第1章 Spring Cloud Alibaba 入门

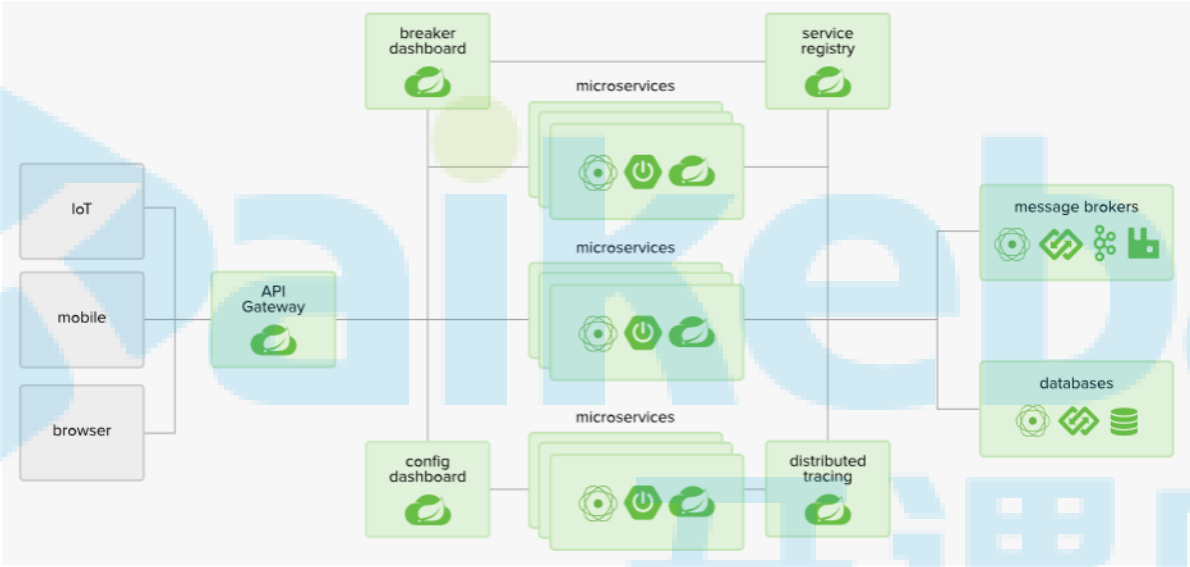
1.1 Spring Cloud 简介

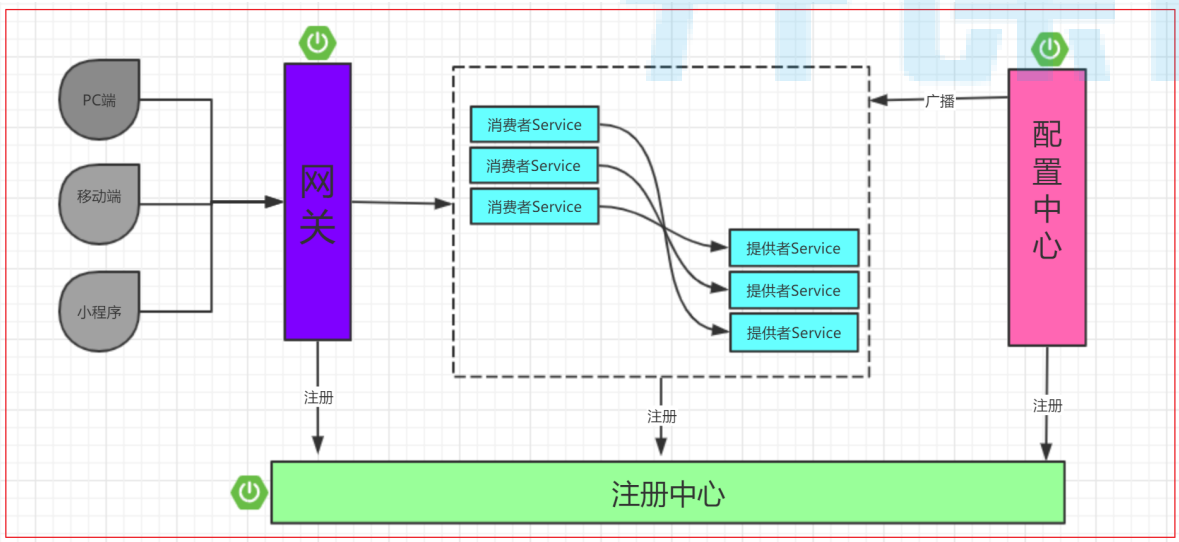
1.1.1 官网简介

打开 Spring Cloud 官网 https://spring.io/projects/spring-cloud 首页，可以看到 Spring Cloud的简介。

【翻译】springcloud 为开发人员提供了在分布式系统中快速构建一些常见模式的工具（例如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性令牌、全局锁、领导选举、分布式会话、集群状态）。分布式系统的协调导致了样板模式，使用 springcloud 的开发人员可以快速地建立实现这些模式的服务和应用程序。它们在任何分布式环境下都能很好地工作，包括开发人员自己的笔记本电脑、裸机数据中心和云计算等托管平台。

下图是原来 Spring Cloud 官网上发布的架构图。





1.1.2 百度百科

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring Cloud 并没有重 复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉

了复杂的配置和实现原理，最终给开发者提供了一套简单易懂、易部署和易维护的分布式系统开发工具包。

1.1.3 总结

Spring Cloud 是什么?

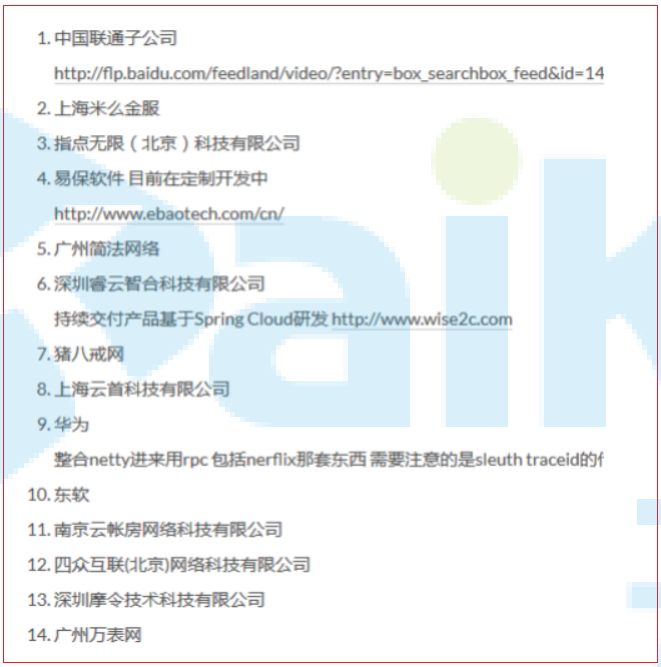
- Spring Cloud 是微服务系统架构的一站式解决方案。

Spring Cloud 与 Spring Boot 是什么关系呢?

- Spring Boot 为 Spring Cloud 提供了代码实现环境，使用 Spring Boot 将其它组件有机融合到了 Spring Cloud 的体系架构中了。所以说，Spring Cloud 是基于 Spring Boot 的微服务系统架构的一站式解决方案。

1.2 Spring Cloud 的国内使用情况

在 Spring Cloud 中国社区博客中可以看到 Spring Cloud 在国内的使用情况。http://blog.springcloud.cn/about/


1. 中国联通子公司
   http://flp.baidu.com/feedland/video/?entry=box_searchbox_feed&id=14
2. 上海米么金服
3. 指点无限（北京）科技有限公司
4. 易保软件 目前在定制开发中
   http://www.ebaotech.com/cn/
5. 广州简法网络
6. 深圳睿云智合科技有限公司
   持续交付产品基于Spring Cloud研发 http://www.wise2c.com
7. 猪八戒网
8. 上海云首科技有限公司
9. 华为
   整合netty进来用rpc 包括nerflix那套东西 需要注意的是sleuth traceid的f
10. 东软
11. 南京云帐房网络科技有限公司
12. 四众互联(北京)网络科技有限公司
13. 深圳摩令技术科技有限公司
14. 广州万表网

1.3 Spring Cloud 在线资源

- Spring Cloud 官网：https://spring.io/projects/spring-cloud
- Spring Cloud 中文网：https://springcloud.cc/
- Spring Cloud 中国社区：http://springcloud.cn/

1.4 Spring Cloud 版本

版本号来源

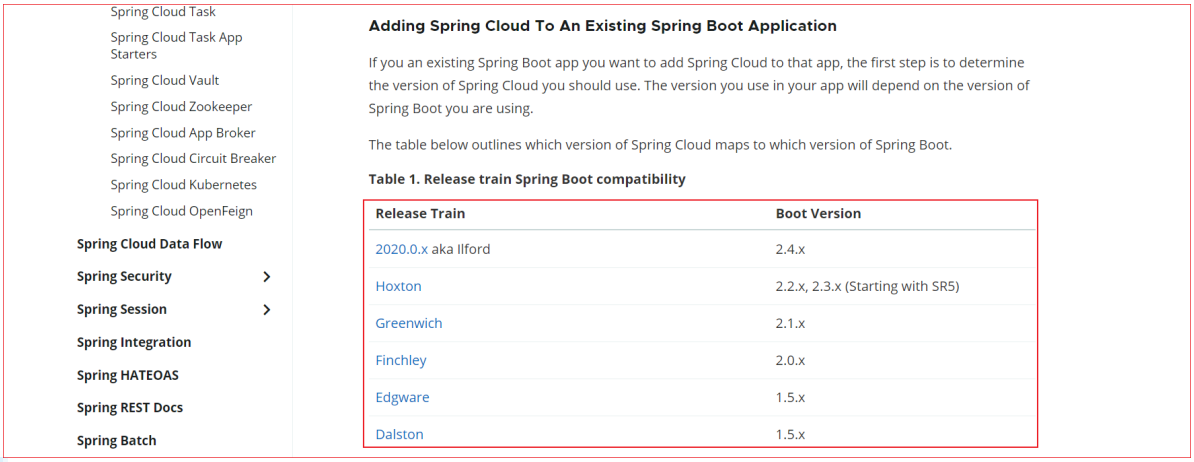
Spring Cloud 的版本号并不是我们通常见的数字版本号，而是一些很奇怪的单词。这些单词均为英国伦敦地铁站的站名。同时根据字母表的顺序来对应版本时间顺序，比如：最早的 Release 版本 Angel（天使），第二个 Release 版本 Brixton（英国地名），然后是 Camden、Dalston、Edgware，目前使用较多的是 Greenwich（格林威治）版本，而最新版本为 Hoxton（英国地名）。

Spring Cloud 的一个大版本在不同的阶段会发布不同类型的小版本号。按照发行的顺序，一般会存在这些版本，但并不一定每个大版本都存在这些小版本。当然，这些小版本后一般还会添加上数字作为其内部的版本。

- PRE：preview，预览版，内测版。

- SNAPSHOT：快照版。
- M 版：Milestone，里程碑版。
- RC 版：Release Candidate，发行候选版本。
- SR 版：Service Release，服务发布版。
- GA：General Availability，则表示这是当前广泛使用的版本。
- CURRENT：表示官方当前推荐版本。

Spring Cloud 与 Spring Boot 版本



1.5 Spring Cloud Alibaba 简介

> 官网wiki：https://github.com/alibaba/spring-cloud-alibaba/wiki

Spring Cloud Alibaba 致力于提供微服务开发的一站式解决方案。此项目包含开发分布式应用服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务。

依托 Spring Cloud Alibaba，您只需要添加一些注解和少量配置，就可以将 Spring Cloud 应用接入阿里分布式应用解决方案，通过阿里中间件来迅速搭建分布式应用系统。

1.6 版本兼容关系

使用 spring cloud alibaba 时特别需要注意版本间的兼容关系，这些关系包括 spring cloud alibaba、spring cloud 与 spring boot 间的版本兼容关系，包括 spring cloud alibaba 与使用的alibaba 中间件版本间的兼容关系。

组件版本关系

Spring Cloud Alibaba Version	Sentinel Version	Nacos Version	RocketMQ Version	Dubbo Version	Seata Version
2.2.3.RELEASE or 2.1.3.RELEASE or 2.0.3.RELEASE	1.8.0	1.3.3	4.4.0	2.7.8	1.3.0
2.2.1.RELEASE or 2.1.2.RELEASE or 2.0.2.RELEASE	1.7.1	1.2.1	4.4.0	2.7.6	1.2.0
2.2.0.RELEASE	1.7.1	1.1.4	4.4.0	2.7.4.1	1.0.0
2.1.1.RELEASE or 2.0.1.RELEASE or 1.5.1.RELEASE	1.7.0	1.1.4	4.4.0	2.7.3	0.9.0
2.1.0.RELEASE or 2.0.0.RELEASE or 1.5.0.RELEASE	1.6.3	1.1.1	4.4.0	2.7.3	0.7.1

毕业版本依赖关系(推荐使用)

Spring Cloud Version	Spring Cloud Alibaba Version	Spring Boot Version
Spring Cloud Hoxton.SR8	2.2.3.RELEASE	2.3.2.RELEASE
Spring Cloud Greenwich.SR6	2.1.3.RELEASE	2.1.13.RELEASE
Spring Cloud Hoxton.SR3	2.2.1.RELEASE	2.2.5.RELEASE
Spring Cloud Hoxton.RELEASE	2.2.0.RELEASE	2.2.X.RELEASE
Spring Cloud Greenwich	2.1.2.RELEASE	2.1.X.RELEASE
Spring Cloud Finchley	2.0.3.RELEASE	2.0.X.RELEASE
Spring Cloud Edgware	1.5.1.RELEASE(停止维护，建议升级)	1.5.X.RELEASE

1.7 SpringCloudAlibaba入门案例环境搭建

本例实现了消费者对提供者的调用，但并未使用到 Spring Cloud，而是使用的 Spring 提供的 RestTemplate 实现。不过后续 Spring Cloud 的运行测试环境就是在此基础上修改出来的。使用 MySQL 数据库，使用 Spring Data JPA 作为持久层技术。

1.7.1 创建提供者工程 01-provider

(1) 创建工程

创建一个 01-provider的SpringBoot工程，并命名为 01-provider。

- 导入 Lombok、Web、JPA 及MySQL 驱动依赖。
- SpringBoot版本：2.3.2.RELEASE

(2) 导入Druid依赖

```
perties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<java.version>1.8</java.version>
operties>

endencies>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.10</version>
</dependency>
<!--修改MySQL驱动版本-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
    <scope>runtime</scope>
</dependency>

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

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

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
pendencies>

ld>
<plugins>
    <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
</plugins>
ild>
```

(3) 定义实体类

```
ta
tity//声明当前类为实体类
onIgnoreProperties({"hibernateLazyInitializer", "handler", "fieldHandler"})//
是json序列化时将java bean中的一些属性忽略掉，序列化和反序列化都受影响。
lic class Depart {
 @Id//声明当前属性为主键id
 @GeneratedValue(strategy = GenerationType.IDENTITY)//配置主键生成策略：使用数据库
键自增策略
 private Integer id;
 private String name;

```

(4) 定义 Repository 接口

```
持久层
lic interface DepartRepository extends JpaRepository<Depart, Integer> {

```

(5) 定义 Service 接口

```
业务层接口
lic interface DepartService {
 boolean saveDepart(Depart depart);
 boolean removeDepartById(int id);
 boolean modifyDepart(Depart depart);
 Depart getDepartById(int id);
 List<Depart> listAllDeparts();

```

(6) 定义 Service 实现类

A、添加数据

```
erride
lic boolean saveDepart(Depart depart) {
 Depart obj = repository.save(depart);
 if(obj != null) {
     return true;
 }
 return false;

```

B、 删除数据

```
erride
 public boolean removeDepartById(int id) {
     if(repository.existsById(id)) {
         repository.deleteById(id);
         return true;
     }
     return false;
 }
```

C、 修改数据

与添加数据代码相同。

```
erride
lic boolean modifyDepart(Depart depart) {
 Depart obj = repository.save(depart);
 if(obj != null) {
     return true;
 }
 return false;

```

D、根据id查询

```
erride
lic Depart getDepartById(int id) {
 if(repository.existsById(id)) {
     return repository.getOne(id);
 }
 Depart depart = new Depart();
 depart.setName("no this depart");
 return depart;

```

E、 查询所有

```
@Override
public List<Depart> listAllDeparts() {
    return repository.findAll();
}
```

(7) 定义处理器

```
//提供者Controller，对外提供接口
@RequestMapping("/provider/depart")
@RestController
public class DepartController {
    @Autowired
    private DepartService service;

    //新增
    @PostMapping("/save")
    public boolean saveHandle(@RequestBody Depart depart) {
        return service.saveDepart(depart);
    }

    //删除
    @DeleteMapping("/del/{id}")
    public boolean deleteHandle(@PathVariable("id") int id) {
        return service.removeDepartById(id);
    }

    //修改
    @PutMapping("/update")
    public boolean updateHandle(@RequestBody Depart depart) {
        return service.modifyDepart(depart);
    }

    //根据id查询
    @GetMapping("/get/{id}")
    public Depart getHandle(@PathVariable("id") int id) {
        return service.getDepartById(id);
    }

    //查询列表
    @GetMapping("/list")
    public List<Depart> listHandle() {
        return service.listAllDeparts();
    }
}
```

(8) 修改配置文件

```
server:
  port: 8081

# 设置Spring-Data-JPA
spring:
  jpa:
    generate-ddl: true
```

```
    show-sql: true
    hibernate:
      ddl-auto: none
  # 配置数据源
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql:///test?useUnicode=true&amp;characterEncoding=utf8
    username: root
    password: root

# 配置日志
logging:
  pattern:
    console: level-%level %msg%n
  level:
    root: info
    org.hibernate: info
    org.hibernate.type.descriptor.sql.BasicBinder: trace
    org.hibernate.type.descriptor.sql.BasicExtractor: trace
    com.abc: debug

```

(9) 启动引导类ProviderApplication

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

1.7.2 创建消费者工程 01-consumer

（1） 创建工程

创建一个 Spring Initializr 工程，并命名为 01-consumer，导入 Lombok 与 Web 依赖。

（2） 定义实体类

```
@Data//作用相当于：@Getter @Setter @RequiredArgsConstructor @ToString
@EqualsAndHashCode
public class Depart {
    private Integer id;
    private String name;
}
```

（3） 定义处理器类

```
//提供者Controller，对外提供接口
@RequestMapping("/provider/depart")
@RestController
public class DepartController {
    @Autowired
    private DepartService service;
```

```
    //新增
    @PostMapping("/save")
    public boolean saveHandle(@RequestBody Depart depart) {
        return service.saveDepart(depart);
    }

    //删除
    @DeleteMapping("/del/{id}")
    public boolean deleteHandle(@PathVariable("id") int id) {
        return service.removeDepartById(id);
    }

    //修改
    @PutMapping("/update")
    public boolean updateHandle(@RequestBody Depart depart) {
        return service.modifyDepart(depart);
    }

    //根据id查询
    @GetMapping("/get/{id}")
    public Depart getHandle(@PathVariable("id") int id) {
        return service.getDepartById(id);
    }

    //查询列表
    @GetMapping("/list")
    public List<Depart> listHandle() {
        return service.listAllDeparts();
    }
}
```

(4) 启动引导类ConsumerApplication

```
@SpringBootApplication
public class ConsumerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class, args);
    }
    /**
     * 注入RestTemplate模板对象，用来发送请求
     * 作用相当于：<bean id=''
class='org.springframework.web.client.RestTemplate'></bean>
     */
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

第2章 Nacos 服务注册与发现

2.1 Nacos 概述

2.1.1 注册中心简介

前面的例子存在一个问题：消费者直接连接的提供者。

这样做的问题是，若提供者出现宕机，或消费者存在高并发情况，那么提供者就会出现问题。所以，我们就需要一个服务注册中心，就像之前的 Zookeeper 一样。提供者对于消费者来说是透明的，不固定的。

所有提供者将自己提供服务的名称及自己主机详情（IP、端口、版本等）写入到另一台主机中的一个列表中，这台主机称为服务注册中心，而这个表称为服务注册表。

所有消费者需要调用微服务时，其会从注册中心首先将服务注册表下载到本地，然后根据消费者本地设置好的负载均衡策略选择一个服务提供者进行调用。

可以充当 Spring Cloud 服务注册中心的服务器很多，如 Zookeeper、Eureka、Consul 等。Spring Cloud Alibaba 中使用的注册中心为 Alibaba 的中间件 Nacos。

2.1.2 Nacos 简介：什么是Nacos?

Nacos 官网 https://nacos.io/

一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。**直白了说：既是一个注册中心也是一个配置中心。**

服务（Service）是 Nacos 世界的一等公民。Nacos 支持几乎所有主流类型的“服务”的发现、配置和管理：

- Kubernetes Service
- gRPC & Dubbo RPC Service
- Spring Cloud RESTful Service

Nacos 的关键特性包括:

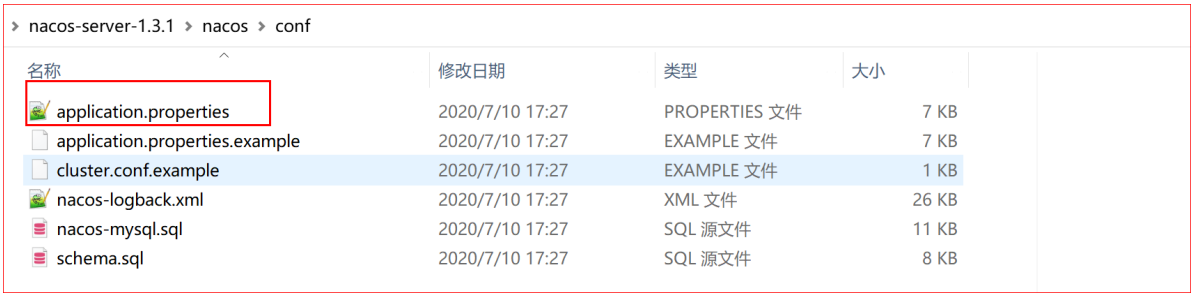
- **服务发现和服务健康监测**
- **动态配置服务**
- **动态 DNS 服务**
- **服务及其元数据管理**
- ...

2.2 Nacos 下载配置与安装

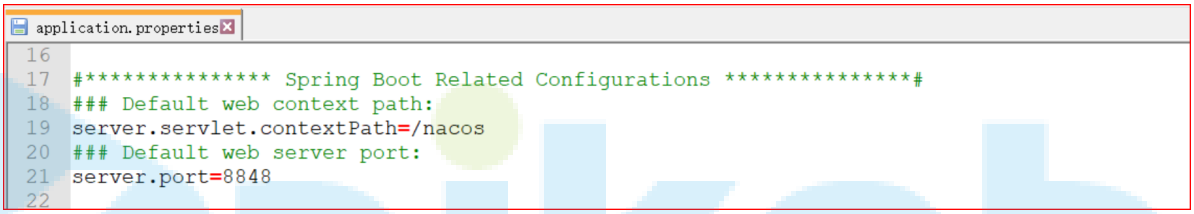
2.2.1 下载

2.2.2 配置

这里下载 nacos-server-1.3.1.zip 这个打过包的 Server。解压到 D 盘根目录。



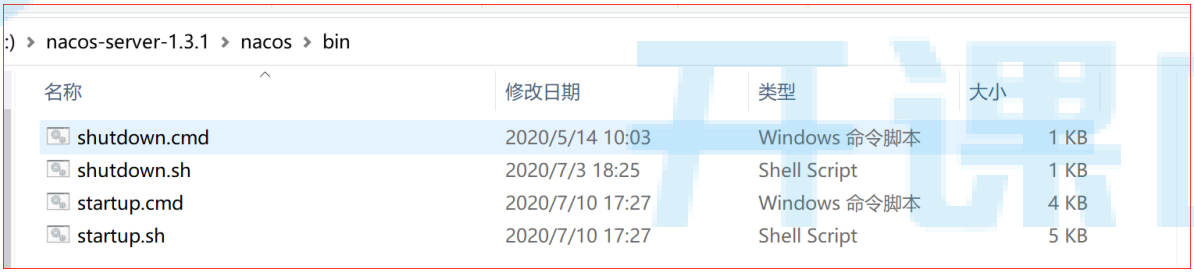
默认 Nacos 服务器的端口号为 8848，上下文路径为/nacos。一般都是采用默认值，但也可以修改。在 nacos/conf/application.properties 中：



2.2.3 Nacos 启动

（1） **启动**

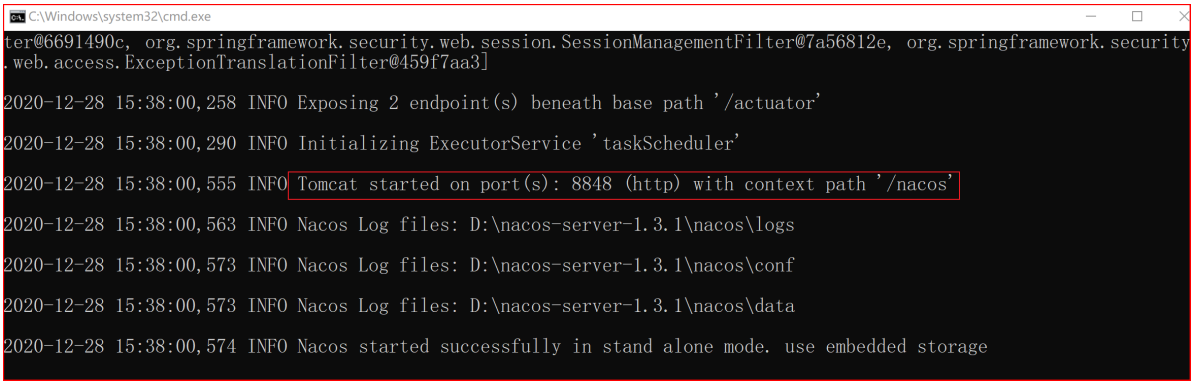
在 nacos/bin 目录中有启动命令文件。



其中 cmd 是 Windows 系统中的命令，sh 是 Linux 系统中的命令。

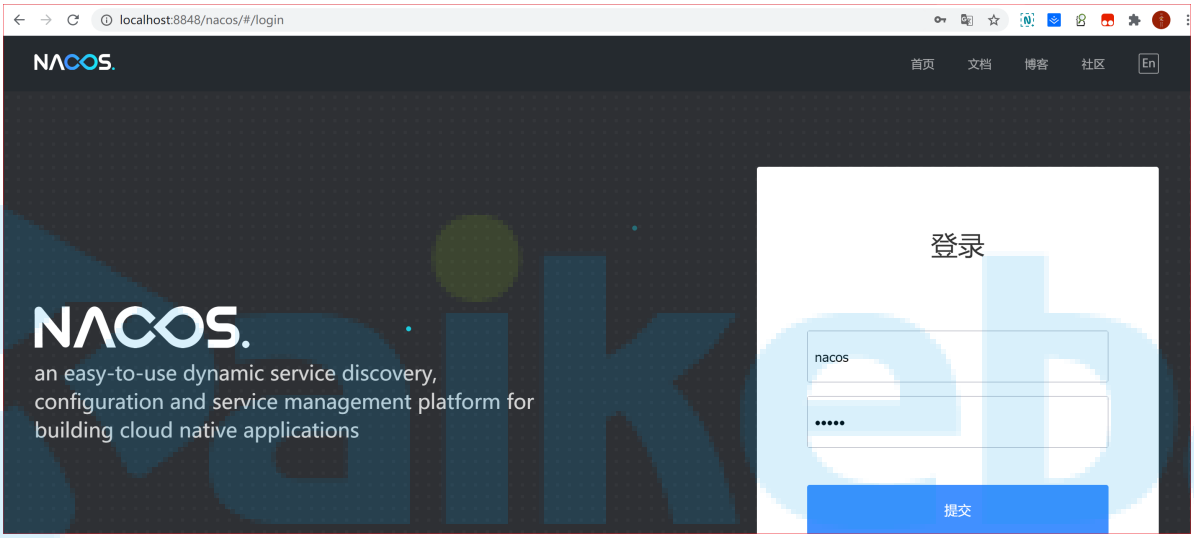
直接双击 startup.cmd 即可运行，并马上可以看到如下内容。

当看到如下日志时，表示启动成功。

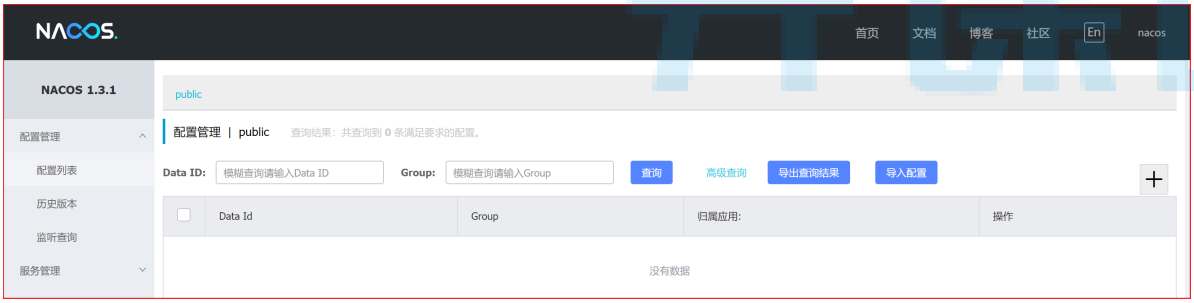


（2） **控制台**

> 地址：http://localhost:8848/nacos



默认账号/密码为 nacos/nacos。 输入账号/密码后即可看到如下界面，说明 nacos server已经启动成功了。



2.3 整合Nacos：定义提供者 02-provider-nacos

2.3.1 定义工程

复制 01-provider工程，并重命名为 02-provider-nacos。

2.3.2 修改 pom

在 pom 中添加了如下依赖：

- spring cloud 依赖
- spring cloud alibaba 依赖
- nacos-discovery 依赖

```xml
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Hoxton.SR8</spring-cloud.version>
    <spring-cloud-alibaba.version>2.2.3.RELEASE</spring-cloud-alibaba.version>
</properties>

<dependencies>
    <!--nacos discovery依赖-->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>

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

    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.1.10</version>
    </dependency>

    <!--修改MySQL驱动版本-->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.47</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </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>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>${spring-cloud-alibaba.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>
```

2.3.3 修改配置文件

在配置文件中添加如下配置。

```yaml
server:
  port: 8081
spring:
  cloud:
    # 指定nacos注册中心的地址
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
  # 指定微服务名称
  application:
    name: msc-provider-depart
  # 设置Spring-Data-JPA
  jpa:
    generate-ddl: true
    show-sql: true
    hibernate:
      ddl-auto: none
  # 配置数据源
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql:///test?useUnicode=true&amp;characterEncoding=utf8
    username: root
    password: root
# 配置日志
logging:
  pattern:
    console: level-%level %msg%n
  level:
    root: info
    org.hibernate: info
    org.hibernate.type.descriptor.sql.BasicBinder: trace
```

```
    org.hibernate.type.descriptor.sql.BasicExtractor: trace
    com.abc: debug
```

2.4 定义消费者 02-consumer-nacos

2.4.1 定义工程

复制 01-consumer 工程，并重命名为 02-consumer-nacos。这个消费者是通过RestTemplate 进行消费的。

2.4.2 修改 pom

与定义 provider 时的相同，consumer 的 pom 也做了如下几处的修改：

- 添加 spring cloud 依赖
- 添加 spring cloud alibaba 依赖
- 添加 nacos-discovery 依赖

```
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Hoxton.SR8</spring-cloud.version>
    <spring-cloud-alibaba.version>2.2.3.RELEASE</spring-cloud-alibaba.version>
</properties>

<dependencies>
    <!--nacos discovery依赖-->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>

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

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

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
```

```
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>${spring-cloud-alibaba.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>
```

2.4.3 修改配置文件

在配置文件中添加如下内容：

```yaml
server:
  port: 8080
spring:
  application:
    name: abcmsc-consumer-depart
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
```

2.4.4 修改处理器类

原来 consumer 中采用的是直连方式访问 provider，现在要根据微服务名称来访问。

```java
@RestController
@RequestMapping("/consumer/depart")
public class DepartController {
    @Autowired
    private RestTemplate restTemplate;

    // private static final String SERVICE_PROVIDER = "http://localhost:8081";
    private static final String SERVICE_PROVIDER = "http://msc-provider-depart";

    @PostMapping("/save")
    public boolean saveHandle(@RequestBody Depart depart) {
```

2.4.5 修改ConsumerApplication类的RestTemplate对象配置

当使用微服务名称来访问 provider 时，其是通过负载均衡方式进行访问的。所以，需要添加如下注解。这里的负载均衡采用的是 spring cloud 自己开发的 spring cloud loadbalancer。

```
    @Bean
    @LoadBalanced    配置开启负载均衡器
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

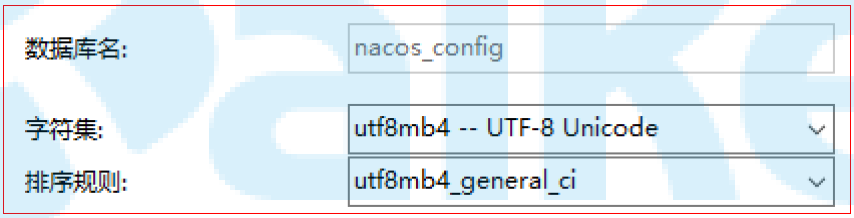
2.5 将数据持久到到外置 MySQL

默认情况下，Nacos 中的配置数据是被持久到到内置的 MySQL 数据库中的，但使用内置数据库，存在很多问题。生产环境下，Nacos 一般会连接到外部的 MySQL。当然，目前 Nacos仅支持 MySQL，并且要求是 5.6.5 及其以上版本。

2.5.1 查找 SQL 脚本文件

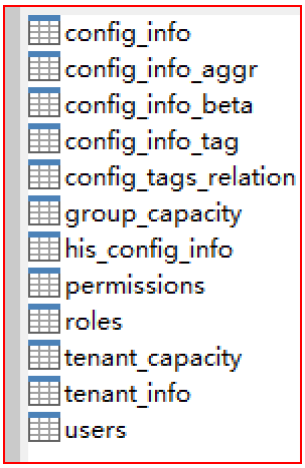
若要连接外置 MySQL，则外置 MySQL 中就要有相应的数据库nacos_config及表。这些表的创建语句Nacos官方已经给出了SQL的脚本文件，在Nacos解压目录的config子目录中nacos-mysql.sql。

首先创建数据库nacos_config



2.5.2 运行脚本文件

脚本文件为Nacos解压目录的config子目录中nacos-mysql.sql。运行该脚本文件，在DBMS中可以查看到创建的 DB 及表。



2.5.3 修改 Nacos 配置

打开 Nacos 安装目录下的 conf/application.properties 文件。

```
#*************** Config Module Related Configurations ***************#
### If use MySQL as datasource:
spring.datasource.platform=mysql

### Count of DB:
db.num=1

### Connect URL of DB:
db.url.0=
jdbc:mysql://127.0.0.1:3306/nacos_config?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true
&useUnicode=true&useSSL=false&serverTimezone=UTC
db.user=root
db.password=root
```

2.5.4 重启服务：看到use external storage表示成功

```
2020-12-28 16:53:23,302 INFO Exposing 2 endpoint(s) beneath base path '/actuator'
2020-12-28 16:53:23,337 INFO Initializing ExecutorService 'taskScheduler'
2020-12-28 16:53:23,800 INFO Tomcat started on port(s): 8848 (http) with context path '/nacos'
2020-12-28 16:53:23,806 INFO Nacos Log files: D:\nacos-server-1.3.1\nacos\logs
2020-12-28 16:53:23,809 INFO Nacos Log files: D:\nacos-server-1.3.1\nacos\conf
2020-12-28 16:53:23,810 INFO Nacos Log files: D:\nacos-server-1.3.1\nacos\data
2020-12-28 16:53:23,810 INFO Nacos started successfully in stand alone mode. use external storage
```

2.6 Nacos 集群搭建

无论是 Nacos Discovery 还是 Nacos Config，单机版 Nacos 都存在单点问题。所以需要搭建高可用的 Nacos 集群。

2.6.1 搭建三台 Nacos

（1） 修改配置并复制目录

首先随意创建一个目录，用于存放三个 Nacos 服务器。然后再复制原来配置好的单机版的 Nacos 到这个目录，并重命名为 nacos8847。

打开 nacos8847/conf，重命名其中的 cluster.conf.example 为 cluster.conf。然后打开该文件，在其中写入三个 nacos 的 ip:port。

cluster.conf

```
# limitations under the License.
#

#it is ip
#example
10.20.5.11:8847
10.20.5.11:8848
10.20.5.11:8849
```

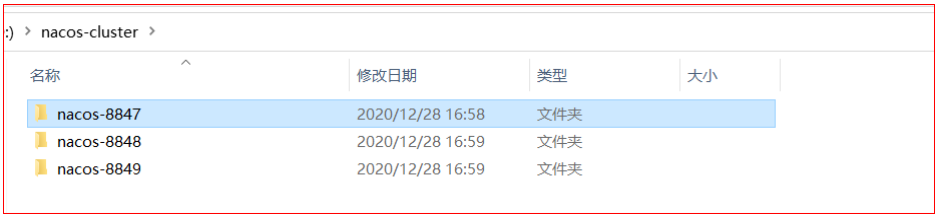
然后再打开 nacos8847/conf/application.properties 文件，修改端口号为 8847。

cluster.conf | application.properties

```
#*************** Spring Boot Related Configurations ***************#
### Default web context path:
server.servlet.contextPath=/nacos
### Default web server port:
server.port=8847

```

然后再将 nacos8847 目录复制三份，分别命名为 nacos8848、nacos8849。并重新指定端口号分别为 8848 与 8849。



(2) 启动集群

```
nacos-cluster\nacos-8847\bin>startup.cmd -m cluster
nacos-cluster\nacos-8848\bin>startup.cmd -m cluster
nacos-cluster\nacos-8849\bin>startup.cmd -m cluster
```