

# 目 录

致谢

介绍

开发环境

Spring Cloud 介绍

Spring Cloud 简介

Spring Cloud 入门配置

Spring Cloud 的子项目介绍

Spring Cloud 与 Spring Boot 的关系

微服务 与 Spring Boot

微服务架构概述

微服务 API 设计原则

微服务存储设计原则

提升微服务的并发访问能力

微服务的注册与发现

服务发现的意义

如何集成Eureka

实现服务的注册与发现

微服务的消费

微服务的消费模式

常见微服务的消费者

实现服务的消费者

实现服务的负载均衡及高可用

API 网关

API 网关的意义

常见 API 网关的实现方式

如何集成Zuul

实现 API 网关

微服务的部署与发布

部署微服务将面临的挑战

持续交付与持续部署微服务

基于容器的部署与发布微服务

Docker 简介

基于 Docker 打包

基于 Docker 发布

基于 Docker 部署

微服务的集中化配置

为什么需要集中化配置

集中化配置的实现原理

如何集成 Spring Cloud Config

实现微服务的集中化配置

微服务的高级主题——自动扩展

什么是自动扩展

自动扩展的意义

自动扩展的常见模式

实现微服务的自动扩展

微服务的高级主题——熔断机制

什么是服务的熔断机制

熔断的意义

熔断与降级的区别

如何集成 Hystrix

实现微服务的熔断机制

参考资料

## 致谢

当前文档《Spring Cloud 教程(Spring Cloud Tutorial)》由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建, 生成于 2018-04-30。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能, 以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理, 书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候, 发现文档内容有不恰当的地方, 请向我们反馈, 让我们共同携手, 将知识准确、高效且有效地传递给每一个人。

同时, 如果您在日常生活、工作和学习中遇到有价值有营养的知识文档, 欢迎分享到 书栈(BookStack.CN), 为知识的传承献上您的一份力量!

如果当前文档生成时间太久, 请到 书栈(BookStack.CN) 获取最新的文档, 以跟上知识更新换代的步伐。

文档地址: <http://www.bookstack.cn/books/spring-cloud-tutorial>

书栈官网: <http://www.bookstack.cn>

书栈开源: <https://github.com/TruthHun>

分享, 让知识传承更久远! 感谢知识的创造者, 感谢知识的分享者, 也感谢每一位阅读到此处的读者, 因为我们都将成为知识的传承者。



## 介绍

- [Spring Cloud Tutorial 《Spring Cloud 教程》](#)
  - [涉及技术](#)
  - [Get Started 如何开始阅读](#)
  - [Code 源码](#)
  - [Issue 意见、建议](#)
  - [联系作者：](#)
  - [相关资料](#)
  - [其他书籍](#)
  - [开源捐赠](#)
  - [来源\(书栈小编注\)](#)

## Spring Cloud Tutorial 《Spring Cloud 教程》

---



Spring Cloud Tutorial takes you to learn Spring Cloud step by step with a large number of samples. There is also a GitBook version of the book:  
<http://www.gitbook.com/book/waylau/spring-cloud-tutorial>.

Let's [READ!](#)

Spring Cloud Tutorial 是一本关于 Spring Cloud 学习的开

源书。利用业余时间写了本书，图文并茂，用大量实例带你一步一步走进 Spring Cloud 的世界。如有疏漏欢迎指正，欢迎提问。感谢您的参与！

注：Spring Cloud 基于 Spring Boot 来进行构建服务。有关 Spring Boot 的内容，可以参阅笔者所著的《Spring Boot 教程》(<https://github.com/waylau/spring-boot-tutorial>)

## 涉及技术

---

本书涉及的相关技术及版本如下。

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Boot Data Redis Starter 2.0.0.M3
- Apache HttpClient 4.5.3
- Redis 3.2.100
- Spring Cloud Netflix Eureka Server Finchley.M2
- Spring Cloud Netflix Eureka Client Finchley.M2
- Spring Cloud Starter OpenFeign Finchley.M2
- Spring Cloud Config Server Finchley.M2
- Spring Cloud Config Client Finchley.M2

## Get Started 如何开始阅读

---

选择下面入口之一：

- <https://github.com/waylau/spring-cloud-tutorial/> 的 SUMMARY.md (源码)
- <http://waylau.gitbooks.io/spring-cloud-tutorial/>

点击 Read 按钮（同步更新，国内访问速度一般）

- <http://git.oschina.net/waylau/spring-cloud-tutorial> 的 SUMMARY.md（码云，手动同步，有所延时）

## Code 源码

---

书中所有示例源码，移步至

<https://github.com/waylau/spring-cloud-tutorial>

[samples](#)

目录下

## Issue 意见、建议

---

如有勘误、意见或建议欢迎拍砖

<https://github.com/waylau/spring-cloud-tutorial/issues>

## 联系作者：

---

您也可以直接联系我：

- 博客：<https://waylau.com>
- 邮箱：[waylau521\(at\)gmail.com](mailto:waylau521@gmail.com)
- 微博：<http://weibo.com/waylau521>
- 开源：<https://github.com/waylau>

## 相关资料

---

其他与 Spring Cloud 相关的学习资料还有：

- 基于Spring Boot的博客系统实战（视频）：<http://coding.imooc.com/class/125.html>

- Spring Boot 教程：<https://github.com/waylau/spring-boot-tutorial>
- 基于Spring Cloud的微服务实战（视频）：<https://coding.imooc.com/class/177.html>

## 其他书籍

---

若您对本书不感冒，笔者还写了其他方面的超过一打的书籍（可见<https://waylau.com/books/>），多是开源电子书。

本人也维护了一个[books-collection](#)项目，里面提供了优质的专门给程序员的开源、免费图书集合。

## 开源捐赠

---



捐赠所得所有款项将用于开源事业！

## 来源(书栈小编注)

---

<https://github.com/waylau/spring-cloud-tutorial>





## 开发环境

- [第一个微服务实现——天气预报服务](#)
  - [开发环境](#)
  - [数据来源](#)
  - [初始化一个 Spring Boot 项目](#)
  - [项目配置](#)
  - [创建天气信息相关的值对象](#)
  - [服务接口及实现](#)
  - [控制器层](#)
  - [配置类](#)
  - [访问API](#)
  - [源码](#)

## 第一个微服务实现——天气预报服务

---

本章节，我们将基于 Spring Boot 技术来实现我们的第一个微服务天气预报应用——micro-weather-basic。micro-weather-basic 的作用是实现简单的天气预报功能，可以根据不同的城市，查询该城市的实时天气情况。

## 开发环境

---

- Gradle 4.0
- Spring Boot 1.5.6
- Apache HttpClient 4.5.3

## 数据来源

---

理论上，天气的数据是天气预报的实现基础。本应用与实际的天气数据无关，理论上，可以兼容多种数据来源。但为求简单，我们在网上找了一个免费、可用的天气数据接口。

- 天气数据来源为中华万年历。例如：
  - 通过城市名字获得天气数据：
   
[http://wthrcdn.etouch.cn/weather\\_mini?city=深圳](http://wthrcdn.etouch.cn/weather_mini?city=深圳)
  - 通过城市id获得天气数据：
   
[http://wthrcdn.etouch.cn/weather\\_mini?citykey=101280601](http://wthrcdn.etouch.cn/weather_mini?citykey=101280601)
- 城市ID列表。每个城市都有一个唯一的ID作为标识。见
   
<http://cj.weather.com.cn/support/Detail.aspx?id=51837fba1b35fe0f8411b6df> 或者
   
<http://mobile.weather.com.cn/js/citylist.xml>。

调用天气服务接口示例，我们以“深圳”城市为例，可用看到如下天气数据返回。

```

1.  {
2.      "data": {
3.          "yesterday": {
4.              "date": "1日星期五",
5.              "high": "高温 33°C",
6.              "fx": "无持续风向",
7.              "low": "低温 26°C",
8.              "fl": "<![CDATA[<3级]]>",
9.              "type": "多云"
10.         },
11.         "city": "深圳",
12.         "aqi": "72",
13.         "forecast": [
14.             {
  
```

```

15.         "date": "2日星期六",
16.         "high": "高温 32℃",
17.         "fengli": "<![CDATA[<3级]]>",
18.         "low": "低温 26℃",
19.         "fengxiang": "无持续风向",
20.         "type": "阵雨"
21.     },
22.     {
23.         "date": "3日星期天",
24.         "high": "高温 29℃",
25.         "fengli": "<![CDATA[5-6级]]>",
26.         "low": "低温 26℃",
27.         "fengxiang": "无持续风向",
28.         "type": "大雨"
29.     },
30.     {
31.         "date": "4日星期一",
32.         "high": "高温 29℃",
33.         "fengli": "<![CDATA[3-4级]]>",
34.         "low": "低温 26℃",
35.         "fengxiang": "西南风",
36.         "type": "暴雨"
37.     },
38.     {
39.         "date": "5日星期二",
40.         "high": "高温 31℃",
41.         "fengli": "<![CDATA[<3级]]>",
42.         "low": "低温 27℃",
43.         "fengxiang": "无持续风向",
44.         "type": "阵雨"
45.     },
46.     {
47.         "date": "6日星期三",
48.         "high": "高温 32℃",
49.         "fengli": "<![CDATA[<3级]]>",
50.         "low": "低温 27℃",
51.         "fengxiang": "无持续风向",
52.         "type": "阵雨"

```

```
53.         }
54.     ],
55.     "ganmao": "风较大，阴冷潮湿，较易发生感冒，体质较弱的朋友请注意适当防
    护。",
56.     "wendu": "29"
57. },
58.     "status": 1000,
59.     "desc": "OK"
60. }
```

我们通过观察数据，来了解每个返回字段的含义。

- “city”：城市名称
- “aqi”：空气指数，
- “wendu”：实时温度
- “date”：日期，包含未来5天
- “high”：最高温度
- “low”：最低温度
- “fengli”：风力
- “fengxiang”：风向
- “type”：天气类型

以上数据，是我们需要的天气数据的核心数据，但是，同时也要关注下面两个字段：

- “status”：接口调用的返回状态，返回值“1000”，意味着数据是接口正常
- “desc”：接口状态的描述，“OK”代表接口正常

重点关注返回值不是“1000”的情况，说明，这个接口调用异常了。

## 初始化一个 Spring Boot 项目

初始化一个 Spring Boot 项目 `micro-weather-basic`，该项目可以直接在我们之前章节课程中的 `basic-gradle` 项目基础进行修改。同时，为了优化项目的构建速度，我们对Maven中央仓库地址和 Gradle Wrapper 地址做了调整。其中细节暂且不表，读者可以自行参阅源码，或者学习笔者所著的《Spring Boot 教程》（<https://github.com/waylau/spring-boot-tutorial>）。其原理，我也整理到我的博客中了：

- <https://waylau.com/change-gradle-wrapper-distribution-url-to-local-file/>
- <https://waylau.com/use-maven-mirrors/>

## 项目配置

添加 Apache HttpClient 的依赖，来作为我们Web请求的客户端。

```
1. // 依赖关系
2. dependencies {
3.     //...
4.
5.     // 添加 Apache HttpClient 依赖
6.     compile('org.apache.httpcomponents:httpclient:4.5.3')
7.
8.     //...
9. }
```

## 创建天气信息相关的值对象

创建 `com.waylau.spring.cloud.vo` 包，用于相关值对象。创建天气信息类 Weather

```
1. public class Weather implements Serializable {
```

```
2.
3.     private static final long serialVersionUID = 1L;
4.
5.     private String city;
6.     private String aqi;
7.     private String wendu;
8.     private String ganmao;
9.     private Yesterday yesterday;
10.    private List<Forecast> forecast;
11.
12.    public String getCity() {
13.        return city;
14.    }
15.    public void setCity(String city) {
16.        this.city = city;
17.    }
18.    public String getAqi() {
19.        return aqi;
20.    }
21.    public void setAqi(String aqi) {
22.        this.aqi = aqi;
23.    }
24.    public String getWendu() {
25.        return wendu;
26.    }
27.    public void setWendu(String wendu) {
28.        this.wendu = wendu;
29.    }
30.    public String getGanmao() {
31.        return ganmao;
32.    }
33.    public void setGanmao(String ganmao) {
34.        this.ganmao = ganmao;
35.    }
36.    public Yesterday getYesterday() {
37.        return yesterday;
38.    }
39.    public void setYesterday(Yesterday yesterday) {
```

```
40.         this.yesterday = yesterday;
41.     }
42.     public List<Forecast> getForecast() {
43.         return forecast;
44.     }
45.     public void setForecast(List<Forecast> forecast) {
46.         this.forecast = forecast;
47.     }
48. }
```

## 昨日天气信息：

```
1.  public class Yesterday implements Serializable {
2.
3.     private static final long serialVersionUID = 1L;
4.
5.     private String date;
6.     private String high;
7.     private String fx;
8.     private String low;
9.     private String fl;
10.    private String type;
11.
12.    public Yesterday() {
13.
14.    }
15.
16.    public String getDate() {
17.        return date;
18.    }
19.
20.    public void setDate(String date) {
21.        this.date = date;
22.    }
23.
24.    public String getHigh() {
25.        return high;
26.    }
```



```
27.  
28.     public void setHigh(String high) {  
29.         this.high = high;  
30.     }  
31.  
32.     public String getFx() {  
33.         return fx;  
34.     }  
35.  
36.     public void setFx(String fx) {  
37.         this.fx = fx;  
38.     }  
39.  
40.     public String getLow() {  
41.         return low;  
42.     }  
43.  
44.     public void setLow(String low) {  
45.         this.low = low;  
46.     }  
47.  
48.     public String getFl() {  
49.         return fl;  
50.     }  
51.  
52.     public void setFl(String fl) {  
53.         this.fl = fl;  
54.     }  
55.  
56.     public String getType() {  
57.         return type;  
58.     }  
59.  
60.     public void setType(String type) {  
61.         this.type = type;  
62.     }  
63.  
64. }
```

## 未来天气信息：

```
1. public class Forecast implements Serializable {
2.
3.     private static final long serialVersionUID = 1L;
4.
5.     private String date;
6.     private String high;
7.     private String fengxiang;
8.     private String low;
9.     private String fengli;
10.    private String type;
11.
12.    public String getDate() {
13.        return date;
14.    }
15.
16.    public void setDate(String date) {
17.        this.date = date;
18.    }
19.
20.    public String getHigh() {
21.        return high;
22.    }
23.
24.    public void setHigh(String high) {
25.        this.high = high;
26.    }
27.
28.    public String getFengxiang() {
29.        return fengxiang;
30.    }
31.
32.    public void setFengxiang(String fengxiang) {
33.        this.fengxiang = fengxiang;
34.    }
35.
36.    public String getLow() {
```

```
37.         return low;
38.     }
39.
40.     public void setLow(String low) {
41.         this.low = low;
42.     }
43.
44.     public String getFengli() {
45.         return fengli;
46.     }
47.
48.     public void setFengli(String fengli) {
49.         this.fengli = fengli;
50.     }
51.
52.     public String getType() {
53.         return type;
54.     }
55.
56.     public void setType(String type) {
57.         this.type = type;
58.     }
59.
60.     public Forecast() {
61.
62.     }
63.
64. }
```

## WeatherResponse 作为整个消息的返回对象

```
1. public class WeatherResponse implements Serializable {
2.
3.     private static final long serialVersionUID = 1L;
4.
5.     private Weather data; // 消息数据
6.     private String status; // 消息状态
7.     private String desc; // 消息描述
```

```
8.
9.     public Weather getData() {
10.         return data;
11.     }
12.
13.     public void setData(Weather data) {
14.         this.data = data;
15.     }
16.
17.     public String getStatus() {
18.         return status;
19.     }
20.
21.     public void setStatus(String status) {
22.         this.status = status;
23.     }
24.
25.     public String getDesc() {
26.         return desc;
27.     }
28.
29.     public void setDesc(String desc) {
30.         this.desc = desc;
31.     }
32.
33. }
```

## 服务接口及实现

定义了获取服务的两个接口方法

```
1. public interface WeatherDataService {
2.
3.     /**
4.      * 根据城市ID查询天气数据
5.      * @param cityId
6.      * @return
```

```

7.      */
8.      WeatherResponse getDataByCityId(String cityId);
9.
10.     /**
11.      * 根据城市名称查询天气数据
12.      * @param cityId
13.      * @return
14.      */
15.     WeatherResponse getDataByCityName(String cityName);
16. }

```

其实现为：

```

1. @Service
2. public class WeatherDataServiceImpl implements WeatherDataService {
3.
4.     @Autowired
5.     private RestTemplate restTemplate;
6.
7.     private final String WEATHER_API =
8.         "http://wthrcdn.etouch.cn/weather_mini";
9.
10.    @Override
11.    public WeatherResponse getDataByCityId(String cityId) {
12.        String uri = WEATHER_API + "?citykey=" + cityId;
13.        return this.doGetWeatherData(uri);
14.    }
15.
16.    @Override
17.    public WeatherResponse getDataByCityName(String cityName) {
18.        String uri = WEATHER_API + "?city=" + cityName;
19.        return this.doGetWeatherData(uri);
20.    }
21.
22.    private WeatherResponse doGetWeatherData(String uri) {
23.        ResponseEntity<String> response =
24.            restTemplate.getForEntity(uri, String.class);
25.        String strBody = null;
26.    }
27. }

```

```

24.
25.         if (response.getStatusCodeValue() == 200) {
26.             strBody = response.getBody();
27.         }
28.
29.         ObjectMapper mapper = new ObjectMapper();
30.         WeatherResponse weather = null;
31.
32.         try {
33.             weather = mapper.readValue(strBody,
WeatherResponse.class);
34.         } catch (IOException e) {
35.             e.printStackTrace();
36.         }
37.
38.         return weather;
39.     }
40.
41. }

```

返回的天气信息采用了 Jackson 来进行反序列化成为 WeatherResponse 对象。

## 控制器层

控制器层暴露了RESTful API 地址。

```

1. @RestController
2. @RequestMapping("/weather")
3. public class WeatherController {
4.
5.     @Autowired
6.     private WeatherDataService weatherDataService;
7.
8.     @GetMapping("/cityId/{cityId}")
9.     public WeatherResponse
getReportByCityId(@PathVariable("cityId") String cityId) {

```

```

10.         return weatherDataService.getDataByCityId(cityId);
11.     }
12.
13.     @GetMapping("/cityName/{cityName}")
14.     public WeatherResponse
15.         getReportByCityName(@PathVariable("cityName") String cityName) {
16.         return weatherDataService.getDataByCityName(cityName);
17.     }
18. }

```

`@RestController` 自动会将返回的数据，序列化成 JSON数据格式。

## 配置类

`RestConfiguration` 是 `RestTemplate` 的配置类。

```

1. @Configuration
2. public class RestConfiguration {
3.
4.     @Autowired
5.     private RestTemplateBuilder builder;
6.
7.     @Bean
8.     public RestTemplate restTemplate() {
9.         return builder.build();
10.    }
11.
12. }

```

## 访问API

运行项目之后，访问项目的 API：

- <http://localhost:8080/weather/cityId/101280601>

- <http://localhost:8080/weather/cityName/惠州>

能看到如下的数据返回

The screenshot shows the HttpRequster application interface. The 'REQUEST' tab is active, displaying a GET request to the URL `http://localhost:8080/weather/cityName/%E6%83%A0%E5%B7%9E`. The 'RESPONSE' tab shows the JSON response from the server, which includes weather data for HuiZhou (惠州). The response is a JSON object with fields like 'city', 'aqi', 'wendu', 'ganmao', 'yesterday', 'forecast', 'status', and 'desc'. The 'HEADERS' section shows 'Content-Type: application/json; charset=UTF-8', 'Date: Tue, 05 Sep 2017 15:19:34 GMT', and 'Transfer-Encoding: chunked'. The 'HISTORY' table at the bottom shows the request details.

Request	Response	Date	Size	Time
GET http://localhost:8080/weather/cityName/%E6%83%A0%E5%B7%9E	200	Sep 5 2017 - 11:19:34 PM	777 B	269 ms

## 源码

本章节的源码，在 `micro-weather-basic` 目录下。



## Spring Cloud 介绍

- [Spring Cloud 介绍](#)

## Spring Cloud 介绍

---

- [Spring Cloud 简介](#)
- [Spring Cloud 入门配置](#)
- [Spring Cloud 的子项目介绍](#)
- [Spring Cloud 与 Spring Boot 的关系](#)

# Spring Cloud 简介

- [Spring Cloud 简介](#)

## Spring Cloud 简介

---

[Spring Cloud](#) 为开发人员提供了快速构建分布式系统中一些常见模式的工具，例如：

- 配置管理
- 服务注册与发现
- 断路器
- 智能路由
- 服务间调用
- 负载均衡
- 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
- 分布式消息
- .....

使用 Spring Cloud 开发人员可以开箱即用的实现这些模式的服务和应用程序。这些服务可以任何环境下运行，包括分布式环境，也包括开发人员自己的笔记本电脑、裸机数据中心，以及 Cloud Foundry 等托管平台。

Spring Cloud 基于 Spring Boot 来进行构建服务。并可以将轻松集成第三方类库，增强应用程序的行为。您可以利用基本的默认行为快速入门，然后在需要时，您可以配置或扩展以创建自定义解决方案。

注：有关 Spring Boot 的内容，可以参阅笔者所著的《Spring Boot 教程》(<https://github.com/waylau/spring-boot-tutorial>)

# Spring Cloud 入门配置

- [Spring Cloud 入门配置](#)
  - [Maven 配置](#)
  - [Gradle 配置](#)
  - [声明式方法](#)
  - [自动生成项目](#)
  - [源码](#)

## Spring Cloud 入门配置

在项目中开始使用 Spring Cloud 的推荐方法是使用依赖关系管理系统，比如，使用 Maven 或者 Gradle 构建。

## Maven 配置

以下是一个基本的 Spring Boot 项目的基本 Maven 配置：

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
4.     <modelVersion>4.0.0</modelVersion>
5.
6.     <groupId>com.waylau.spring</groupId>
7.     <artifactId>cloud</artifactId>
8.     <version>0.0.1-SNAPSHOT</version>
9.     <packaging>jar</packaging>
10.
11.     <name>basic-maven</name>
12.     <description>Basic project for Maven</description>
13.
```

```
14.     <parent>
15.         <groupId>org.springframework.boot</groupId>
16.         <artifactId>spring-boot-starter-parent</artifactId>
17.         <version>1.5.6.RELEASE</version>
18.         <relativePath/> <!-- lookup parent from repository -->
19.     </parent>
20.
21.     <properties>
22.         <project.build.sourceEncoding>UTF-
23. 8</project.build.sourceEncoding>
24.         <project.reporting.outputEncoding>UTF-
25. 8</project.reporting.outputEncoding>
26.         <java.version>1.8</java.version>
27.     </properties>
28.
29.     <dependencies>
30.         <dependency>
31.             <groupId>org.springframework.boot</groupId>
32.             <artifactId>spring-boot-starter-web</artifactId>
33.         </dependency>
34.
35.         <dependency>
36.             <groupId>org.springframework.boot</groupId>
37.             <artifactId>spring-boot-starter-test</artifactId>
38.             <scope>test</scope>
39.         </dependency>
40.     </dependencies>
41.
42.     <build>
43.         <plugins>
44.             <plugin>
45.                 <groupId>org.springframework.boot</groupId>
46.                 <artifactId>spring-boot-maven-plugin</artifactId>
47.             </plugin>
48.         </plugins>
49.     </build>
```

```
50. </project>
```

在此基础上，您可以按需添加不同的依赖，以使您的应用程序增强功能。

## Gradle 配置

以下是一个基本的 Spring Boot 项目的基本 Maven 配置：

```
1. buildscript {
2.     ext {
3.         springBootVersion = '1.5.6.RELEASE'
4.     }
5.     repositories {
6.         mavenCentral()
7.     }
8.     dependencies {
9.         classpath("org.springframework.boot:spring-boot-gradle-
10.             plugin:${springBootVersion}")
11.     }
12.
13. apply plugin: 'java'
14. apply plugin: 'eclipse'
15. apply plugin: 'org.springframework.boot'
16.
17. version = '0.0.1-SNAPSHOT'
18. sourceCompatibility = 1.8
19.
20. repositories {
21.     mavenCentral()
22. }
23.
24.
25. dependencies {
26.     compile('org.springframework.boot:spring-boot-starter-web')
27.     testCompile('org.springframework.boot:spring-boot-starter-
```

```
        test')  
28.    }
```

在此基础上，您可以按需添加不同的依赖，以使您的应用程序增强功能。

## 声明式方法

Spring Cloud 采用声明的方法，通常只需要一个类路径更改和/或注释即可获得很多功能。下面是 Spring Boot 最简单的应用程序示例：

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

## 自动生成项目

Spring 官方提供了基于 Web 的 Spring Initializr 项目，用于 Spring 项目的快速生成。作为项目的初始化，采用 Spring Initializr 是个不错的选择。访问 <http://start.spring.io>，填入相关的项目信息，选择合适的依赖，即可实现项目源码的下载。而这个过程，你无需关心你的依赖是如何来管理的，Spring Initializr 为你准备好了一切。

start.spring.io

# SPRING INITIALIZR

bootstrap your application now

Generate a Maven Project with Java and Spring Boot 1.5.6

## Project Metadata

Artifact coordinates

**Group**

**Artifact**

## Dependencies

Add Spring Boot Starters and dependencies to your application

**Search for dependencies**

**Selected Dependencies**

Web

**Generate Project** alt + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)

## 源码

本章节源码，见 basic-maven 和 basic-gradle 。



## Spring Cloud 的子项目介绍

- Spring Cloud 的子项目介绍
  - Spring Cloud Config
  - Spring Cloud Netflix
  - Spring Cloud Bus
  - Spring Cloud for Cloud Foundry
  - Spring Cloud Cloud Foundry Service Broker
  - Spring Cloud Cluster
  - Spring Cloud Consul
  - Spring Cloud Security
  - Spring Cloud Sleuth
  - Spring Cloud Data Flow
  - Spring Cloud Stream
  - Spring Cloud Stream App Starters
  - Spring Cloud Task
  - Spring Cloud Task App Starters
  - Spring Cloud Zookeeper
  - Spring Cloud for Amazon Web Services
  - Spring Cloud Connectors
  - Spring Cloud Starters
  - Spring Cloud CLI
  - Spring Cloud Contract

## Spring Cloud 的子项目介绍

---

Spring Cloud 由以下子项目组成。

## Spring Cloud Config

---

Centralized external configuration management backed by a git repository. The configuration resources map directly to Spring `Environment` but could be used by non-Spring applications if desired.

项目地址为: <http://cloud.spring.io/spring-cloud-config>。

## Spring Cloud Netflix

---

Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).

项目地址为: <http://cloud.spring.io/spring-cloud-netflix>。

## Spring Cloud Bus

---

An event bus for linking services and service instances together with distributed messaging. Useful for propagating state changes across a cluster (e.g. config change events).

项目地址为: <http://cloud.spring.io/spring-cloud-bus>。

## Spring Cloud for Cloud Foundry

---

Integrates your application with Pivotal Cloudfoundry. Provides a service discovery implementation and also makes it easy to implement

SSO and OAuth2 protected resources, and also to create a Cloudfoundry service broker.

项目地址为: <http://cloud.spring.io/spring-cloud-cloudfoundry>。

## Spring Cloud Cloud Foundry Service Broker

---

Provides a starting point for building a service broker that manages a Cloud Foundry managed service.

项目地址为: <http://cloud.spring.io/spring-cloud-cloudfoundry-service-broker/>。

## Spring Cloud Cluster

---

Leadership election and common stateful patterns with an abstraction and implementation for Zookeeper, Redis, Hazelcast, Consul.

项目地址为: <http://projects.spring.io/spring-cloud>。

## Spring Cloud Consul

---

Service discovery and configuration management with Hashicorp Consul.

项目地址为: <http://cloud.spring.io/spring-cloud-consul>。

## Spring Cloud Security

---

---

Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.

项目地址为: <http://cloud.spring.io/spring-cloud-security>。

## Spring Cloud Sleuth

---

Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.

项目地址为: <http://cloud.spring.io/spring-cloud-sleuth>。

## Spring Cloud Data Flow

---

A cloud-native orchestration service for composable microservice applications on modern runtimes. Easy-to-use DSL, drag-and-drop GUI, and REST-APIs together simplifies the overall orchestration of microservice based data pipelines.

项目地址为: <http://cloud.spring.io/spring-cloud-dataflow>。

## Spring Cloud Stream

---

A lightweight event-driven microservices framework

to quickly build applications that can connect to external systems. Simple declarative model to send and receive messages using Apache Kafka or RabbitMQ between Spring Boot apps.

项目地址为: <http://cloud.spring.io/spring-cloud-stream>。

## Spring Cloud Stream App Starters

---

Spring Cloud Stream App Starters are Spring Boot based Spring Integration applications that provide integration with external systems.

项目地址为: <http://cloud.spring.io/spring-cloud-stream-app-starters>。

## Spring Cloud Task

---

A short-lived microservices framework to quickly build applications that perform finite amounts of data processing. Simple declarative for adding both functional and non-functional features to Spring Boot apps.

项目地址为: <http://cloud.spring.io/spring-cloud-task>。

## Spring Cloud Task App Starters

---

Spring Cloud Task App Starters are Spring Boot applications that may be any process including

Spring Batch jobs that do not run forever, and they end/stop after a finite period of data processing.

项目地址为: <http://cloud.spring.io/spring-cloud-config>。

## Spring Cloud Zookeeper

---

Service discovery and configuration management with Apache Zookeeper.

项目地址为: <http://cloud.spring.io/spring-cloud-task-app-starters>。

## Spring Cloud for Amazon Web Services

---

Easy integration with hosted Amazon Web Services. It offers a convenient way to interact with AWS provided services using well-known Spring idioms and APIs, such as the messaging or caching API. Developers can build their application around the hosted services without having to care about infrastructure or maintenance.

项目地址为: <https://cloud.spring.io/spring-cloud-aws>。

## Spring Cloud Connectors

---

Makes it easy for PaaS applications in a variety of platforms to connect to backend services like databases and message brokers (the project formerly

known as “Spring Cloud”).

项目地址为: <http://cloud.spring.io/spring-cloud-config>。

## Spring Cloud Starters

---

Spring Boot-style starter projects to ease dependency management for consumers of Spring Cloud. (Discontinued as a project and merged with the other projects after Angel.SR2.)

项目地址为: <https://cloud.spring.io/spring-cloud-connectors>。

## Spring Cloud CLI

---

Spring Boot CLI plugin for creating Spring Cloud component applications quickly in Groovy

项目地址为: <https://github.com/spring-cloud/spring-cloud-cli>。

## Spring Cloud Contract

---

Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach.

项目地址为: <http://cloud.spring.io/spring-cloud-contract>。





## Spring Cloud 与 Spring Boot 的关系

- [Spring Cloud 与 Spring Boot 的关系](#)

## Spring Cloud 与 Spring Boot 的关系

---

Spring Boot 是构建 Spring Cloud 架构的基石，是一种快速启动项目的方式。

Spring Cloud 对应于 Spring Boot 版本，由以下依赖关系。

Finchley builds and works with Spring Boot 2.0.x, and is not expected to work with Spring Boot 1.5.x.

The Dalston and Edgware release trains build on Spring Boot 1.5.x, and are not expected to work with Spring Boot 2.0.x.

The Camden release train builds on Spring Boot 1.4.x, but is also tested with 1.5.x.

The Brixton release train builds on Spring Boot 1.3.x, but is also tested with 1.4.x.

The Angel release train builds on Spring Boot 1.2.x, and is incompatible in some areas with Spring Boot 1.3.x. Brixton builds on Spring Boot 1.3.x and is similarly incompatible with 1.2.x. Some libraries and most apps built on Angel will run fine on Brixton, but changes will be required anywhere that the OAuth2 features from spring-cloud-security 1.0.x are used (they were mostly moved to Spring Boot in

### 1.3.0).

Use your dependency management tools to control the version. If you are using Maven remember that the first version declared wins, so declare the BOMs in order, with the first one usually being the most recent (e.g. if you want to use Spring Boot 1.3.6 with Brixton.RELEASE, put the Boot BOM first). The same rule applies to Gradle if you use the Spring dependency management plugin.

NOTE: The release train contains a `spring-cloud-dependencies` as well as the `spring-cloud-starter-parent`. You can use the parent as you would the `spring-boot-starter-parent` (if you are using Maven). If you only need dependency management, the “dependencies” version is a BOM-only version of the same thing (it just contains dependency management and no plugin declarations or direct references to Spring or Spring Boot). If you are using the Spring Boot parent POM, then you can use the BOM from Spring Cloud. The opposite is not true: using the Cloud parent makes it impossible, or at least unreliable, to also use the Boot BOM to change the version of Spring Boot and its dependencies.

# 微服务 与 Spring Boot

- [微服务 与 Spring Boot](#)

## 微服务 与 Spring Boot

---

- [微服务架构概述](#)
- [微服务 API 设计原则](#)
- [微服务存储设计原则](#)
- [提升微服务的并发访问能力](#)

# 微服务架构概述

- [微服务架构概述](#)

## 微服务架构概述

---

# 微服务 **API** 设计原则

- [微服务 API 设计原则](#)

## 微服务 API 设计原则

---

# 微服务存储设计原则

- [微服务存储设计原则](#)

# 微服务存储设计原则

---

# 提升微服务的并发访问能力

- 提升微服务的并发访问能力
  - 开发环境
  - 项目配置
  - 下载安装、运行 Redis
  - 修改 WeatherDataServiceImpl
  - 运行
  - 源码

## 提升微服务的并发访问能力

有时，为了提升整个网站的性能，我们会将经常需要访问数据缓存起来，这样，在下次查询的时候，能快速的找到这些数据。

缓存的使用与系统的时效性有着非常大的关系。当我们的系统时效性要求不高时，则选择使用缓存是极好的。当，系统要求的时效性比较高时，则并不适合用缓存。

本章节，我们将演示如何通过集成Redis服务器来进行数据的缓存，以提高微服务的并发访问能力。

天气数据接口，本身时效性不是很高，而且又因为是Web服务，在调用过程中，本身是存在延时的。所以，采用缓存，一方面可以有效减轻访问天气接口服务带来的延时问题，另一方面，也可以减轻天气接口的负担，提高并发访问量。

在 `micro-weather-basic` 的基础上，我们构建了一个 `micro-weather-redis` 项目，作为示例。

## 开发环境

- Spring Data Redis 1.5.6
- Redis 3.2.100 :  
<https://github.com/MicrosoftArchive/redis/releases>

## 项目配置

添加 Spring Data Redis 的依赖。

```
1. // 依赖关系
2. dependencies {
3.     //...
4.
5.     // 添加 Spring Data Redis 依赖
6.     compile('org.springframework.boot:spring-boot-starter-data-redis')
7.
8.     //...
9. }
```

## 下载安装、运行 Redis

安装后，默认运行在 `127.0.0.1:6379` 地址端口。

## 修改 WeatherDataServiceImpl

增加了 StringRedisTemplate 用于操作 Redis。

```
1. @Service
2. public class WeatherDataServiceImpl implements WeatherDataService {
3.
4.     @Autowired
5.     private RestTemplate restTemplate;
```



```

6.
7.     @Autowired
8.     private StringRedisTemplate stringRedisTemplate;
9.
10.    private final String WEATHER_API =
    "http://wthrcdn.etouch.cn/weather_mini";
11.
12.    @Override
13.    public WeatherResponse getDataByCityId(String cityId) {
14.        String uri = WEATHER_API + "?citykey=" + cityId;
15.        return this.doGetWeatherData(uri);
16.    }
17.
18.    @Override
19.    public WeatherResponse getDataByCityName(String cityName) {
20.        String uri = WEATHER_API + "?city=" + cityName;
21.        return this.doGetWeatherData(uri);
22.    }
23.
24.    /**
25.     * 获取天气数据
26.     * @param uri
27.     * @return
28.     */
29.    private WeatherResponse doGetWeatherData(String uri) {
30.
31.        ValueOperations<String, String> ops =
    this.stringRedisTemplate.opsForValue();
32.        String key = uri;
33.        WeatherResponse weather = null;
34.        String strBody = null;
35.
36.        if (!this.stringRedisTemplate.hasKey(key)) {
37.            System.out.println("Not Found key " + key);
38.
39.            ResponseEntity<String> response =
    restTemplate.getForEntity(uri, String.class);
40.

```

```

41.
42.         if (response.getStatusCodeValue() == 200) {
43.             strBody = response.getBody();
44.         }
45.
46.         ops.set(key, strBody, 10L, TimeUnit.SECONDS);
47.     } else {
48.         System.out.println("Found key " + key + ", value=" +
ops.get(key));
49.
50.         strBody = ops.get(key);
51.     }
52.
53.     ObjectMapper mapper = new ObjectMapper();
54.
55.     try {
56.         weather = mapper.readValue(strBody,
WeatherResponse.class);
57.     } catch (IOException e) {
58.         e.printStackTrace();
59.     }
60.
61.     return weather;
62. }
63.
64. }

```

修改了 doGetWeatherData 方法，增加了 Redis 数据的判断。

- 当存在某个key（天气接口的uri，是唯一代表某个地区的天气数据）时，我们就从 Redis 里面拿缓存数据；
- 当不存在某个key（没有初始化数据或者数据过期了），从去天气接口里面去取最新的数据，并初始化到 Redis 中；
- 由于天气数据更新频率的特点（基本上一个小时或者半个小时更新一次），或者，我们在Redis里面设置了 30分钟的超时时间。

# 运行

多次访问某个天气接口时，来测试效果。

1. Not Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`
2. Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`,  
value={"data":{"yesterday":{"date":"5日星期二","high":"高温 32°C","fx":"无持续风向","low":"低温 27°C","fl":"<![CDATA[<3级]]>","type":"中雨"},"city":"深圳","aqi":"34","forecast":[{"date":"6日星期三","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"中雨"}, {"date":"7日星期四","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"小雨"}, {"date":"8日星期五","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 27°C","fengxiang":"无持续风向","type":"雷阵雨"}, {"date":"9日星期六","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 27°C","fengxiang":"无持续风向","type":"多云"}, {"date":"10日星期天","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"多云"}]}, "ganmao":"各项气象条件适宜，发生感冒机率较低。但请避免长期处于空调房间中，以防感冒。", "wendu":"27"}, "status":1000, "desc":"OK"}
3. Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`,  
value={"data":{"yesterday":{"date":"5日星期二","high":"高温 32°C","fx":"无持续风向","low":"低温 27°C","fl":"<![CDATA[<3级]]>","type":"中雨"},"city":"深圳","aqi":"34","forecast":[{"date":"6日星期三","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"中雨"}, {"date":"7日星期四","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"小雨"}, {"date":"8日星期五","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 27°C","fengxiang":"无持续风向","type":"雷阵雨"}, {"date":"9日星期六","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 27°C","fengxiang":"无持续风向","type":"多云"}, {"date":"10日星期天","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温 26°C","fengxiang":"无持续风向","type":"多云"}]}, "ganmao":"各项气象条件适宜，发生感冒机率较低。但请避免长期处于空调房间中，以防感冒。", "wendu":"27"}, "status":1000, "desc":"OK"}

4. Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`,  
value={"data":{"yesterday":{"date":"5日星期二","high":"高温  
32°C","fx":"无持续风向","low":"低温 27°C","fl":"<![CDATA[<3  
级]]>","type":"中雨"},"city":"深圳","aqi":"34","forecast":[{"date":"6  
日星期三","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"中雨"}, {"date":"7日星期  
四","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"小雨"}, {"date":"8日星期  
五","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
27°C","fengxiang":"无持续风向","type":"雷阵雨"}, {"date":"9日星期  
六","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
27°C","fengxiang":"无持续风向","type":"多云"}, {"date":"10日星期  
天","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"多云"}],"ganmao":"各项气象条件适  
宜，发生感冒机率较低。但请避免长期处于空调房间中，以防感  
冒。","wendu":"27"},"status":1000,"desc":"OK"}
5. Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`,  
value={"data":{"yesterday":{"date":"5日星期二","high":"高温  
32°C","fx":"无持续风向","low":"低温 27°C","fl":"<![CDATA[<3  
级]]>","type":"中雨"},"city":"深圳","aqi":"34","forecast":[{"date":"6  
日星期三","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"中雨"}, {"date":"7日星期  
四","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"小雨"}, {"date":"8日星期  
五","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
27°C","fengxiang":"无持续风向","type":"雷阵雨"}, {"date":"9日星期  
六","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
27°C","fengxiang":"无持续风向","type":"多云"}, {"date":"10日星期  
天","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"多云"}],"ganmao":"各项气象条件适  
宜，发生感冒机率较低。但请避免长期处于空调房间中，以防感  
冒。","wendu":"27"},"status":1000,"desc":"OK"}
6. Found key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`,  
value={"data":{"yesterday":{"date":"5日星期二","high":"高温  
32°C","fx":"无持续风向","low":"低温 27°C","fl":"<![CDATA[<3  
级]]>","type":"中雨"},"city":"深圳","aqi":"34","forecast":[{"date":"6  
日星期三","high":"高温 32°C","fengli":"<![CDATA[<3级]]>","low":"低温  
26°C","fengxiang":"无持续风向","type":"中雨"}, {"date":"7日星期

```
四", "high": "高温 32°C", "fengli": "<![CDATA[<3级]]>", "low": "低温 26°C", "fengxiang": "无持续风向", "type": "小雨"}, {"date": "8日星期五", "high": "高温 32°C", "fengli": "<![CDATA[<3级]]>", "low": "低温 27°C", "fengxiang": "无持续风向", "type": "雷阵雨"}, {"date": "9日星期六", "high": "高温 32°C", "fengli": "<![CDATA[<3级]]>", "low": "低温 27°C", "fengxiang": "无持续风向", "type": "多云"}, {"date": "10日星期天", "high": "高温 32°C", "fengli": "<![CDATA[<3级]]>", "low": "低温 26°C", "fengxiang": "无持续风向", "type": "多云"}], "ganmao": "各项气象条件适宜, 发生感冒机率较低。但请避免长期处于空调房间中, 以防感冒。", "wendu": "27"}, {"status": 1000, "desc": "OK"}
```

7. Found key [http://wthrcdn.etouch.cn/weather\\_mini?citykey=101280601](http://wthrcdn.etouch.cn/weather_mini?citykey=101280601), value={"data":{"yesterday":{"date":"5日星期二", "high":"高温 32°C", "fx":"无持续风向", "low":"低温 27°C", "f1":"<![CDATA[<3级]]>", "type":"中雨"}, "city":"深圳", "aqi":"34", "forecast":[{"date":"6日星期三", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 26°C", "fengxiang":"无持续风向", "type":"中雨"}, {"date":"7日星期四", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 26°C", "fengxiang":"无持续风向", "type":"小雨"}, {"date":"8日星期五", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 27°C", "fengxiang":"无持续风向", "type":"雷阵雨"}, {"date":"9日星期六", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 27°C", "fengxiang":"无持续风向", "type":"多云"}, {"date":"10日星期天", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 26°C", "fengxiang":"无持续风向", "type":"多云"}], "ganmao": "各项气象条件适宜, 发生感冒机率较低。但请避免长期处于空调房间中, 以防感冒。", "wendu":"27"}, {"status": 1000, "desc": "OK"}
8. Found key [http://wthrcdn.etouch.cn/weather\\_mini?citykey=101280601](http://wthrcdn.etouch.cn/weather_mini?citykey=101280601), value={"data":{"yesterday":{"date":"5日星期二", "high":"高温 32°C", "fx":"无持续风向", "low":"低温 27°C", "f1":"<![CDATA[<3级]]>", "type":"中雨"}, "city":"深圳", "aqi":"34", "forecast":[{"date":"6日星期三", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 26°C", "fengxiang":"无持续风向", "type":"中雨"}, {"date":"7日星期四", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 26°C", "fengxiang":"无持续风向", "type":"小雨"}, {"date":"8日星期五", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 27°C", "fengxiang":"无持续风向", "type":"雷阵雨"}, {"date":"9日星期六", "high":"高温 32°C", "fengli":"<![CDATA[<3级]]>", "low":"低温 27°C", "fengxiang":"无持续风向", "type":"多云"}, {"date":"10日星期

```
天", "high": "高温 32°C", "fengli": "<![CDATA[<3级]]>", "low": "低温  
26°C", "fengxiang": "无持续风向", "type": "多云"}], "ganmao": "各项气象条件适  
宜，发生感冒机率较低。但请避免长期处于空调房间中，以防感  
冒。", "wendu": "27"}, {"status": 1000, "desc": "OK"}
```

9. **Not Found** key `http://wthrcdn.etouch.cn/weather_mini?citykey=101280601`

可以看到，第一次访问接口时，没有找到 Redis 里面的数据，所以，就初始化了数据。

后面几次访问，都是访问 Redis 里面的数据。最后一次，由于超时了，所以 Redis 里面又没有数据了，所以又会拿天气接口的数据。

## 源码

本章节的源码，在 `micro-weather-redis` 目录下。

## 微服务的注册与发现

- [微服务的注册与发现](#)

## 微服务的注册与发现

---

- [服务发现的意义](#)
- [如何集成Eureka](#)
- [实现服务的注册与发现](#)

## 服务发现的意义

- 服务发现的意义

## 服务发现的意义

---



## 如何集成Eureka

- 如何集成Eureka
  - 开发环境
  - 从 Spring Initializr 进行项目的初始化
  - 更改配置
  - 启用 Eureka Server
  - 修改项目配置
  - 启动

## 如何集成Eureka

---

本章节，我们将创建一个 `micro-weather-eureka-server` 作为注册服务器。

## 开发环境

---

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Server Finchley.M2

## 从 Spring Initializr 进行项目的初始化

---

访问<http://start.spring.io/> 进行项目的初始化。

Generate a Gradle Project with Java and Spring Boot 2.0.0 M3

**Project Metadata**  
Artifact coordinates

**Group**

**Artifact**

**Dependencies**  
Add Spring Boot Starters and dependencies to your application

**Search for dependencies**

**Selected Dependencies**  
Eureka Server

Generate Project alt + ⌘

Don't know what to look for? Want more options? [Switch to the full version.](#)

## 更改配置

根据下面两个博客的指引来配置，加速项目的构建。

- Gradle Wrapper 引用本地的发布包：  
<https://waylau.com/change-gradle-wrapper-distribution-url-to-local-file/>
- 使用Maven镜像：<https://waylau.com/use-maven-mirrors/>

## 启用 Eureka Server

为启用 Eureka Server，在 Application 上增加 `@EnableEurekaServer` 注解即可。

```
1. @SpringBootApplication
2. @EnableEurekaServer
3. public class Application {
```

```
4.  
5.     public static void main(String[] args) {  
6.         SpringApplication.run(Application.class, args);  
7.     }  
8. }
```

## 修改项目配置

修改 `application.properties`，增加如下配置。

```
1. server.port: 8761  
2.  
3. eureka.instance.hostname: localhost  
4. eureka.client.registerWithEureka: false  
5. eureka.client.fetchRegistry: false  
6. eureka.client.serviceUrl.defaultZone:  
    http://${eureka.instance.hostname}:${server.port}/eureka/
```

其中：

- `server.port`：指明了应用启动的端口号
- `eureka.instance.hostname`：应用的主机名称
- `eureka.client.registerWithEureka`：值为 `false` 意味着自身仅作为服务器，不作为客户端
- `eureka.client.fetchRegistry`：值为 `false` 意味着无需注册自身
- `eureka.client.serviceUrl.defaultZone`：指明了应用的 URL

## 启动

启动应用，访问<http://localhost:8761/>，可以看到 Eureka Server 自带的 UI 管理界面。

The screenshot displays the Spring Eureka web application interface. The browser address bar shows 'localhost:8761'. The page header includes the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'.

### System Status

Environment	test	Current time	2017-09-09T23:02:37 +0800
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			

### General Info

Name	Value
total-avail-memory	392mb
environment	test
num-of-cpus	8
current-memory-usage	254mb (64%)

# 实现服务的注册与发现

- 实现服务的注册与发现
  - 开发环境
  - 更改配置
  - 一个最简单的 Eureka Client
  - 运行
  - 源码

## 实现服务的注册与发现

本章节，我们将创建一个 `micro-weather-eureka-client` 作为客户端，并演示如何让将自身向注册服务器进行注册，让其可以其他服务都调用。

## 开发环境

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Client Finchley.M2

## 更改配置

增加如下配置：

```
1. dependencies {  
2.     //...  
3.  
4.     compile('org.springframework.cloud:spring-cloud-starter-  
       netflix-eureka-client')  
5.  
6.     //...  
7. }
```

## 一个最简单的 Eureka Client

```
1. @SpringBootApplication
2. @EnableDiscoveryClient
3. @RestController
4. public class Application {
5.
6.     @RequestMapping("/hello")
7.     public String home() {
8.         return "Hello world";
9.     }
10.
11.     public static void main(String[] args) {
12.         SpringApplication.run(Application.class, args);
13.     }
14. }
```

项目配置：

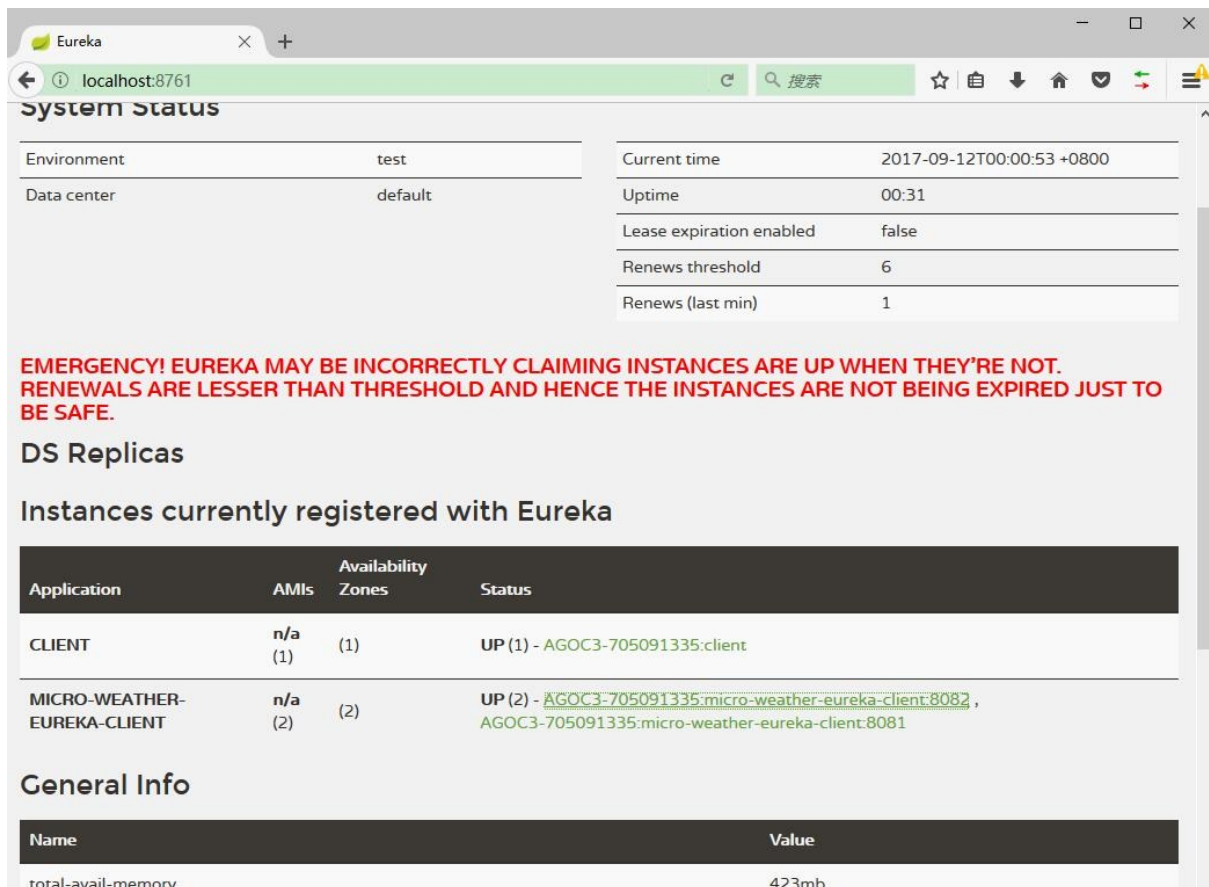
```
1. spring.application.name: micro-weather-eureka-client
2.
3. eureka.client.serviceUrl.defaultZone: http://localhost:8761/eureka/
```

## 运行

分别在 8081 和 8082 上启动了客户端示例。

```
1. java -jar micro-weather-eureka-client-1.0.0.jar --server.port=8081
2.
3. java -jar micro-weather-eureka-client-1.0.0.jar --server.port=8082
```

可以在 Eureka Server 上看到这两个实体的信息。



The screenshot shows the Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The page title is 'System Status'. Below the title, there are two tables. The first table shows 'Environment' as 'test' and 'Data center' as 'default'. The second table shows 'Current time' as '2017-09-12T00:00:53 +0800', 'Uptime' as '00:31', 'Lease expiration enabled' as 'false', 'Renews threshold' as '6', and 'Renews (last min)' as '1'. Below these tables, there is a red warning message: 'EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.' Below the warning, there is a section titled 'DS Replicas'. Underneath, there is a section titled 'Instances currently registered with Eureka'. This section contains a table with columns: 'Application', 'AMIs', 'Availability Zones', and 'Status'. The table has two rows: one for 'CLIENT' and one for 'MICRO-WEATHER-EUREKA-CLIENT'. Below the table, there is a section titled 'General Info' which contains a table with columns: 'Name' and 'Value'. The table has one row: 'total-avail-memory' with a value of '423mb'.

Application	AMIs	Availability Zones	Status
CLIENT	n/a (1)	(1)	UP (1) - AGOC3-705091335:client
MICRO-WEATHER-EUREKA-CLIENT	n/a (2)	(2)	UP (2) - AGOC3-705091335:micro-weather-eureka-client:8082 , AGOC3-705091335:micro-weather-eureka-client:8081

Name	Value
total-avail-memory	423mb

## 源码

本章节源码，见 `micro-weather-eureka-server` 和 `micro-weather-eureka-client` 。

## 微服务的消费

- [微服务的消费](#)

## 微服务的消费

---

- [微服务的消费模式](#)
- [常见微服务的消费者](#)
- [实现服务的消费者](#)
- [实现服务的负载均衡及高可用](#)



## 微服务的消费模式

- [微服务的消费模式](#)

## 微服务的消费模式

---

## 常见微服务的消费者

- 常见微服务的消费者

## 常见微服务的消费者

---

## 实现服务的消费者

- [实现服务的消费者](#)
  - [开发环境](#)
  - [更改配置](#)
  - [声明式 REST 客户端—Feign](#)
  - [一个最简单的 Feign](#)
  - [如何测试](#)
  - [源码](#)

## 实现服务的消费者

本章节，我们将创建一个 `micro-weather-feign` 作为服务器的消费者，并演示如何使用 Feign 来消费服务。

在 `micro-weather-eureka-client` 的基础上稍作修改即可。

## 开发环境

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Client Finchley.M2
- Spring Cloud Starter OpenFeign Finchley.M2

## 更改配置

增加如下配置：

```
1. dependencies {  
2.     //...  
3.
```

```

4.     compile('org.springframework.cloud:spring-cloud-starter-
        openfeign')
5.
6.     //...
7. }

```

## 声明式 REST 客户端—Feign

Feign 是一个声明式的 Web 服务客户端。这使得Web服务客户端的写入更加方便。它具有可插拔注释支持，包括Feign注释和JAX-RS注释。Feign还支持可插拔编码器和解码器。Spring Cloud增加了对Spring MVC注释的支持，并且使用了在Spring Web中默认使用的相同的HttpMessageConverter。 在使用Feign时，Spring Cloud集成了Ribbon和Eureka来提供负载平衡的http客户端。

## 一个最简单的 Feign

主应用：

```

1. @SpringBootApplication
2. @EnableDiscoveryClient
3. @EnableFeignClients
4. public class Application {
5.
6.     public static void main(String[] args) {
7.         SpringApplication.run(Application.class, args);
8.     }
9.
10. }

```

编写 Feign 请求接口：

```

1. package com.waylau.spring.cloud.service;
2.

```

```

3. import org.springframework.cloud.netflix.feign.FeignClient;
4. import org.springframework.web.bind.annotation.RequestMapping;
5. import org.springframework.web.bind.annotation.RequestMethod;
6.
7. /**
8.  * Hello Client
9.  *
10.  * @since 1.0.0 2017年9月17日
11.  * @author <a href="https://waylau.com">Way Lau</a>
12.  */
13. @FeignClient("micro-weather-eureka-client")
14. public interface HelloClient {
15.     @RequestMapping(method = RequestMethod.GET, value = "/hello")
16.     String getHello();
17. }

```

其中： `@FeignClient` 指定了要访问的服务的名称“micro-weather-eureka-client”。

项目配置：

```

1. spring.application.name: micro-weather-feign
2.
3. eureka.client.serviceUrl.defaultZone: http://localhost:8761/eureka/
4.
5. feign.client.config.feignName.connectTimeout: 5000
6. feign.client.config.feignName.readTimeout: 5000

```

## 如何测试

编写测试用例：

```

1. @RunWith(SpringRunner.class)
2. @SpringBootTest
3. public class HelloClientTest {
4.

```

```
5.     @Autowired
6.     private HelloClient helloClient;
7.
8.     @Test
9.     public void testHello() {
10.         String hello = helloClient.getHello();
11.         System.out.println(hello);
12.     }
13.
14. }
```

启动在之前章节中搭建的 `micro-weather-eureka-server` 和 `micro-weather-eureka-client` 两个项目。这样，`micro-weather-eureka-client` 服务，就能被 `micro-weather-feign` 发现，并进行访问。

启动测试用例，如果一切正常，可以在控制台看到“Hello world”字样。这个就是请求 `micro-weather-eureka-client` 服务时响应的内容。

如果同时也启动了 `micro-weather-feign` 项目，则能在启动在之前章节中搭建的 `micro-weather-eureka-server` 管理界面，看到这个服务的信息。

Eureka

agoc3-705091335:8080/


agoc3-705091335:8080/

新标签页

localhost:8761

搜索

☆ 自 ↓ 家 心 旗

spring Eureka

HOME   LAST 1000 SINCE STARTUP

### System Status

Environment	test	Current time	2017-09-17T20:53:44 +0800
Data center	default	Uptime	01:26
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	4

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

### DS Replicas

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MICRO-WEATHER-EUREKA-CLIENT	n/a (2)	(2)	UP (2) - AGOC3-705091335:micro-weather-eureka-client , AGOC3-705091335:micro-weather-eureka-client:8082
MICRO-WEATHER-FEIGN	n/a (1)	(1)	UP (1) - AGOC3-705091335:micro-weather-feign:8090

# 源码

本章节源码，见 `micro-weather-feign` 。

## 实现服务的负载均衡及高可用

- 实现服务的负载均衡及高可用
  - 客户端负载均衡器—Ribbon
  - 实现高可用

## 实现服务的负载均衡及高可用

如果你自己观察 Feign 依赖，可以看到，Feign 是依赖了 Ribbon 的。

```
1. <dependency>
2.     <groupId>org.springframework.cloud</groupId>
3.     <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
4. </dependency>
```

有兴趣的朋友，可以自行查看依赖信

息：<https://github.com/spring-cloud/spring-cloud-netflix/blob/master/spring-cloud-starter-netflix/spring-cloud-starter-openfeign/pom.xml>

## 客户端负载均衡器—Ribbon

Ribbon 是一个客户端负载平衡器，它可以很好地控制HTTP和TCP客户端的行为。 Feign 已经使用 Ribbon，所以如果你使用 `@FeignClient`，就已经启用了客户端负载均衡功能。

Ribbon 的一个中心概念就是命名客户端 (named client)。 每个负载均衡器都是组合的组件的一部分，它们一起工作以根据需要联系远程服务器，并且集合具有您将其作为应用程序开发人员（例如使用 `@FeignClient` 注解）的名称。 Spring Cloud使用



RibbonClientConfiguration为每个命名的客户端根据需要创建一个新的集合作为ApplicationContext。 这包含（其中包括）一个ILoadBalancer，一个RestClient和一个ServerListFilter。

## 实现高可用

---

将我们需要访问的服务（比如，“micro-weather-eureka-client”），启动为多个示例。当客户端需要访问“micro-weather-eureka-client”时，会自行去选择其中任意一个服务实例来访问，这样，即便其中的某个服务实例不可用，也不会影响整个服务功能。

这样就实现了服务的高可用。

# API 网关

- [API 网关](#)

## API 网关

---

- [API 网关的意义](#)
- [常见 API 网关的实现方式](#)
- [如何集成Zuul](#)
- [实现 API 网关](#)

# API 网关的意义

- [API 网关的意义](#)

## API 网关的意义

---

## 常见 API 网关的实现方式

- 常见 API 网关的实现方式

## 常见 API 网关的实现方式

---

## 如何集成Zuul

- [如何集成Zuul](#)
  - [开发环境](#)
  - [更改配置](#)

## 如何集成Zuul

---

路由是微服务架构中必须的一部分，比如，“/” 可能映射到你的WEB程序上，“/api/users”可能映射到你的用户服务上，“/api/shop”可能映射到你的商品服务商。通过路由，让不同的服务，都集中到统一的入口上来，这就是 API 网关的作用。

Zuul是Netflix出品的一个基于JVM路由和服务端的负载均衡器。

Zuul功能包括：

- 认证
- 压力测试
- 金丝雀测试
- 动态路由
- 负载削减
- 安全
- 静态响应处理
- 主动/主动交换管理

Zuul的规则引擎允许通过任何JVM语言来编写规则和过滤器，支持基于Java和Groovy的构建。

配置属性 `zuul.max.host.connections` 已经被两个新的配置属性替代，`zuul.host.maxTotalConnections`（总连接数）和

`zuul.host.maxPerRouteConnections`, ( 每个路由连接数 ) 默认值分别是200和20。

在 `micro-weather-eureka-client` 的基础上稍作修改即可。新的项目名称为 `micro-weather-zuul` 。

## 开发环境

---

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Client Finchley.M2
- Spring Cloud Netflix Zuul Finchley.M2

## 更改配置

---

增加如下配置：

```
1. dependencies {  
2.     //...  
3.  
4.     compile('org.springframework.cloud:spring-cloud-starter-  
       netflix-zuul')  
5.  
6.     //...  
7. }
```

# 实现 API 网关

- 实现 API 网关
  - 一个最简单的 Zuul 应用
  - 如何测试
  - 源码

## 实现 API 网关

### 一个最简单的 Zuul 应用

主应用：

```
1. @SpringBootApplication
2. @EnableDiscoveryClient
3. @EnableZuulProxy
4. public class Application {
5.
6.     public static void main(String[] args) {
7.         SpringApplication.run(Application.class, args);
8.     }
9.
10. }
```

其中： `@EnableZuulProxy` 启用了 Zuul 作为反向代理服务器。

项目配置：

```
1. spring.application.name: micro-weather-zuul
2.
3. eureka.client.serviceUrl.defaultZone: http://localhost:8761/eureka/
4.
5. zuul.routes.users.path: /hi/**
6. zuul.routes.users.serviceId: micro-weather-eureka-client
```

---

其中：

- `zuul.routes.users.path` ： 为要拦截请求的路径；
- `zuul.routes.users.serviceId`： 为要拦截请求的路径所要映射的服务。本例，我们将所有 `/hi` 下的请求，都转发到 `micro-weather-eureka-client` 服务中去。

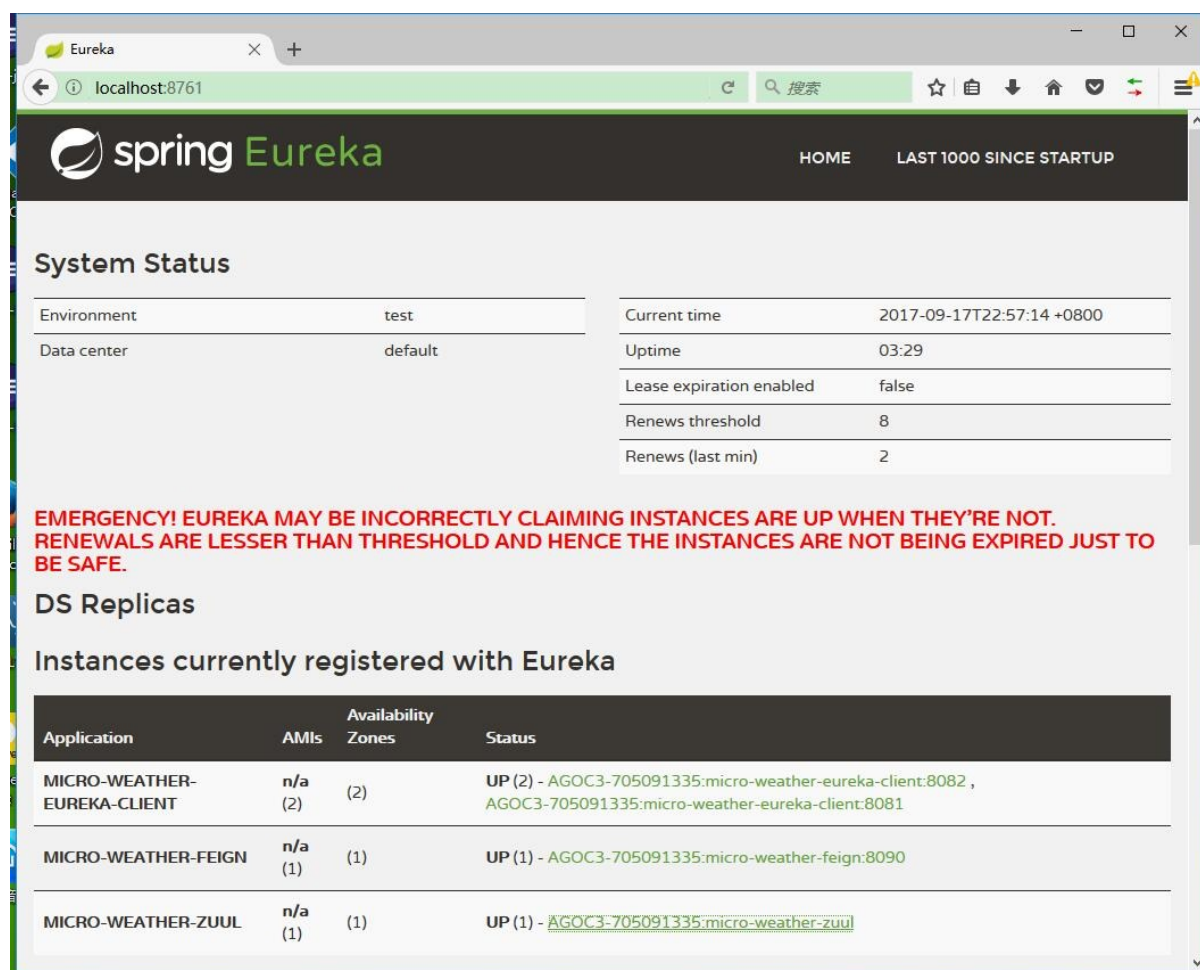
## 如何测试

---

启动在之前章节中搭建的 `micro-weather-eureka-server` 和 `micro-weather-eureka-client` 两个项目，以及本例的 `micro-weather-zuul` 。

如果一切正常，在之前章节中搭建的 `micro-weather-eureka-server` 管理界面，能看到上述服务的信息。





在浏览器访问 `micro-weather-zuul` 服务（本例，地址为），当我们试图访问接口时，只需要访问如果一切正常，可以在控制台看到“Hello world”字样，这个就是转发请求到 `micro-weather-eureka-client` 服务时响应的内容。

## 源码

本章节源码，见 `micro-weather-zuul` 。

## 微服务的部署与发布

- [微服务的部署与发布](#)

## 微服务的部署与发布

---

- [部署微服务将面临的挑战](#)
- [持续交付与持续部署微服务](#)
- [基于容器的部署与发布微服务](#)
- [Docker 简介](#)
- [基于 Docker 打包](#)
- [基于 Docker 发布](#)
- [基于 Docker 部署](#)

## 部署微服务将面临的挑战

- 部署微服务将面临的挑战

## 部署微服务将面临的挑战

---

# 持续交付与持续部署微服务

- 持续交付与持续部署微服务

## 持续交付与持续部署微服务

---

# 基于容器的部署与发布微服务

- 基于容器的部署与发布微服务

## 基于容器的部署与发布微服务

---

# Docker 简介

- [Docker 简介](#)

## Docker 简介

---

## 基于 Docker 打包

- 基于 Docker 打包

## 基于 Docker 打包

---

## 基于 Docker 发布

- 基于 Docker 发布

## 基于 Docker 发布

---



## 基于 Docker 部署

- 基于 Docker 部署

## 基于 Docker 部署

---

# 微服务的集中化配置

- [微服务的集中化配置](#)

## 微服务的集中化配置

---

- [为什么需要集中化配置](#)
- [集中化配置的实现原理](#)
- [如何集成 Spring Cloud Config](#)
- [实现微服务的集中化配置](#)

# 为什么需要集中化配置

- [为什么需要集中化配置](#)

# 为什么需要集中化配置

---

## 集中化配置的实现原理

- [集中化配置的实现原理](#)

## 集中化配置的实现原理

---

## 如何集成 Spring Cloud Config

- [如何集成 Config](#)
  - [开发环境](#)
  - [更改配置](#)
  - [一个最简单的 Config Server](#)
  - [测试](#)
  - [源码](#)

## 如何集成 Config

本章节，我们将创建一个 `micro-weather-config-server` 作为配置服务器的服务端。

## 开发环境

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Client Finchley.M2
- Spring Cloud Config Server Finchley.M2

## 更改配置

增加如下配置：

```
1. dependencies {
2.     //...
3.
4.     compile('org.springframework.cloud:spring-cloud-starter-
       netflix-eureka-client')
5.     compile('org.springframework.cloud:spring-cloud-config-server')
```

```

6.
7.     //...
8. }

```

## 项目配置：

```

1. spring.application.name: micro-weather-config-server
2. server.port=8888
3.
4. eureka.client.serviceUrl.defaultZone: http://localhost:8761/eureka/
5.
6. spring.cloud.config.server.git.uri=https://github.com/waylau/spring-
  cloud-tutorial
7. spring.cloud.config.server.git.searchPaths=config

```

## 其中：

- `spring.cloud.config.server.git.uri`：配置Git仓库地址
- `spring.cloud.config.server.git.searchPaths`：配置查找配置的路径

## 一个最简单的 Config Server

### 主应用：

```

1. @SpringBootApplication
2. @EnableDiscoveryClient
3. @EnableConfigServer
4. public class Application {
5.
6.     public static void main(String[] args) {
7.         SpringApplication.run(Application.class, args);
8.     }
9.
10. }

```

在程序的入口Application类加上 `@EnableConfigServer` 注解开启配置服务器的功能。

## 测试

在<https://github.com/waylau/spring-cloud-tutorial/tree/master/config> 我们放置了一个配置文件 `micro-weather-config-client-dev.properties`，里面简单的放置了测试内容：

```
1. auther=waylau.com
```

启动应用，访问<http://localhost:8888/auther/dev>，应能看到如下输出内容，说明服务启动正常。

```
1. {"name":"auther","profiles":
  ["dev"],"label":null,"version":"00836f0fb49488bca170c8227e3ef13a5aff2
  []}
```

其中，在配置中心的文件命名规则如下：

1. `/application/profile[/label]`
2. `/application-profile.yml`
3. `/label/application-profile.yml`
4. `/application-profile.properties`
5. `/label/application-profile.properties`

## 源码

本章节源码，见 `micro-weather-config-server`。





## 实现微服务的集中化配置

- [实现微服务的集中化配置](#)
  - [开发环境](#)
  - [更改配置](#)
  - [一个最简单的 Config Client](#)
  - [如何测试](#)
  - [源码](#)

## 实现微服务的集中化配置

创建一个 `micro-weather-config-client` 作为配置服务器的客户端。

### 开发环境

- Gradle 4.0
- Spring Boot 2.0.0.M3
- Spring Cloud Netflix Eureka Client Finchley.M2
- Spring Cloud Config Client Finchley.M2

### 更改配置

增加如下配置：

```
1. dependencies {  
2.     //...  
3.  
4.     compile('org.springframework.cloud:spring-cloud-starter-  
        netflix-eureka-client')  
5.     compile('org.springframework.cloud:spring-cloud-starter-  
        config')  
6.
```

```
7.      //...
8.  }
```

## 项目配置：

```
1.  spring.application.name: micro-weather-config-client
2.  server.port=8089
3.
4.  eureka.client.serviceUrl.defaultZone: http://localhost:8761/eureka/
5.
6.  spring.cloud.config.profile=dev
7.  spring.cloud.config.uri= http://localhost:8888/
```

## 其中：

- `spring.cloud.config.uri` ： 指向了配置服务器 `micro-weather-config-server` 的位置。

## 一个最简单的 Config Client

### 主应用：

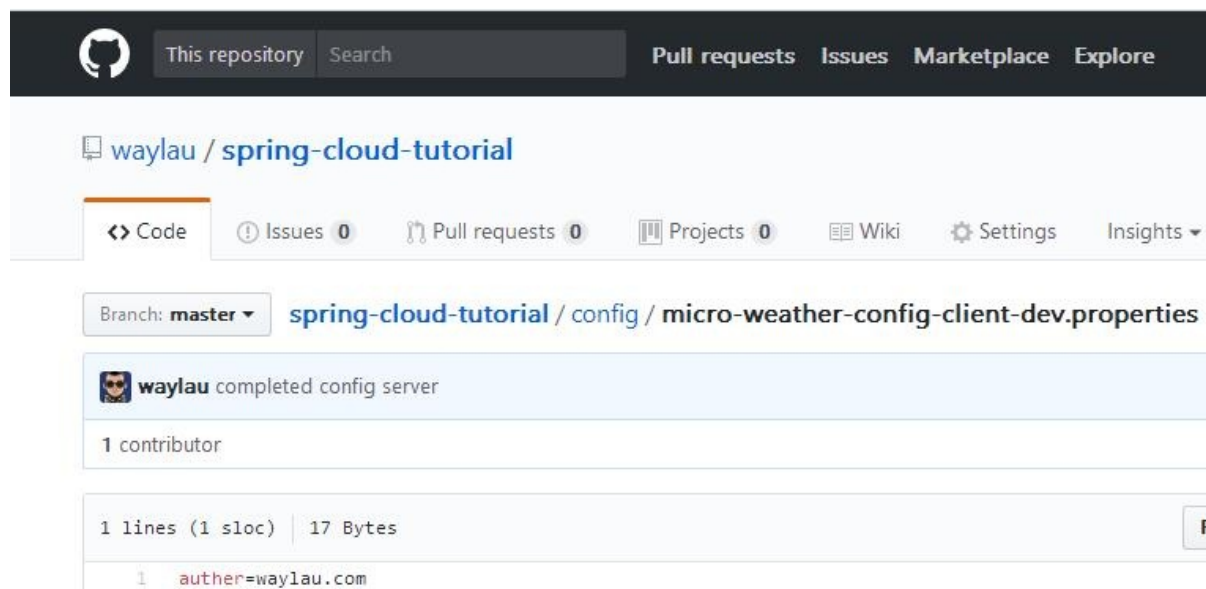
```
1.  @SpringBootApplication
2.  @EnableDiscoveryClient
3.  public class Application {
4.
5.      public static void main(String[] args) {
6.          SpringApplication.run(Application.class, args);
7.      }
8.
9.  }
```

## 如何测试

在<https://github.com/waylau/spring-cloud->

[tutorial/tree/master/config](#) 我们放置了一个配置文件 `micro-weather-config-client-dev.properties`，里面简单的放置了测试内容：

```
1.  auther=waylau.com
```



编写测试用例：

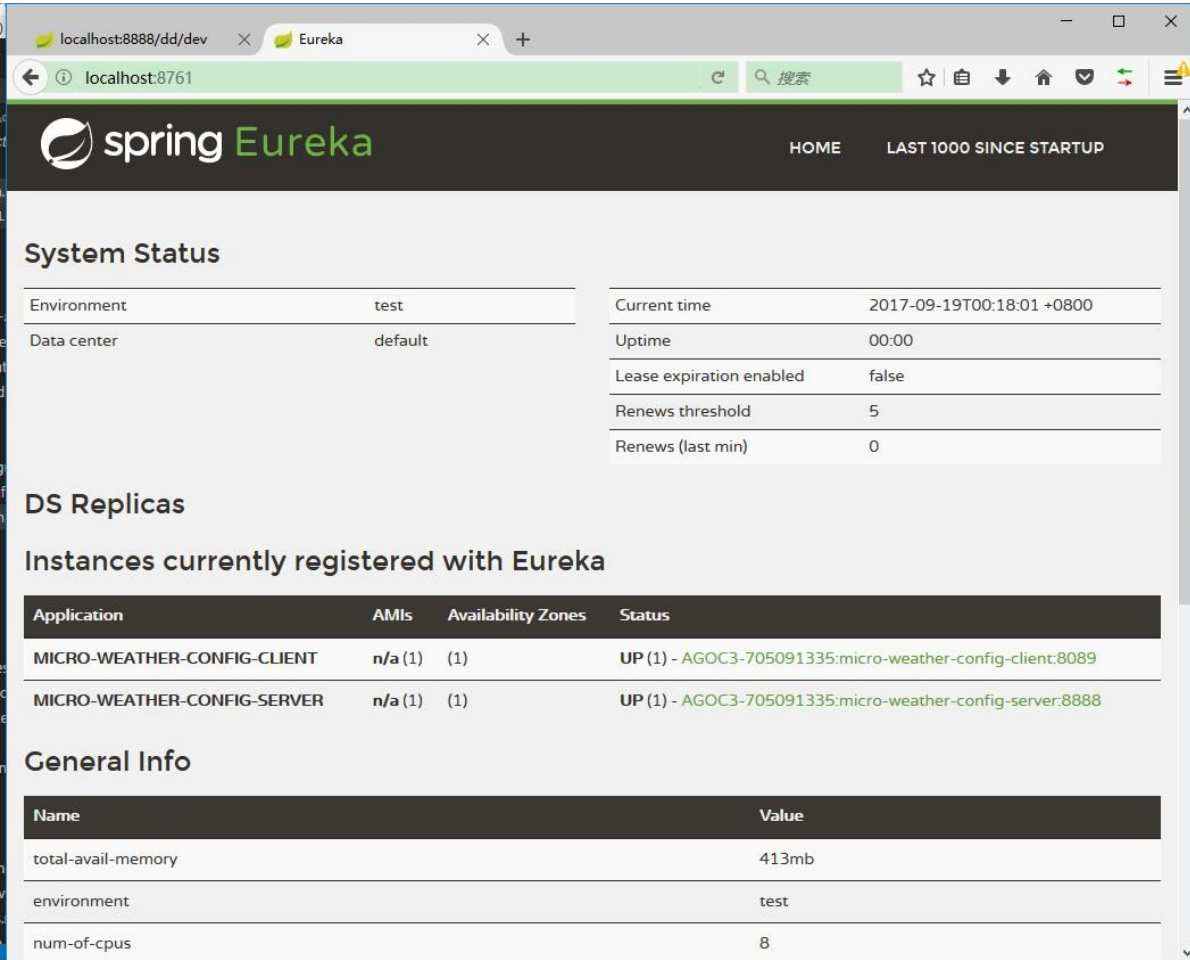
```
1.  @RunWith(SpringRunner.class)
2.  @SpringBootTest
3.  public class ApplicationTests {
4.
5.      @Value("${auther}")
6.      private String auther;
7.
8.      @Test
9.      public void contextLoads() {
10.          System.out.println(auther);
11.      }
12.
13. }
```

启动在之前章节中搭建的 `micro-weather-eureka-server` 和 `micro-`

`weather-config-server` 两个项目。

启动测试用例，如果一切正常，可以在控制台看到“waylau.com”字样，说明，我们拿到了 `author` 在配置服务器中的内容。

如果同时也启动了 `micro-weather-config-client` 项目，则能在启动在之前章节中搭建的 `micro-weather-eureka-server` 管理界面，看到这个服务的信息。



The screenshot shows the Spring Eureka management interface in a web browser. The browser tabs include 'localhost:8888/dd/dev' and 'Eureka'. The address bar shows 'localhost:8761'. The page title is 'spring Eureka'. The navigation bar has 'HOME' and 'LAST 1000 SINCE STARTUP' links.

### System Status

Environment	test	Current time	2017-09-19T00:18:01 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

### DS Replicas

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MICRO-WEATHER-CONFIG-CLIENT	n/a (1)	(1)	UP (1) - AGOC3-705091335:micro-weather-config-client:8089
MICRO-WEATHER-CONFIG-SERVER	n/a (1)	(1)	UP (1) - AGOC3-705091335:micro-weather-config-server:8888

### General Info

Name	Value
total-avail-memory	413mb
environment	test
num-of-cpus	8

## 源码

本章节源码，见 `micro-weather-config-client` 。



# 微服务的高级主题——自动扩展

- [微服务的高级主题——自动扩展](#)

## 微服务的高级主题——自动扩展

---

- [什么是自动扩展](#)
- [自动扩展的意义](#)
- [自动扩展的常见模式](#)
- [实现微服务的自动扩展](#)

# 什么是自动扩展

- 什么是自动扩展

# 什么是自动扩展

---

# 自动扩展的意义

- 自动扩展的意义

## 自动扩展的意义

---



## 自动扩展的常见模式

- [自动扩展的常见模式](#)

## 自动扩展的常见模式

---

# 实现微服务的自动扩展

- 实现微服务的自动扩展

## 实现微服务的自动扩展

---

## 微服务的高级主题——熔断机制

- [微服务的高级主题——熔断机制](#)

## 微服务的高级主题——熔断机制

---

# 什么是服务的熔断机制

- [什么是服务的熔断机制](#)

# 什么是服务的熔断机制

---

# 熔断的意义

- 熔断的意义

## 熔断的意义

---

## 熔断与降级的区别

- [熔断与降级的区别](#)

## 熔断与降级的区别

---

## 如何集成 Hystrix

- [如何集成 Hystrix](#)

## 如何集成 Hystrix

---

# 实现微服务的熔断机制

- [实现微服务的熔断机制](#)

## 实现微服务的熔断机制

---



## 参考资料

- [参考资料](#)

## 参考资料

---

- <http://projects.spring.io/spring-cloud/>