

Spring-cloud 入坑系列之 cloud-config

作者: [Myike](#)

原文链接: <https://ld246.com/article/1568270659822>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

入门介绍:

作为 Spring-cloud 全家桶下的配置中心, cloud-config 的出现实现了配置文件与项目的解耦; 也可避免环境配置的泄露, 一定程度上减轻了运维工作人员的工作量。说到配置中心, 最近公司在使用携开源的 apollo 分布式配置中心, 提供了各版本环境配置的可视化管理工具。相信这种独立的配置管会是以后 Web 开发中的一大趋势。下面介绍一下 Spring-cloud 中的 Spring-cloud-config 组件:

首先来个入门案例, 创建一个基本的 Spring-boot 项目, 并引入 Spring-cloud-config 依赖:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Greenwich.SR1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

第二步则是在启动类上添加配置注解@EnableConfigServer,接下来就是配置中心的相关配置; 在 application.YAM 的配置如下:

```
Spring:
  application:
    name: config-server
  cloud:
    config:
      server:
        Git:
          uri: Git@github.com:xxxx/Spring-cloud-config-demo.Git
          # ignore-local-SSH-settings: true
          # private-key: |
          # -----BEGIN RSA PRIVATE KEY-----
          # -----END RSA PRIVATE KEY-----
          # hostKeyAlgorithm: SSH-rsa
          # host-key:
          # strict-host-key-checking: false
          # force-pull: true
          # search-paths: config-test
          # username:
          # password:
        server:
          port: 7001
```

上面标红的配置使用的是 SSH 协议的仓库地址, 当然你也可以使用 http 协议地址; 使用 http 协议话可以配置 username 和 password 进行认证; 而 SSH 地址则需要配置公钥认证, 上图配置中, 我配置了一个 Git 仓库地址, 并没有配置公钥, 是因为程序会默认使用当前系统用户下的.SSH/的配置这也为你省去了配置的烦恼。当然你可以改变相应的配置来配合你自身的 Git 仓库。比如我的配置:

```
Host "xx.xx.xx.xx";
Port 3001
User "Git";
identityfile C:\Users\Win10.SSH\id_rsa
```

因为我的 Git 仓库是搭建在自己云服务器上的 Git 服务端，而 SSH 端口默认是 22，这会导致一定的网络攻击，所以我把服务器的 SSH 协议的端口改了，因此你本机的配置也需要相应的改变，不然无法进行 SSH 通信的。

现在说一说如果不使用本机默认的配置，该怎么配置。首先第一步需要把 ignore-local-SSH-settings: true 放开，该配置的启用忽略了本机默认的配置；同时还必须要配置 private-key，不然系统无法认证；private-key 也就是你本机的秘钥，这里需要注意的是 private-key 必须要按照指定的格式，即：|privatekey，前面的“|”不能忽略，private-key 全部拷过来就行。还有一个配置选项是 strict-host-key-checking，该配置是用来检测你本机.SSH/known_hosts 中是否有该仓库主机的记录；如果你没有接过，请设置为 false。说一下我遇到的问题吧，因为我的 Git 服务是安装在阿里云服务器上；有几次被别人攻击了添加了对外攻击脚本，因此将 SSH 端口修改了其他端口。那么 Git 客户端也需要修改连接的端口，Spring-cloud-config 中的 Git 配置是无法指定端口的；因此只能在 Git 配置 config 文中指定端口，如上图的 Port:3001。不使用本地配置是无法修改端口的，所以只能使用本地配置。还有一点，阿里云服务器的 SSH 的公钥配置和 Git 服务器的公钥配置不一定是同一个；因此你需要连接 Git 的话你只需要添加公钥到 Git 安装路径下的配置文件即可。以上就是搭建的过程，另外最好禁止 Git 客户端使用客户端登录服务器，具体的做法是：

```
vim /etc/passwd
#git:x:1000:1000::/home/git:/bin/bash //原来配置
git:x:1000:1000::/home/git:/usr/local/git/bin/git-shell //后面的路径为git的git-shell路径
```

这样就可以禁止 Git 用户使用 SSH 登录服务器，当然你还可以结合其他 Git 权限管理的其他工具管相应的权限。

2.动态刷新：可以将新的配置动态加载到内存中，省去了重启应用的时间

具体实现，首先引入依赖：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

actuator模块提供了项目状态监测的功能，其次需要暴露动态更新的接口，在配置中

```
management.endpoints.web.exposure.include=* //全部开放
management.endpoints.web.exposure.include=refresh,env //开放部分接口
```

现在发送一个 POST 请求/refresh 至配置客户端便可更新配置

3.通过消息总线 +Rabbitmq 提供配置动态更新

加入依赖：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
```

```

</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>

```

Rabbitmq配置:

```

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
management.endpoints.web.exposure.include=* //全部开放
management.endpoints.web.exposure.include=refresh,env //开放部分接口

```

其思想就是利用 Git 的 hook 钩子功能，在 Git 有提交的修改后会触发项目提供的 refresh 接口，rabbitmq 便会收到发送更新配置的通知到其他的配置客户端，便可同步动态更新配置，下面需要配置一下 GitHub 的 webhook 回调接口

需要注意的是，GitHub 默认会在请求中加上相关提交的载荷:

如果你没有配置消息转化器去处理，便会如下的错:

```

{"timestamp": "2019-07-17T03:22:36.859+0000",
 "status": 400,
 "error": "Bad Request",
 "message": "JS
N parse error: Cannot deserialize instance of java.lang.String out of START_ARRAY token; nested exception is com.fasterxml.jackson.databind.exc.MismatchedInputException: Cannot deserialize instance of java.lang.String out of START_ARRAY token\n at [Source: (PushbackInputStream; line: 1, column: 304) (through reference chain: java.util.LinkedHashMap[\"commits\"]);\"; path: \"/actuator/bus-refresh\";}

```

因此需要处理相应的头部，代码如下:

```

@WebFilter(filterName = "bodyFilter", urlPatterns = "/*")
@Order(1)
public class RequestFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest)servletRequest;
        String url = new String(httpRequest.getRequestURI());
    }
}

```

```
//只过滤/actuator/bus-refresh请求
if (!url.endsWith("/bus-refresh")) {
chain.doFilter(servletRequest, response);
return;
}
```

```
    //使用HttpServletRequest包装原始请求达到修改post请求中body内容的目的
    CustometRequestWrapper requestWrapper = new CustometRequestWrapper(httpServle
Request);
    chain.doFilter(requestWrapper, response);
}
@Override
public void destroy() {
}
}
```

上面是一个过滤器

```
public class CustometRequestWrapper extends HttpServletRequestWrapper {
public CustometRequestWrapper(HttpServletRequest request) {
super(request);
}
@Override
public ServletInputStream getInputStream() throws IOException {
byte[] bytes = new byte[0];
ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(bytes);
return new ServletInputStream() {
@Override
public boolean isFinished() {
return byteArrayInputStream.read() == -1 ? true:false;
}
@Override
public boolean isReady() {
return false;
}
@Override
public void setReadListener(ReadListener readListener) {}

@Override
public int read() throws IOException {
return byteArrayInputStream.read();
}
}
```

```
};  
}  
}
```

上面是一个消息处理器
另外需要在启动类加上@ServletComponentScan 注解就行