# 基本概念

Spring Cloud Config项目是一个解决分布式系统的配置管理方案。它包含了Client和Server两个部分，server提供配置文件的存储、以接口的形式将配置文件的内容提供出去，client通过接口获取数据、并依据此数据初始化自己的应用。

通过 Config Server (配置中心)可以管理 Config Client (应用程序)的外部属性。应用程序可以通过bootstrap.yml配置文件来加载配置中心下指定配置环境的属性。

# 传统配置的痛点

1、在以前的项目中，我们通过配置文件、操作系统变量、Java系统属性等方式配置Java项目；在spring boot爆火之后我们的配置信息都写在application.yml或application.properties文件中，这些配置文件随着项目的打包与应用一起发布；但是当我们需要修改配置文件中的配置信息的时候，需要更新配置文件重新构建、重新发布，如果配置信息配置在操作系统环境变量或者Java系统属性中则需要重启应用。

2、配置文件中往往有一些敏感信息，比如数据库密码、Redis密码、加密秘钥等信息。这些信息如果直接配置在配置文件中容易泄露。

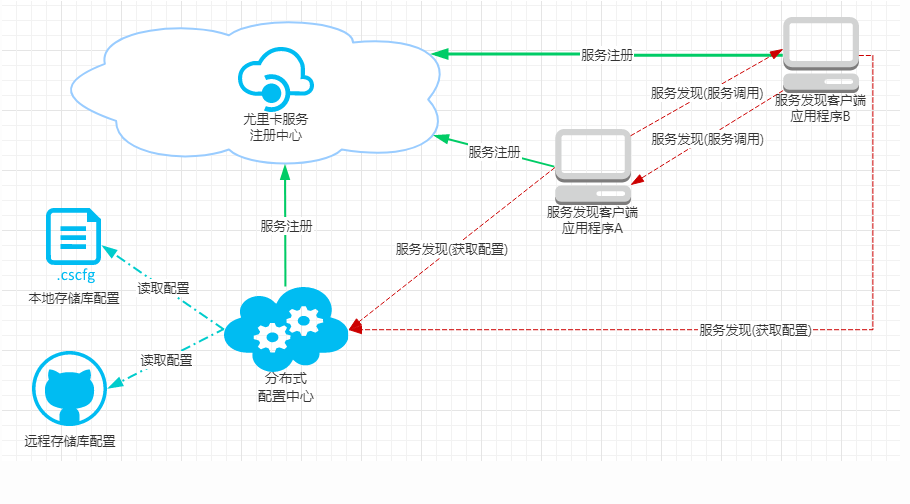
# Config的优点

1、配置文件与应用解耦，可以在不重启应用的前提下随时更新发布、回滚配置文件。

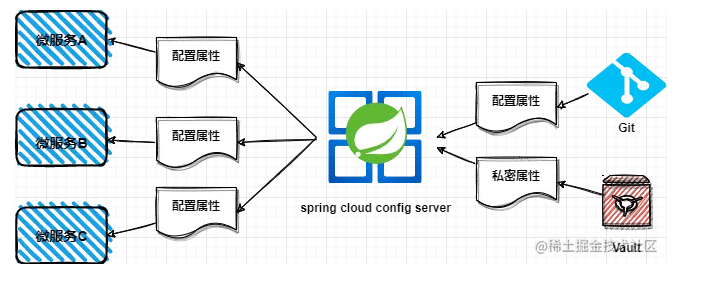
2、不同的服务可以共享配置，这在微服务架构系统中非常有用，避免重复配置，大大降低了微服务配置的维护成本。

3、配置与应用隔离之后，敏感信息得到保护。

# 架构图



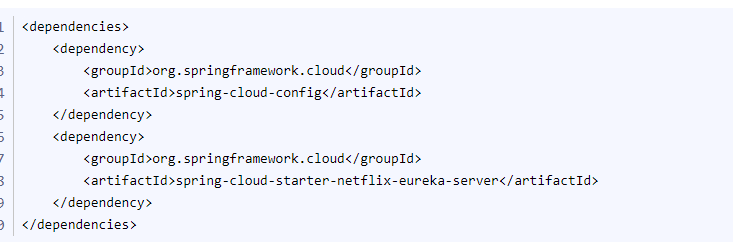
Spring Cloud Config Server通过Git仓库给微服务提供配置属性架构图



# 示例

## 1、添加依赖

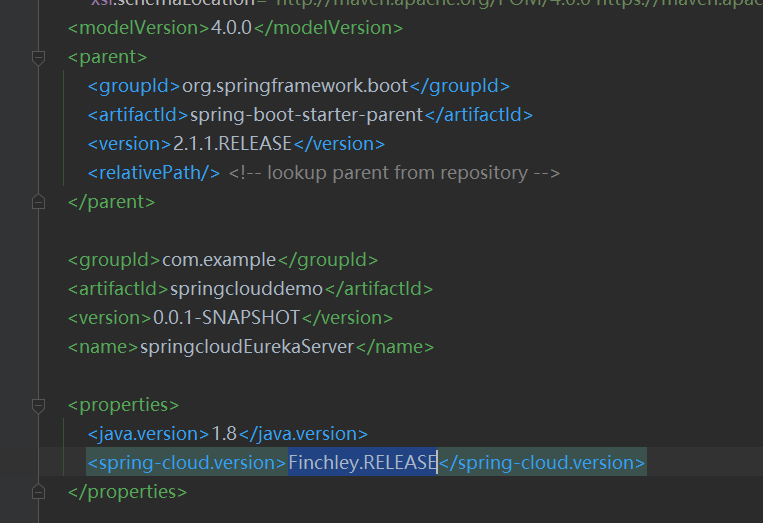
说明：注意的是SpringBoot2.x以后，jdk的版本必须是1.8以上。

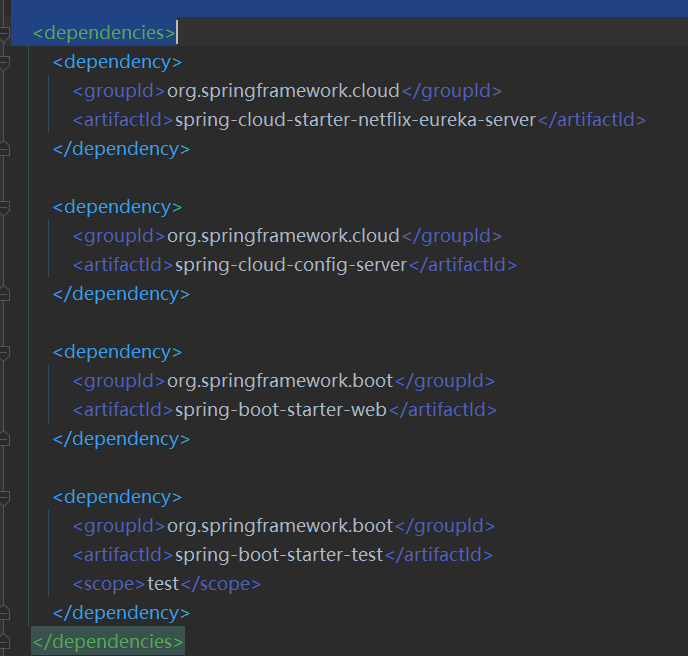


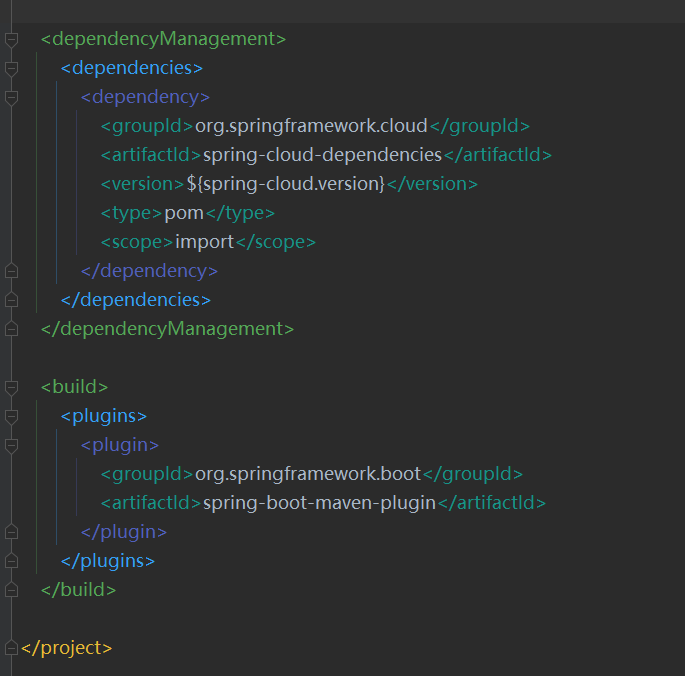
目前SpringCloud Config的使用主要是通过Git/SVN方式做一个配置中心，然后每个服务从其中获取自身配置所需的参数。SpringCloud Config也支持本地参数配置的获取。如果使用本地存储的方式，在 application.properties 或 application.yml 文件添加 spring.profiles.active=native 配置即可，它会从项目的 resources路径下读取配置文件。如果是读取指定的配置文件，那么可以使用 spring.cloud.config.server.native.searchLocations = file:D:/properties/ 来读取。

## 2、创建注册中心服务端

### pom.xml文件

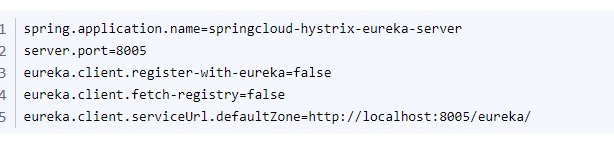






### 编写application.properties文件

首先创建一个Eureka注册中心，为了进行区分，创建一个springcloud-config-eureka的项目。在application.properties文件中的内容如下：



配置说明：

spring.application.name： 这个是指定服务名称。

server.port:服务指定的端口。

eureka.client.register-with-eureka：表示是否将自己注册到Eureka Server，默认是true。

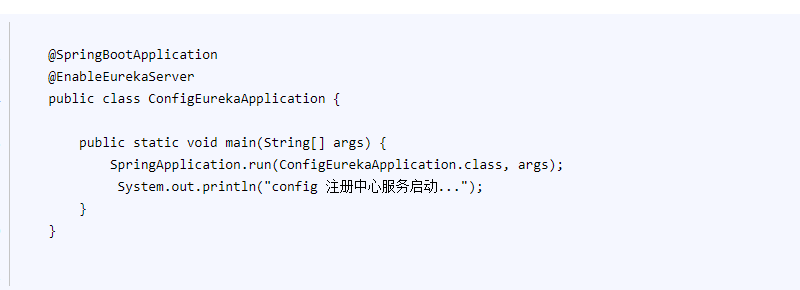
eureka.client.fetch-registry：表示是否从Eureka Server获取注册信息，默认为true。

eureka.client.serviceUrl.defaultZone： 这个是设置与Eureka Server交互的地址，客户端的查询服务和注册服务都需要依赖这个地址。

### 编写启动类

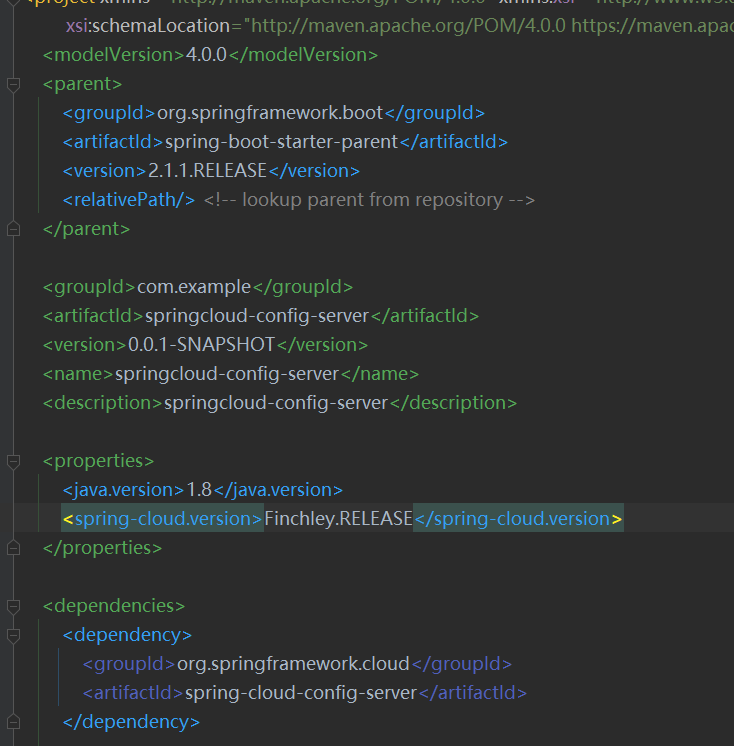
服务端这边只需要在SpringBoot启动类添加@EnableEurekaServer注解就可以了，该注解表示此服务是一个服务注册中心服务。

代码示例：

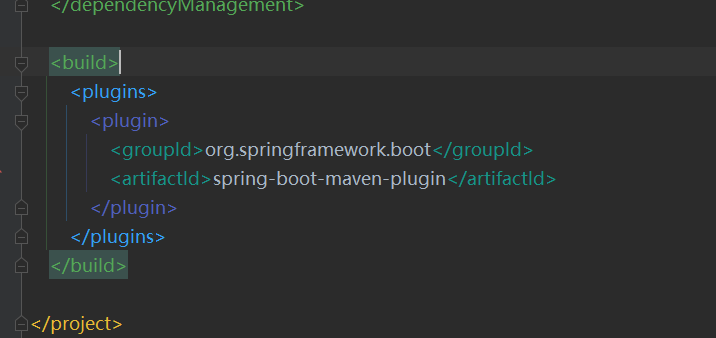


## 3、创建配置中心服务端

### pom.xml文件







### 编写application.properties配置文件

创建一个springcloud-config-server的项目。然后在application.properties配置文件添加如下配置



配置说明：

spring.application.name： 这个是指定服务名称。

server.port:服务指定的端口。

eureka.client.serviceUrl.defaultZone： 这个是设置与Eureka Server交互的地址，客户端的查询服务和注册服务都需要依赖这个地址。

spring.cloud.config.server.git.uri: 配置Git仓库的地址。

spring.cloud.config.server.git.search-paths: git仓库地址下的相对地址，多个用逗号","分割。

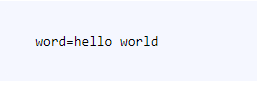
spring.cloud.config.server.git.username:git仓库的账号。

spring.cloud.config.server.git.password:git仓库的密码。

注:如果想使用本地方式读取配置信息，那么只需将spring.cloud.config.server.git的配置改成spring.profiles.active=native，然后在resources路径下新增一个文件即可。

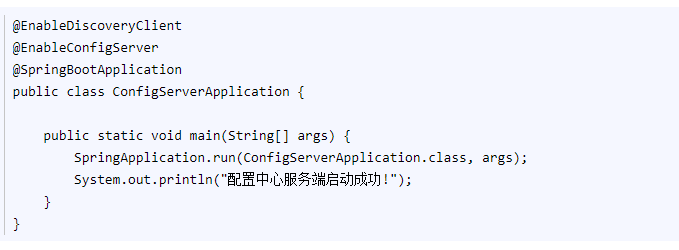
### 创建配置文件

为了进行本地配置文件测试，新建一个configtest.properties配置文件，添加如下内容:



代码这块也很简单，在程序主类中，额外添加@EnableConfigServer注解，该注解表示启用config配置中心功能。

代码如下:

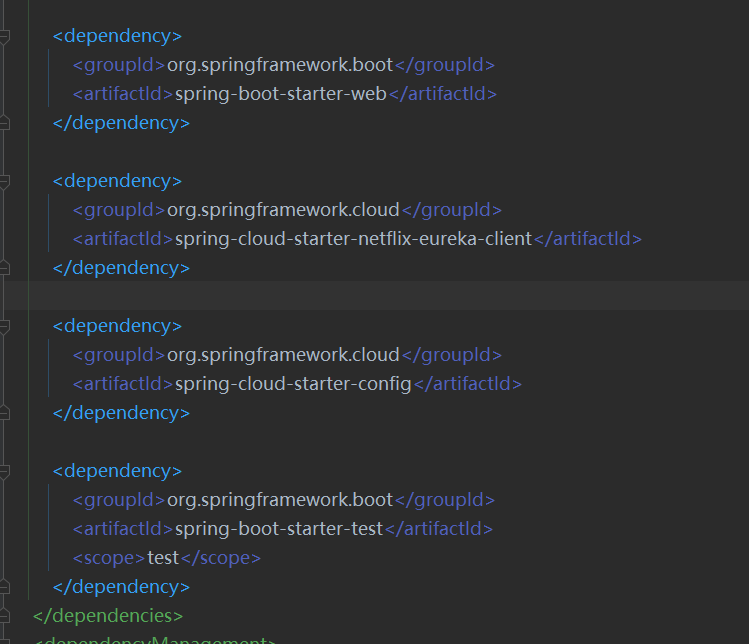


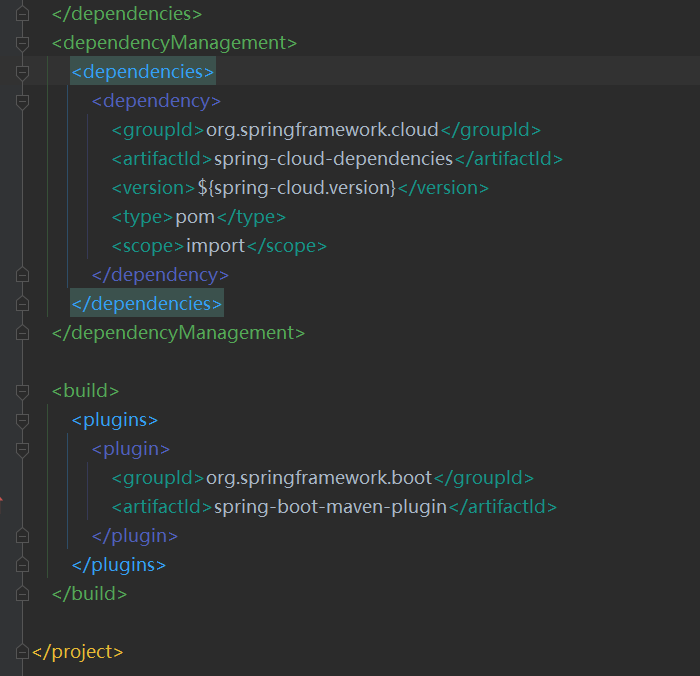
我们的配置中心服务端已经构建完成了。

## 4、创建配置中心客户端

### pom.xml文件



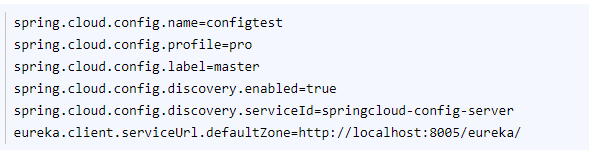




我们新建一个springcloud-config-client的项目，用于读取配置中心的配置信息，pom依赖还是和配置中心一样，不过需要新增一个配置文件，用于指定配置的读取。

### 创建bootstrap.properties配置文件

创建一个bootstrap.properties文件，并添加如下信息:



配置说明：

spring.cloud.config.name： 获取配置文件的名称。

spring.cloud.config.profile: 获取配置的策略。

spring.cloud.config.label：获取配置文件的分支，默认是master。如果是本地获取的话，则无用。

spring.cloud.config.discovery.enabled: 开启配置信息发现。

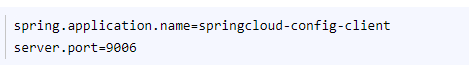
spring.cloud.config.discovery.serviceId: 指定配置中心的service-id，便于扩展为高可用配置集群。

eureka.client.serviceUrl.defaultZone： 这个是设置与Eureka Server交互的地址，客户端的查询服务和注册服务都需要依赖这个地址。

注意：上面这些与spring-cloud相关的属性必须配置在bootstrap.properties中，config部分内容才能被正确加载。因为bootstrap.properties的相关配置会先于application.properties，而bootstrap.properties的加载也是先于application.properties。需要注意的是eureka.client.serviceUrl.defaultZone要配置在bootstrap.properties，不然客户端是无法获取配置中心参数的，会启动失败。

### 创建****application.properties配置文件****

配置文件内容如下：



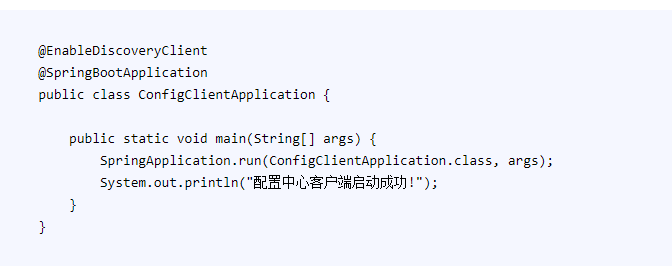
配置说明：

[spring.application.name](http://spring.application.name/)： 这个是指定服务名称。

server.port:服务指定的端口。

### 编写程序入口类

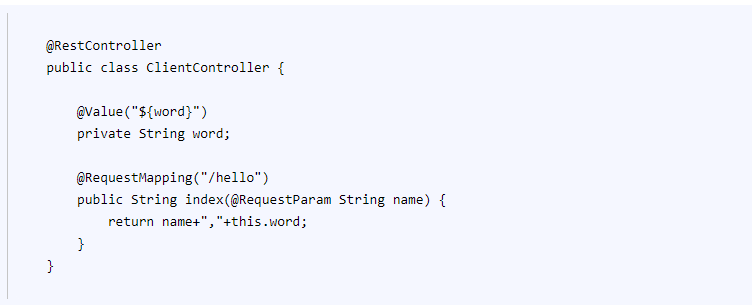
程序主代码如下：



### 编写Controller类去测试

为了方便查询，在控制中进行参数的获取并返回。@Value注解是默认是从application.properties配置文件获取参数，但是这里我们在客户端并没有进行配置，该配置在配置中心服务端，我们只需指定好了配置文件之后即可进行使用。

代码示例



到此客户端项目也就构建完成了。

## 5、本地测试

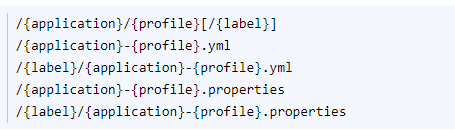
首先我们把springcloud-config-server项目的application.properties配置文件添加spring.profiles.active=native配置，注释掉spring.cloud.config.server.git相关的配置，然后在src/main/resources目录下新建一个configtest.properties文件，然后在里面添加一个配置word=hello world。

添加完成之后，我们依次启动springcloud-config-eureka、springcloud-config-server、springcloud-config-client这三个项目。启动成功之前，先看来看看配置中心服务端的配置文件获取，在浏览器输入:

http://localhost:9005/configtest-1.properties

**注**:配置文件的名称是configtest.properties，但是如果直接该名称的话是获取不到的，因为在配置文件名需要通过-来进行获取，如果配置文件名称没有-，那么添加了-之后，会自动进行匹配搜索。

springcloud config 的URL与配置文件的映射关系如下:



上面的url会映射{application}-{profile}.properties对应的配置文件，{label}对应git上不同的分支，默认为master。

界面返回如下：



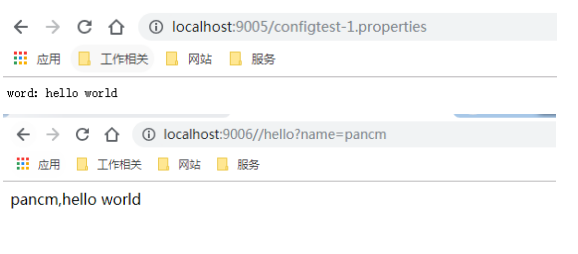
然后调用客户端的接口，查看是否能够获取配置信息。在浏览器上输入:

http://localhost:9006/hello?name=pancm

界面返回：



示例图：



## 6、git测试

我们把这个spring.profiles.active=native配置注释掉，解除spring.cloud.config.server.git相关的注释(账号和密码要填写真实的)，然后在git仓库上建立一个config-repo 文件夹，新建configtest-pro.properties、configtest-dev.properties两个配置，这两个的配置分别是word=hello world！！和word=hello world！， 然后和configtest.properties配置文件一起上传到config-repo 文件夹中。

首先在浏览器输入：

<http://localhost:9005/configtest-dev.properties>

浏览器返回：



然后再浏览器输入:

<http://localhost:9005/configtest-pro.properties>

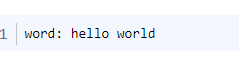
浏览器返回：



上传了configtest.properties文件，但是这个文件名称没有-,我们想获取其中参数的信息的话，可以在然后-随意添加一个参数，它会自动进行匹配，在浏览器输入:

<http://localhost:9005/configtest-1.properties>

浏览器返回：



然后进行客户端接口调用测试，在浏览器输入:

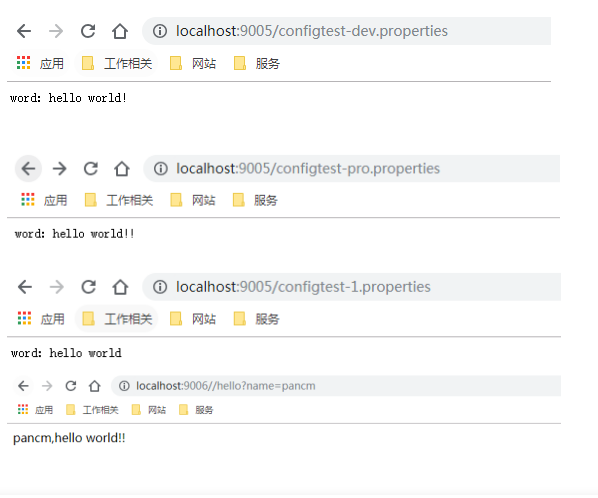
http://localhost:9006/hello?name=pancm

浏览器返回：



由于这里我配置的前缀是 pro ，所以读取的是 **configtest-pro.properties** 文件的数据，想要获取其他的配置，修改spring.cloud.config.profile配置即可。

示例：



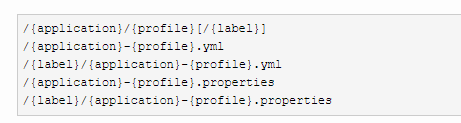
# 配置中心服务端配置

## 配置文件命名规范

1、{label} 是可选的 git 标签，默认 master；

2、{profile}映射到客户端上的"spring.profiles.active" 或 “spring.cloud.config.profile”; 是可选的环境配置，常见有 local，dev，test，prod；

3、{application} 映射到客户端的"spring.application.name" 或 “spring.cloud.config.name”;



## 配置环境支持

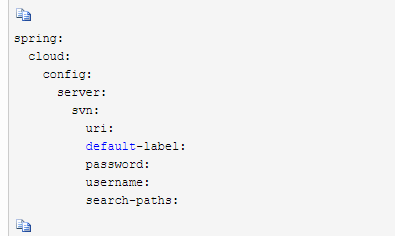
配置中心环境配置支持 git 存储器，并支持 label 来控制环境配置版本；也支持 svn 存储器；同时也支持本地配置，可以不使用远程存储器。

### git配置

配置中心 git 存储器；  
uri 是仓库路径；  
username/password 账户/密码，此处是 GitHub 账户密码；  
default-label 是可选的 git 标签，默认 master；  
search-paths 配置文件所在路径，全路径即：https://github.com/niaonao/spring-cloud/doc/config

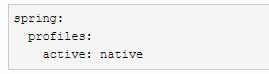


### svn配置



### native本地配置

配置中心会读取本地配置文件，配置文件路径 /src/main/resources，如下图在resources 下创建配置文件 /{application}-{profile}.yml



# 常用的注解

## @EnableConfigServer