.xml        # mybatis配置文件所在路径
  type-aliases-package: com.atguigu.springcloud.entities    # 所有Entity别名类所在包
  mapper-locations:
  - classpath:mybatis/mapper/**/*.xml             # mapper映射文件

spring:
   profiles:
    active:
    - dev
   application:
    name: springcloud-dept  # 当前微服务向外暴露的微服务名称
   datasource:
    type: com.alibaba.druid.pool.DruidDataSource            # 当前数据源操作类型
    driver-class-name: org.gjt.mm.mysql.Driver              # mysql驱动包
    url: jdbc:mysql://localhost:3306/cloudDB01              # 数据库名称
    username: root
    password: 123456
```

```yaml
    dbcp2:
      min-idle: 5
            # 数据库连接池的最小维持连接数
      initial-size: 5
            # 初始化连接数
      max-total: 5
           # 最大连接数
      max-wait-millis: 200
           # 等待连接获取的最大超时时间
eureka:
  client: #客户端注册进eureka服务列表内
    service-url:
#      defaultZone: http://localhost:7001/eureka
  #单机版
      defaultZone:
http://eureka7001.com:7001/eureka/,http://eureka
7002.com:7002/eureka/,http://eureka7003.com:7003
/eureka/
  instance:
    instance-id: deptService8001      #对当前服务
起的别名
    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
info:
  app.name: Deptservicecloud
  company.name: www.haoge.com
  build.artifactId: $project.artifactId$
  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

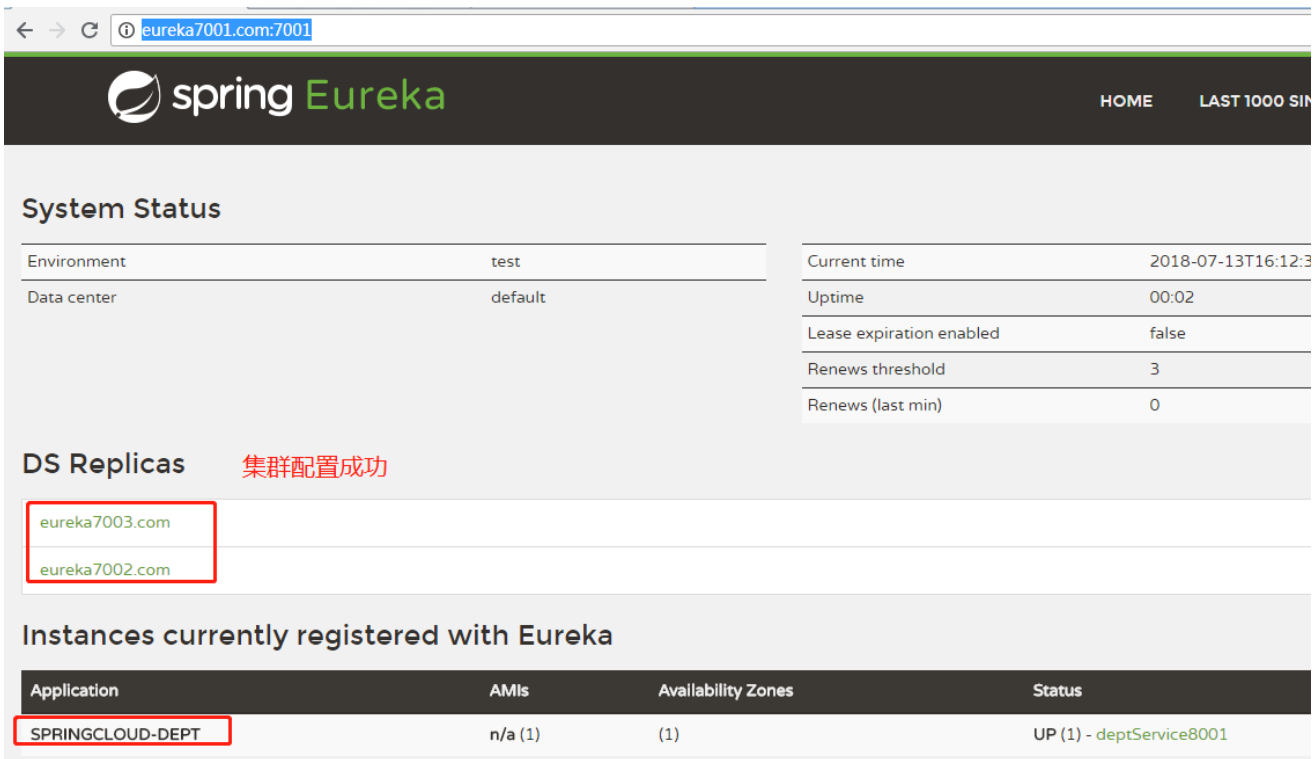
分别访问

http://eureka7001.com:7001/

http://eureka7002.com:7002/

http://eureka7003.com:7003/

效果



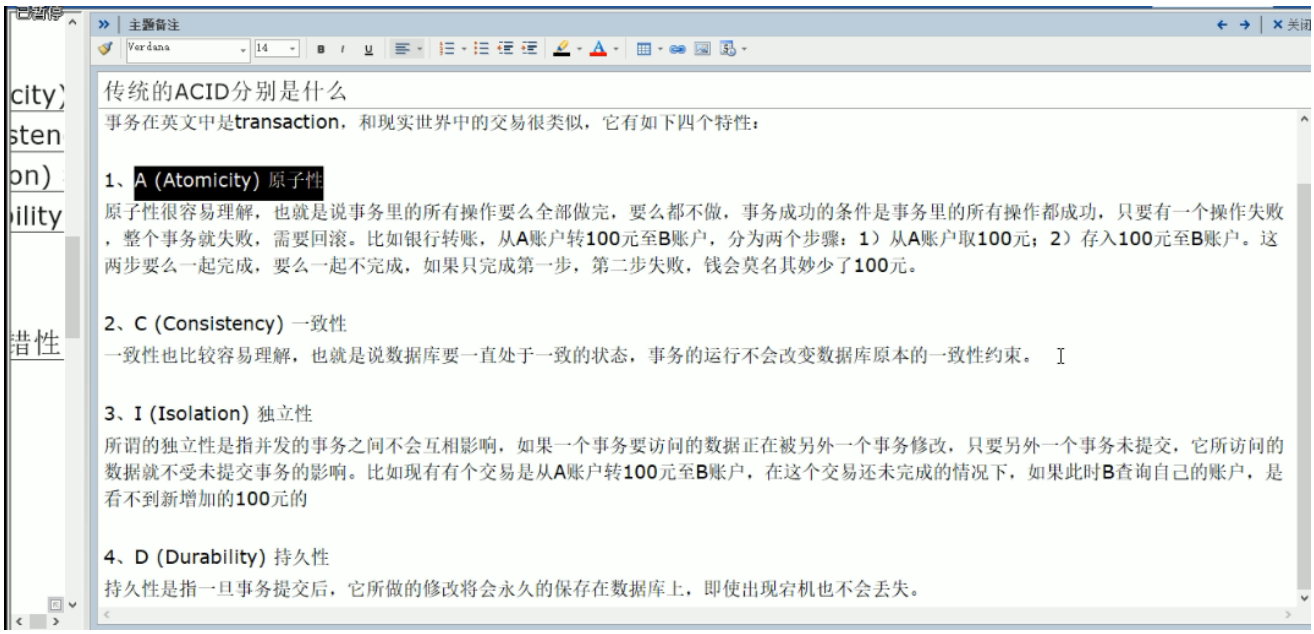
7. Eureka和zookeeper的比较

Zookeeper保证的是CP原则

Eureka保证的是AP原则

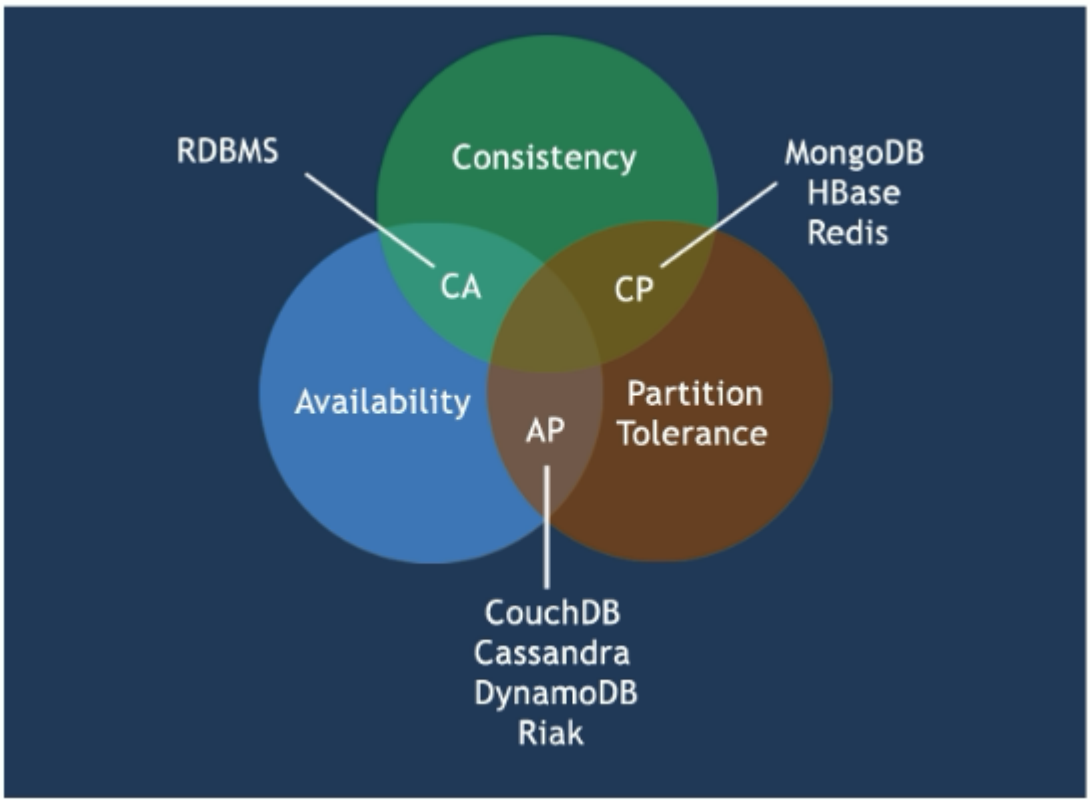
在电商网站的高并发访问情况下，比如双11当天。高可用性比强一致性更加重要。

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。

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不能容忍的。

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。