# Spring Cloud 预备知识

## 设计模式

### 发布/订阅模式（观察者模式）

* **Java内置的观察则模式**

在java.util包中包含有基本的Observer接口和Observable抽象类. 许多功能比如说注册,删除,通知观察者的那些功能已经内置好了.

（1）如何使对象变为观察者?

实现观察者接口(java.util.Observer),然后调用Observable对象的addObserver()方法.不想再当观察者时,调用deleteObserver()就可以了.

（2）被观察者(主题)如何发出通知?

第一步:先调用setChanged()方法,标识状态已经改变的事实.

第二步:调用notifyObservers()方法或者notifyObservers(Object arg),这就牵扯到推(push)和拉(pull)的方式传送数据.如果想用push的方式"推"数据给观察者,可以把数据当做数据对象传送给notifyObservers(Object arg)方法,其中的arg可以为任意对象,意思是你可以将任意对象传送给每一个观察者.如果调用不带参数的notifyObserver()方法,则意味着你要使用pull的方式去主题对象中"拉"来所需要的数据.

（3）观察者如何接收通知?

观察者只需要实现一个update(Observable o,Object arg)方法,第一个参数o,是指定通知是由哪个主题下达的,第二个参数arg就是上面notifyObserver(Object arg)里传入的数据,如果不传该值,arg为null.

### 事件/监听模式

java.util.EventObject

java.util.EventListener

## Spring事件监听

org.springframework.context.ApplicationEvent:应用事件

org.springframework.context.ApplicationListener:应用监听器

# Spring Cloud Config

## Spring Boot事件/监听

## Spring Cloud事件/监听

### 每个Bean都有一个代理

### 所有Bean共享一个代理基类

## ApplicationContext层次性

### READ\_UNCOMMITTED

## EnvironmentEndpoint

### PROPAGATION\_REQUIRED

## 客户端

### 配置第三方库

配置三方库

commons-configuration

Configuration : 提供大多数常见类型的 Value 转换（而Properties **extends** Hashtable<Object,Object> 类型都是Object）

* PropertiesConfiguration: 将 Properties 作为 Configuration 配置
* MapConfiguration
* EnvironmentConfiguration ： OS 环境变量
* SystemConfiguration : Java 系统属性
* CompositeConfiguration

核心概念：

**配置源(存储媒介)**、以及它们**优先次序（配置优先级覆盖）**、**配置转换能力（编程便利）**

HTTP 资源算不算一个配置？

配置源：文件、HTTP 资源、数据源、 Git ->

URL -> file:/// , http://, jdbc:// , git://

### Spring Environment

Environment -> ConfigurableEnvironment: 父子层次

ConfigurableEnvironment -> MutablePropertySources: 获取可变多个配置源

MutablePropertySources -> List PropertySource : 包含多个 PropertySource

PropertySource : 配置源

MapPropertySource

PropertiesPropertySource

CompositePropertySource : 组合

SystemEnvironmentPropertySource 环境变量

Spring Cloud 客户端配置定位扩展 : PropertySourceLocator

## 服务端

### 基于 Git 的实现

版本化配置

/应用名/profile/${label}

/应用名/profile/ = /应用名/profile/master

/应用名/ = /应用名.properties

${label} : 分支

Spring Cloud Config 实现一套完整的配置管理 API 设计

Git 实现缺陷：

复杂的版本更新机制（ Git 仓库）

版本

分支

提交

配置

憋足的内容更新（实时性不高）

客户端第一次启动拉取

需要整合 BUS 做更新通知

### 设计原理

**分析 @EnableConfigServer**

实际配置类 ConfigServerConfiguration

当应用配置类标注了@EnableConfigSever

导入 ConfigServerConfiguration

注册 Marker Bean

作为 ConfigServerAutoConfiguration 条件之一

**案例分析 JDBC 实现**

JdbcTemplate Bean 来源

JdbcTemplateAutoConfiguration

SQL 来源

JdbcEnvironmentProperties

spring.cloud.config.server.jdbc.sql

不配置，默认：DEFAULT\_SQL

sql

SELECT KEY, VALUE from PROPERTIES where APPLICATION=? and PROFILE=? and LABEL=?

**本质说明：**

JDBC 连接技术

DB 存储介质

EnvironmentRepository 核心接口

思考是否可以自定义 EnvironmentRepository 实现？

前提：如何激活自定义的 EnvironmentRepository 实现

找到了为什么默认是 Git 作为配置仓库的原因：



当 Spring 应用上下文没有出现 EnvironmentRepository Bean 的时候，那么，默认激活 DefaultRepositoryConfiguration (Git 实现)，否则采用自定义实现。

### 自定义实现

自定义 EnvironmentRepository Bean



以上实现将失效 DefaultRepositoryConfiguration 装配。

**HTTP 请求模式**

/${application}/$${profile}/$${label}

@Controller 或者 @RestController

@RequestMapping("/{application}/{profile}/{label}")

/config/test/master

config : application

test : profile

master : label

# 服务注册与发现

# 客户端负载均衡

# Spring Cloud 服务熔断