

# 服务配置



扫码试看/订阅  
《玩转 Spring 全家桶》

# 基于 Git 的配置中心

# Spring Cloud Config Server

## 目的

- 提供针对外置配置的 HTTP API 默认localhost:8888

## 依赖

- spring-cloud-config-server
- @EnableConfigServer 表示这个程序就是一个config-server
- 支持 Git / SVN / Vault / JDBC ...

# 使用 Git 作为后端存储

## 配置

- MultipleJGitEnvironmentProperties  
application.properties配置
- spring.cloud.config.server.git.uri 告诉config server git的URI是什么

## 配置文件的要素

bootstrap.properties里面配置

spring.application.name=configserver 客户端起个名字

- {application}, 即客户端的 spring.application.name
- {profile}, 即客户端的 spring.profiles.active 不同Profile逗号分割
- {label}, 配置文件的特定标签, 默认 master 比如配置了waiter-service-dev.yml  
访问localhost:8888/waiter-service/dev

访问localhost:8888/waiter-service/dev/master

# 使用 Git 作为后端存储

比如配置了一份waiter-service.yml在Git里面，启动后访问localhost:8888/waiter-service.yml即可。

## 访问配置内容

如果再添加了一份waiter-service-dev.yml,访问localhost:8888/waiter-service-dev.yml会把两份配置合并，相同项以dev配置数据为准

- HTTP 请求
  - GET `/ {application} / {profile} [ / {label} ]` 获得不同branch下面
  - GET `/ {application} - {profile} .yml`
  - GET `/ {label} / {application} - {profile} .yml` 这个配置文件放到Git仓库里面
  - GET `/ {application} - {profile} .properties`
  - GET `/ {label} / {application} - {profile} .properties`

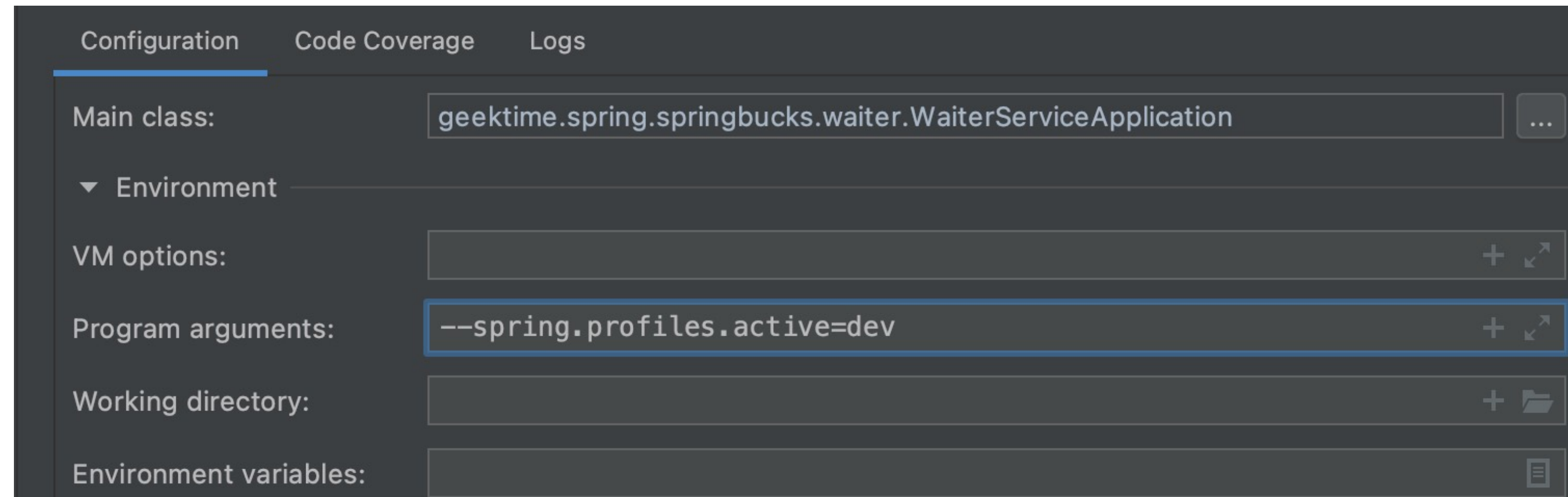
**“Talk is cheap, show me the code.”**

*Chapter 14 / config-server*

把config-server作为服务注册到consul里面：

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

切换active环境：





client获取server的配置

# Spring Cloud Config Client

## 依赖

- spring-cloud-starter-config

## 发现配置中心

两种方式发现配置中心：

- 1.配置死地址
- 2.服务注册发现

- `bootstrap.properties` | yml
- `spring.cloud.config.fail-fast=true` 如果访问不上，做一个快速失败
- 通过配置 指定配置中心的URL
  - `spring.cloud.config.uri=http://localhost:8888`

# Spring Cloud Config Client

## 发现配置中心

- `bootstrap.properties` | yml
- 通过服务发现
  - `spring.cloud.config.discovery.enabled=true`
  - `spring.cloud.config.discovery.service-id=configserver` 就是server中起的名字

## 配置刷新

- `@RefreshScope` 希望配置项可以刷新的话，把配置项写到bean里面，然后加上该注解
- Endpoint - `/actuator/refresh` 访问这个endpoint的时候，spring-cloud会刷新配置  
`curl -X POST http://localhost:8080/actuator/refresh`

**“Talk is cheap, show me the code.”**

*Chapter 14 / git-config-waiter-service*

# 基于 Zookeeper 的配置中心

# Spring Cloud Zookeeper Config

## 依赖

- spring-cloud-starter-zookeeper-config
- 注意 Zookeeper 版本

## 启用

- `bootstrap.properties` | yml 配置zk注册信息
- `spring.cloud.zookeeper.connect-string=localhost:2181` zk地址
- `spring.cloud.zookeeper.config.enabled=true` 默认就是true

# Zookeeper 中的数据怎么存

配置项 根节点创建/config节点

create /config

- /config/**应用名,profile**/key=value create /config/waiter-service/order.discount 60
- /config/application,profile/key=value 默认名

## 如何定制

- spring.cloud.zookeeper.config.root=config 配置默认上下文
- spring.cloud.zookeeper.config.default-context=application
- spring.cloud.zookeeper.config.profile-separator=', ' 应用名跟profile分隔符，默认是逗号

zk本身有节点更新功能，如果变化了可以推送到zk客户端，这样应用不用强制refresh也可以变化

**“Talk is cheap, show me the code.”**

*Chapter 14 / zk-config-waiter-service*

# 深入理解 Spring Cloud 的配置抽象



# Spring Cloud Config

## 目标

- 在分布式系统中，提供外置配置支持

## 实现

- 类似于 Spring 应用中的 Environment 与 PropertySource
- 在上下文中增加 Spring Cloud Config 的 PropertySource

增加了propertySource抽象

# Spring Cloud Config 的 PropertySource

## PropertySource


- Spring Cloud Config Client - CompositePropertySource
- Zookeeper - ZookeeperPropertySource
- Consul - ConsulPropertySource / ConsulFilesPropertySource


## PropertySourceLocator


配置中心的PropertySource优先级大于Spring本身的PropertySource


- 通过 PropertySourceLocator 提供 PropertySource  
同构Locator找PropertySource


接口定义


▼  Maven: org.springframework.cloud:spring-cloud-context:2.1.1.RELEASE


▼  spring-cloud-context-2.1.1.RELEASE.jar library root


>  META-INF


▼  org.springframework.cloud


>  autoconfigure


▼  bootstrap


▼  config


 PropertySourceBootstrapConfiguration


 PropertySourceBootstrapProperties


 PropertySourceLocator


>  encrypt

>  BootstrapApplicationListener


 BootstrapConfiguration


>  BootstrapImportSelector


 BootstrapImportSelectorConfiguration


 LoggingSystemShutdownListener


服务端接口实现


 Maven: org.springframework.cloud:spring-cloud-config-client:2.1.1.RELEASE


 spring-cloud-config-client-2.1.1.RELEASE.jar library root


>  META-INF


▼  org.springframework.cloud.config


▼  client


>  ConfigClientAutoConfiguration


 ConfigClientHealthProperties


>  ConfigClientProperties


 ConfigClientStateHolder


 ConfigClientWatch


 ConfigServerHealthIndicator

 ConfigServerInstanceProvider

>  ConfigServiceBootstrapConfiguration

>  ConfigServicePropertySourceLocator

 DiscoveryClientConfigServiceBootstrapConfiguration

 RetryProperties

# Spring Cloud Config Server

**EnvironmentRepository** 支持多个backened

- Git / SVN / Vault / JDBC ... [Spring cloud Config server 文档观看](#)

## 功能特性

- SSH、代理访问、配置加密 ...

## 配置刷新

- `/actuator/refresh` 如果集群很大，每台机器刷新不现实
- Spring Cloud Bus - `RefreshRemoteApplicationEvent`  
[这个事件可以对整个集群刷新](#)



# Spring Cloud Config Zookeeper

## **ZookeeperConfigBootstrapConfiguration**

- 注册 ZookeeperPropertySourceLocator
- 提供 ZookeeperPropertySource

## **ZookeeperConfigAutoConfiguration**

- 注册 ConfigWatcher 监听配置节点，配置项发生变化通过这个刷新

▼ Maven: org.springframework.cloud:spring-cloud-starter-zookeeper-config:2.1.1.RELEASE

- ▼ spring-cloud-zookeeper-config-2.1.1.RELEASE.jar library root
  - > META-INF
  - ▼ org.springframework.cloud.zookeeper.config
    - AbstractZookeeperPropertySource
    - ConfigWatcher**
    - > ZookeeperConfigAutoConfiguration

```
@PostConstruct
public void start() {
    if (this.running.compareAndSet( expectedValue: false, newValue: true)) {
        this.caches = new HashMap();
        Iterator var1 = this.contexts.iterator();

        while(var1.hasNext()) {
            String context = (String)var1.next();
            if (!context.startsWith("/")) {
                context = "/" + context;
            }

            try {
                TreeCache cache = TreeCache.newBuilder(this.source, context).build();
                cache.getListenable().addListener( t: this);
                cache.start();
                this.caches.put(context, cache);
            } catch (NoNodeException var4) {
            } catch (Exception var5) {
                log.error( message: "Error initializing listener for context " + context, var5);
            }
        }
    }
}
```

收到信息后

```
public void childEvent(CuratorFramework client, TreeCacheEvent event) throws Exception {
    Type eventType = event.getType();
    if (eventType == Type.NODE_ADDED || eventType == Type.NODE_REMOVED || eventType == Type.NODE_UPDATED) {
        this.publisher.publishEvent(new RefreshEvent( source: this, event, this.getEventDesc(event)));
    }
}
```

```
public void handle(RefreshEvent event) {
    if (this.ready.get()) { // don't handle events before app is ready
        log.debug("Event received " + event.getEventDesc());
        Set<String> keys = this.refresh.refresh();
        log.info("Refresh keys changed: " + keys);
    }
}
```



# 配置的组合顺序

以 yml 为例

- 应用名-profile.yml
- 应用名.yml
- application-profile.yml
- application.yml



# 基于 Consul 的配置中心

# Spring Cloud Consul Config

## 依赖

- `spring-cloud-starter-consul-config`

## 启用

- `bootstrap.properties | yml`
  - `spring.cloud.consul.host=localhost`
  - `spring.cloud.consul.port=8500`
  - `spring.cloud.consul.config.enabled=true`

# Consul 中的数据怎么存

## 配置项

- `spring.cloud.consul.config.format=`  
`KEY_VALUE | YAML | PROPERTIES | FILES`
- `/config/应用名,profile/data`
- `/config/application,profile/data`

# Consul 中的数据怎么存

## 如何定制

- `spring.cloud.consul.config.data-key=data`
- `spring.cloud.consul.config.root=config`
- `spring.cloud.consul.config.default-context=application`
- `spring.cloud.consul.config.profile-separator=','`

# 配置项变更

## 自动刷新配置

- `spring.cloud.consul.config.watch.enabled=true`
- `spring.cloud.consul.config.watch.delay=1000`

## 实现原理

- 单线程 `ThreadPoolTaskScheduler`
- `ConsulConfigAutoConfiguration.CONFIG_WATCH_TASK_SCHEDULER_NAME`

**“Talk is cheap, show me the code.”**

*Chapter 14 / consul-config-waiter-service*

# 基于 Nacos 的配置中心

# Spring Cloud Alibaba Nacos Config

## 依赖

- `spring-cloud-starter-alibaba-nacos-config`
- `spring-cloud-alibaba-dependencies:0.9.0`
  - 注意 Spring Cloud 与 Spring Boot 的对应版本

## 启用

- `bootstrap.properties | yml`
  - `spring.cloud.nacos.config.server-addr=127.0.0.1:8848`
  - `spring.cloud.nacos.config.enabled=true`



# Nacos 中的数据怎么存

## 配置项

- dataId
  - `${prefix}-${spring.profile.active}.${file-extension}`
  - `spring.cloud.nacos.config.prefix`
  - `spring.cloud.nacos.config.file-extension`
- `spring.cloud.nacos.config.group`

**“Talk is cheap, show me the code.”**

*Chapter 14 / nacos-config-waiter-service*

# SpringBucks 实战项目进度小结

# 本章小结

## 几种不同的配置中心

- Spring Cloud Config Server
  - Git / SVN / RDBMS / Vault
- Zookeeper
- Consul
- Nacos

# SpringBucks 进度小结

## **waiter-service**

- 增加了订单金额与折扣
- 增加了 Waiter 名称
- 使用了不同的配置中心
  - Spring Cloud Config Client
  - 使用 Zookeeper
  - 使用 Consul
  - 使用 Nacos

# 携程 Apollo

## 官方地址

- <https://github.com/ctripcorp/apollo>

## 特性

- 统一管理不同环境、不同集群的配置
- 配置修改实时生效（热发布）
- 版本发布管理
- 灰度发布
- 权限管理、发布审核、操作审计
- 客户端配置信息监控
- 提供开放平台API



扫码试看/订阅  
《玩转 Spring 全家桶》