芋道 Spring Boot 注册中心 Nacos 入门

⭐⭐⭐ Spring Boot 项目实战	⭐⭐⭐ Spring Cloud 项目实战
《Dubbo 实现原理与源码解析 —— 精品合集》	《Netty 实现原理与源码解析 —— 精品合集》
《Spring 实现原理与源码解析 —— 精品合集》	《MyBatis 实现原理与源码解析 —— 精品合集》
《Spring MVC 实现原理与源码解析 —— 精品合集》	《数据库实体设计合集》
《Spring Boot 实现原理与源码解析 —— 精品合集》	《Java 面试题 + Java 学习指南》

摘要: 原创出处 http://www.iocoder.cn/Spring-Boot/registry-nacos/ 「芋道源码」欢迎转载，保留摘要，谢谢！

> 本文在提供完整代码示例，可见 https://github.com/YunaiV/SpringBoot-Labs 的 lab-44 目录。
>
> 原创不易，给点个 Star 嘿，一起冲鸭！

1. 概述

在《Nacos 极简入门》中，我们已经学习了如何搭建一个 Nacos 服务。如果还没有的胖友，赶紧先去简单学习下，重点是跟着该文「2. 单机部署」小节，自己搭建一个 Nacos 服务。

本文，我们来学习下如何在 Spring Boot 中，将 Nacos 作为一个**注册中心**，实现分布式环境下的服务注册与发现。

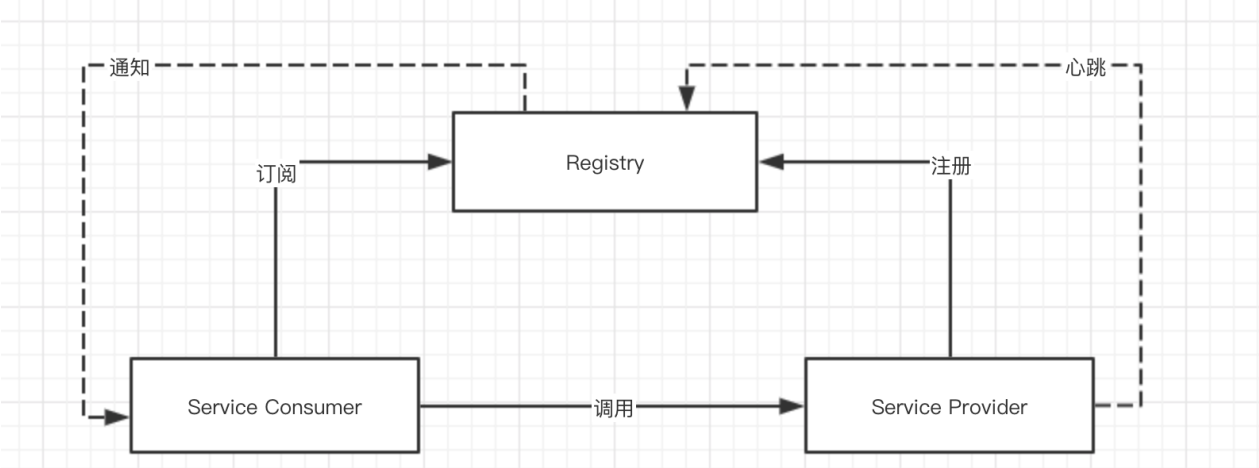
> 友情提示：对 Nacos 作为配置中心感兴趣的胖友，可以看看《芋道 Spring Boot 配置中心 Nacos 入门》文章。

2. 注册中心原理

在使用注册中心时，一共有三种角色：服务提供者（Service Provider）、服务消费者（Service Consumer）、注册中心（Registry）。

> 在一些文章中，服务提供者被称为 Server，服务消费者被称为 Client。胖友们知道即可。

三个角色交互如下图所示：



- Provider：
  - 启动时，向 Registry **注册**自己为一个服务（Service）的实例（Instance）。
  - 同时，定期向 Registry 发送**心跳**，告诉自己还存活。
  - 关闭时，向 Registry **取消注册**。
- Consumer：

    - 启动时，向 Registry **订阅**使用到的服务，并缓存服务的实例列表在内存中。
    - 后续，Consumer 向对应服务的 Provider 发起**调用**时，从内存中的该服务的实例列表选择一个，进行远程调用。
    - 关闭时，向 Registry **取消订阅**。
- Registry：
    - Provider 超过一定时间未**心跳**时，从服务的实例列表移除。
    - 服务的实例列表发生变化（新增或者移除）时，通知订阅该服务的 Consumer，从而让 Consumer 能够刷新本地缓存。

当然，不同的注册中心可能在实现原理上会略有差异。例如说，Eureka 注册中心，并不提供通知功能，而是 Eureka Client 自己定期轮询，实现本地缓存的更新。

另外，Provider 和 Consumer 是角色上的定义，一个服务**同时**即可以是 Provider 也可以作为 Consumer。例如说，优惠券服务可以给订单服务提供接口，同时又调用用户服务提供的接口。

3. 快速入门

> 示例代码对应仓库：lab-44-nacos-discovery-demo。

本小节，我们来搭建一个 Spring Boot 示例，注册到 Nacos 服务器上。

3.1 引入依赖

在 `pom.xml` 文件中，引入相关依赖。

```
<?xml version="1.0" encoding="UTF-8"?>
<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://
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.2.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

```
    <modelVersion>4.0.0</modelVersion>

    <artifactId>lab-44-nacos-discovery-demo</artifactId>

    <dependencies>
        <!-- 实现对 SpringMVC 的自动化配置 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- 实现对 Nacos 作为注册中心的自动化配置 -->
        <dependency>
            <groupId>com.alibaba.boot</groupId>
            <artifactId>nacos-discovery-spring-boot-starter</artifactI
            <version>0.2.4</version>
        </dependency>
    </dependencies>

</project>
```

- 重点是引入 nacos-discovery-spring-boot-starter 依赖，实现对 Nacos 作为注册中心的自动化配置。

3.2 配置文件

在 `application.yml` 中，添加 Nacos 配置，如下：

```yaml
spring:
  application:
    name: demo-application # 应用名

nacos:
  # Nacos 配置中心的配置项，对应 NacosDiscoveryProperties 配置类
  discovery:
    server-addr: 127.0.0.1:18848 # Nacos 服务器地址
    auto-register: true # 是否自动注册到 Nacos 中。默认为 false。
    namespace: # 使用的 Nacos 的命名空间，默认为 null。
    register:
      service-name: ${spring.application.name} # 注册到 Nacos 的服务名
      group-name: DEFAULT_GROUP # 使用的 Nacos 服务分组，默认为 DEFAULT_
      cluster-name: # 集群名，默认为空。
```

`spring.application.name` 配置项，应用名。默认情况下，如果我们不设置 `nacos.discovery.register.service-name` 配置项，Spring Boot 注册到 Nacos 的服务名为它。

`nacos.discovery` 配置项，为 Nacos 作为注册中心的配置，对应 NacosDiscoveryProperties 配置类。

- `server-addr`：Nacos 服务器地址。
- `auto-register`：是否自动注册到 Nacos 中。默认为 `false`。😈 这里，我们设置为 `true`，自动注册到 Nacos 中。
- `namespace`：使用的 Nacos 的命名空间，默认为 `null`。

> FROM 《Nacos 文档 —— Nacos 概念》
>
> **命名空间**
> 用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置。Namespace 的常用场景之一是不同环境的配置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。 * 也就是说，如果我们多个环境共享同一个 Nacos 服务，可以通过 `namespace` 来区分正式、预发布、UAT、开发等环境。

- `register`：注册服务的实例的信息，对应 Register 类。
  - `service-name`：服务名称。未配置的情况下，默认使用 `spring.application.name` 配置项。
  - `group-name`：使用的 Nacos 服务分组，默认为 `DEFAULT_GROUP`。

    > FROM 《Nacos 文档 —— Nacos 概念》
    >
    > **服务分组**
    > 不同的服务可以归类到同一分组。

  - `cluster-name`：使用的 Nacos 服务集群，默认为空。

    > FROM 《Nacos 文档 —— Nacos 概念》
    >
    > **虚拟集群**
    > 同一个服务下的所有服务实例组成一个默认集群, 集群可以被进一步按需求划分，划分的单位可以是虚拟集群。

😜 突然看到 Nacos 一大片的概念，胖友会有点懵逼。问题不大，跟着示例继续往下，会逐步有感觉的。

更多 Nacos Discovery Spring Boot 配置项，可以看看《nacos-spring-boot-project —— WIKI》文章。

3.3 ProviderController

在 `cn.iocoder.springboot.lab44.nacosdemo.controller` 包下，创建 ProviderController 类，作为 Provider 提供示例 API 接口。代码如下：

```
@RestController
@RequestMapping("/provider")
public class ProviderController {

    @GetMapping("/demo")
    public String provider() {
        return "echo";
    }

}
```

3.4 ConsumerController

在 `cn.iocoder.springboot.lab44.nacosdemo.controller` 包下，创建 ConsumerController 类，模拟 Consumer 调用「3.3 ProviderController」提供示例 API 接口。代码如下：

> 友情提示：通过直接使用当前 Spring Boot 示例，来模拟 Consumer 角色，方便简洁。

```
@RestController
@RequestMapping("/consumer")
public class ConsumerController {

    @NacosInjected
    private NamingService namingService;

    private RestTemplate restTemplate = new RestTemplate();

    @GetMapping("/demo")
    public String consumer() throws IllegalStateException, NacosExcept
        // <1> 获得实例
        Instance instance = null;
        if (false) {
            List<Instance> instances = namingService.getAllInstances("
            // 获得首个实例，进行调用
            instance = instances.stream().findFirst()
                    .orElseThrow(() -> new IllegalStateException("未找
        } else {
            instance = namingService.selectOneHealthyInstance("demo-ap
```

```
        }
        // <2> 执行请求
        return restTemplate.getForObject("http://" + instance.toInetAd
                String.class);
    }

}
```

- `namingService` 属性，NamingService 对象，用于调用 Nacos 作为命名服务提供的 API。不过它比较特殊，需要通过 @NacosInjected 注解，来进行注入到 Bean 中。
- `/consumer/demo` 接口，模拟 Consumer 调用「3.3 ProviderController」提供示例 API 接口。
  - `<1>` 处，获得一个 `demo-application` 服务对应的实例。
  - `<2>` 处，使用 RestTemplate 向获得的服务实例，发起 HTTP 远程调用。

3.5 Application

创建 `Application.java` 类，配置 `@SpringBootApplication` 注解即可。代码如下：

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

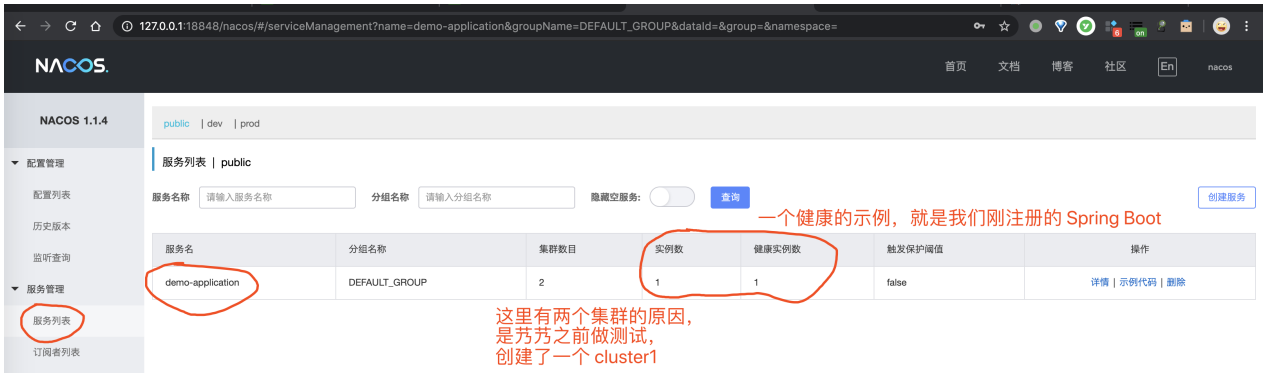
3.6 简单测试

① 启动 Spring Boot 应用，开始我们本轮的测试。此时，控制台输出 Spring Boot 应用注册到 Nacos 中的日志如下：

```
2020-01-24 22:34:03.592  INFO 39657 --- [           main] c.a.b.n.d.a.
```

- 从日志中，我们可以看到 Spring Boot 注册到 Nacos 中的基本信息。

打开 Nacos UI 界面的「服务列表」菜单，进入「服务列表」功能。此时，我们可以看到 `demo-application` 服务。如下图所示：



② 使用浏览器，访问 http://127.0.0.1:8080/consumer/demo 接口，模拟 Consumer 调用「3.3 ProviderController」提供示例 API 接口。返回结果如下：

```
echo
```

- 调用成功，美滋滋~

打开 Nacos UI 界面的「订阅者列表」菜单，进入「订阅者列表」功能。同时，我们使用 `demo-application` 服务，进行搜索。此时，我们可一个订阅者。如下图所示：

666. 彩蛋

至此，我们已经完成了在 Spring Boot 项目中使用 Nacos 作为注册中心。不过一般使用到注册中心，公司一般已经采用微服务架构。此时，更多的采用以 Spring Cloud 作为基础选型，可以考虑使用 spring-cloud-alibaba-nacos-discovery 项目。

> 友情提示：如果想要了解 Spring Cloud Alibaba 的胖友，可以阅读《芋道 Spring Cloud Alibaba 注册中心 Nacos 入门》文章。

目前感觉，在 Spring Boot 项目中使用 Nacos 作为注册中心，貌似场景并不多。暂时能想到的好处，可以方便通过 Nacos 知道哪些 Spring Boot 节点正在启动中。

Spring Boot

PREVIOUS:
记一次由 Redis 分布式锁造成的 P0 重大事故，避免以后踩坑！
NEXT:
Java 离 Linux 内核有多远?

微信公众号福利：芋道源码

0. 阅读源码葵花宝典

1. RocketMQ / MyCAT / Sharding-JDBC 详细中文注释源码

2. 您对于源码的疑问每条留言都将得到认真回复

3. 新的源码解析文章实时收到通知，每周六十点更新

4. 认真的源码交流微信群

分类

- Kong [1]
- Maven [1]
- MongoDB [3]
- MyBatis [7]
- MyCAT [9]
- Nacos [13]
- Netty [11]
- Nginx [3]
- NodeJS [2]
- Onemall [1]
- Prometheus [1]
- RabbitMQ [2]
- Redis [3]
- Resilience4j [12]
- Ribbon [9]
- RocketMQ [29]
- RxJava [7]
- SOFA Mosn [1]
- SOFA RPC [4]
- Seata [6]
- Sentinel [17]
- Sentry [1]
- Sharding Sphere [6]
- Sharding-JDBC [19]
- Shiro [8]

SkyWalking 42

Solr 1

Soul 1

Spring 24

Spring Boot 99

Spring Cloud 33

Spring Security 34

Spring Session 1

Spring Webflux 8

Spring-Cloud-Config 1

Spring-Cloud-Gateway 26

Spring-MVC 13

TCC-Transaction 7

Tomcat 18

XXL-JOB 1

Yudao 2

Zipkin 11

Zookeeper 2

Zuul 8

书单 33

工作内推 4

性能测试 9

数据结构与算法 4

电商开源项目 2

精进 4791

芋道源码的周八 [20]

设计模式 [25]

鸡汤 [7]