Spring Boot 外部化配置实战解析

来源：https://github.com/shijw823/springboot-externalized-configuration-extend.git

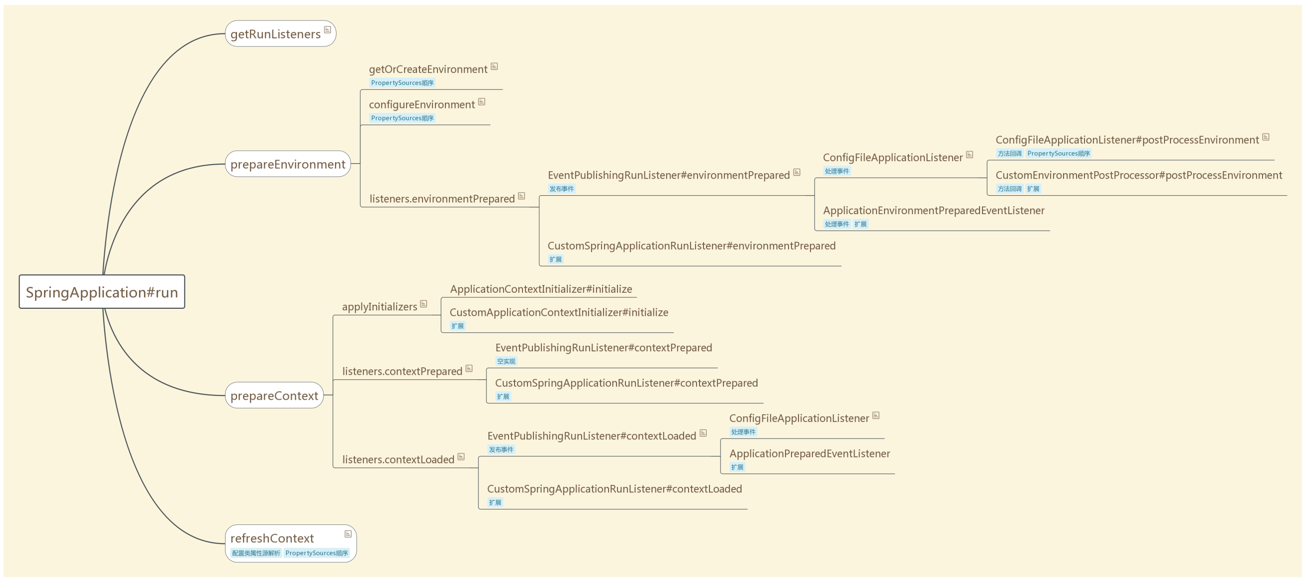
一、流程分析

入口程序

在 `SpringApplication#run(String... args)` 方法中，外部化配置关键流程分为以下四步

```java
public ConfigurableApplicationContext run(String... args) {
    ...
    SpringApplicationRunListeners listeners = getRunListeners(args); // 1
    listeners.starting();
    try {
        ApplicationArguments applicationArguments = new DefaultApplicationArguments(
            args);
        ConfigurableEnvironment environment = prepareEnvironment(listeners,
                                                                 applicationArguments); // 2
        configureIgnoreBeanInfo(environment);
        Banner printedBanner = printBanner(environment);
        context = createApplicationContext();
        exceptionReporters = getSpringFactoriesInstances(
            SpringBootExceptionReporter.class,
            new Class[] { ConfigurableApplicationContext.class }, context);
        prepareContext(context, environment, listeners, applicationArguments,
                       printedBanner); // 3
        refreshContext(context); // 4
        afterRefresh(context, applicationArguments);
        stopWatch.stop();
        if (this.logStartupInfo) {
            new StartupInfoLogger(this.mainApplicationClass)
                .logStarted(getApplicationLog(), stopWatch);
        }
        listeners.started(context);
        callRunners(context, applicationArguments);
    }
    ...
}
```

关键流程思维导图

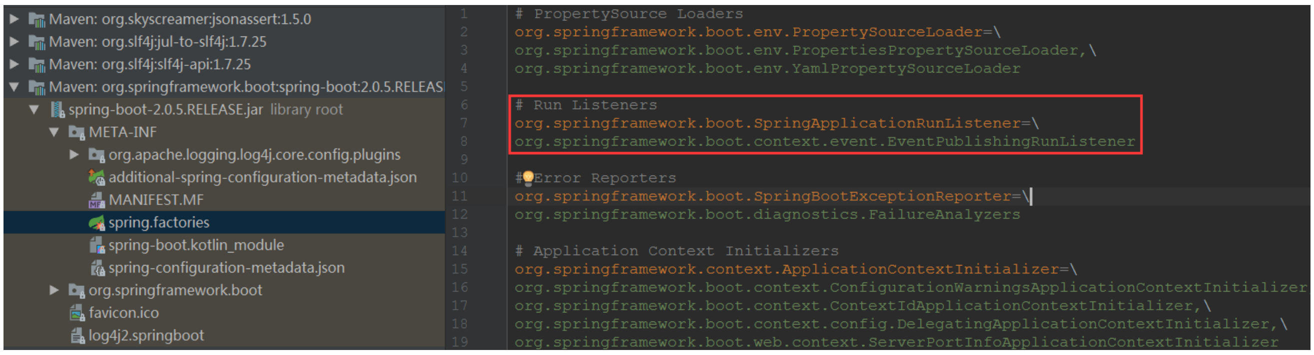

SpringApplication#run
getRunListeners
prepareEnvironment
getOrCreateEnvironment
PropertySources顺序
configureEnvironment
PropertySources顺序
listeners.environmentPrepared
EventPublishingRunListener#environmentPrepared
发布事件
ConfigFileApplicationListener
处理事件
ConfigFileApplicationListener#postProcessEnvironment
方法回调 PropertySources顺序
CustomEnvironmentPostProcessor#postProcessEnvironment
方法回调 扩展
ApplicationEnvironmentPreparedEventListener
处理事件 扩展
CustomSpringApplicationRunListener#environmentPrepared
扩展
prepareContext
applyInitializers
ApplicationContextInitializer#initialize
CustomApplicationContextInitializer#initialize
扩展
listeners.contextPrepared
EventPublishingRunListener#contextPrepared
空实现
CustomSpringApplicationRunListener#contextPrepared
扩展
listeners.contextLoaded
EventPublishingRunListener#contextLoaded
发布事件
ConfigFileApplicationListener
处理事件
ApplicationPreparedEventListener
扩展
CustomSpringApplicationRunListener#contextLoaded
扩展
refreshContext
配置类属性源解析 PropertySources顺序

关键流程详解

对入口程序中标记的四步，分析如下

1、`SpringApplication#getRunListeners`

加载 `META-INF/spring.factories` 获取 `SpringApplicationRunListener` 的实例集合，存放的对象是 `EventPublishingRunListener` 类型 以及自定义的 `SpringApplicationRunListener` 实现类型



```
▶ Maven: org.skyscreamer:jsonassert:1.5.0
▶ Maven: org.slf4j:jul-to-slf4j:1.7.25
▶ Maven: org.slf4j:slf4j-api:1.7.25
▼ Maven: org.springframework.boot:spring-boot:2.0.5.RELEAS
  ▼ spring-boot-2.0.5.RELEASE.jar library root
    ▼ META-INF
      ▶ org.apache.logging.log4j.core.config.plugins
        additional-spring-configuration-metadata.json
        MANIFEST.MF
        spring.factories
        spring-boot.kotlin_module
        spring-configuration-metadata.json
    ▶ org.springframework.boot
      favicon.ico
      log4j2.springboot
```

```
# PropertySource Loaders
org.springframework.boot.env.PropertySourceLoader=\
org.springframework.boot.env.PropertiesPropertySourceLoader,\
org.springframework.boot.env.YamlPropertySourceLoader

# Run Listeners
org.springframework.boot.SpringApplicationRunListener=\
org.springframework.boot.context.event.EventPublishingRunListener

#Error Reporters
org.springframework.boot.SpringBootExceptionReporter=\
org.springframework.boot.diagnostics.FailureAnalyzers

# Application Context Initializers
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.context.ConfigurationWarningsApplicationContextInitializer
org.springframework.boot.context.ContextIdApplicationContextInitializer,\
org.springframework.boot.context.config.DelegatingApplicationContextInitializer,\
org.springframework.boot.web.context.ServerPortInfoApplicationContextInitializer
```

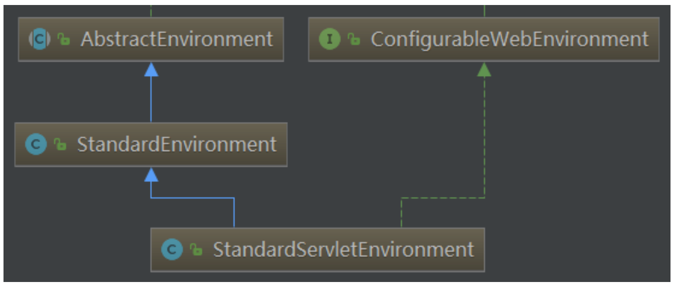
2、`SpringApplication#prepareEnvironment`

`prepareEnvironment` 方法中，主要的三步如下

```
private ConfigurableEnvironment prepareEnvironment(SpringApplicationRunListeners listeners,
    ApplicationArguments applicationArguments) {
    // Create and configure the environment
    ConfigurableEnvironment environment = getOrCreateEnvironment(); // 2.1
    configureEnvironment(environment, applicationArguments.getSourceArgs()); // 2.2
    listeners.environmentPrepared(environment); // 2.3
    ...
    return environment;
}
```

2.1、 `getOrCreateEnvironment` 方法

在 `WebApplicationType.SERVLET` web应用类型下，会创建 `StandardServletEnvironment`，本文以 `StandardServletEnvironment` 为例，类的层次结构如下



当创建 `StandardServletEnvironment` ， `StandardServletEnvironment` 父类 `AbstractEnvironment` 调用 `customizePropertySources` 方法，会执行 `StandardServletEnvironment#customizePropertySources` 和 `StandardEnvironment#customizePropertySources` ，源码如下

`AbstractEnvironment`

```
public AbstractEnvironment() {
    customizePropertySources(this.propertySources);
    if (logger.isDebugEnabled()) {
        logger.debug("Initialized " + getClass().getSimpleName() + " with PropertySources " +
this.propertySources);
    }
}
```

`StandardServletEnvironment#customizePropertySources`

```
/** Servlet context init parameters property source name: {@value} */
public static final String SERVLET_CONTEXT_PROPERTY_SOURCE_NAME = "servletContextInitParams";

/** Servlet config init parameters property source name: {@value} */
public static final String SERVLET_CONFIG_PROPERTY_SOURCE_NAME = "servletConfigInitParams";

/** JNDI property source name: {@value} */
public static final String JNDI_PROPERTY_SOURCE_NAME = "jndiProperties";

@Override
protected void customizePropertySources(MutablePropertySources propertySources) {
    propertySources.addLast(new StubPropertySource(SERVLET_CONFIG_PROPERTY_SOURCE_NAME));
    propertySources.addLast(new StubPropertySource(SERVLET_CONTEXT_PROPERTY_SOURCE_NAME));
    if (JndiLocatorDelegate.isDefaultJndiEnvironmentAvailable()) {
        propertySources.addLast(new JndiPropertySource(JNDI_PROPERTY_SOURCE_NAME));
    }
    super.customizePropertySources(propertySources);
}
```

`StandardEnvironment#customizePropertySources`

```java
/** System environment property source name: {@value} */
public static final String SYSTEM_ENVIRONMENT_PROPERTY_SOURCE_NAME = "systemEnvironment";

/** JVM system properties property source name: {@value} */
public static final String SYSTEM_PROPERTIES_PROPERTY_SOURCE_NAME = "systemProperties";

@Override
protected void customizePropertySources(MutablePropertySources propertySources) {
    propertySources.addLast(new MapPropertySource(SYSTEM_PROPERTIES_PROPERTY_SOURCE_NAME,
getSystemProperties()));
    propertySources.addLast(new
SystemEnvironmentPropertySource(SYSTEM_ENVIRONMENT_PROPERTY_SOURCE_NAME,getSystemEnvironment());
}
```

`PropertySources` 顺序：

1. servletConfigInitParams
2. servletContextInitParams
3. jndiProperties
4. systemProperties
5. systemEnvironment

> `PropertySources` 与 `PropertySource` 关系为 1 对 N

2.2、 `configureEnvironment` 方法

调用 `configurePropertySources(environment, args)`， 在方法里面设置 `Environment` 的 `PropertySources` ，包含 `defaultProperties` 和 `SimpleCommandLinePropertySource` （commandLineArgs），`PropertySources` 添加 `defaultProperties` 到最后，添加 `SimpleCommandLinePropertySource` （commandLineArgs）到最前面

`PropertySources` 顺序：

1. commandLineArgs
2. servletConfigInitParams
3. servletContextInitParams
4. jndiProperties
5. systemProperties
6. systemEnvironment
7. defaultProperties

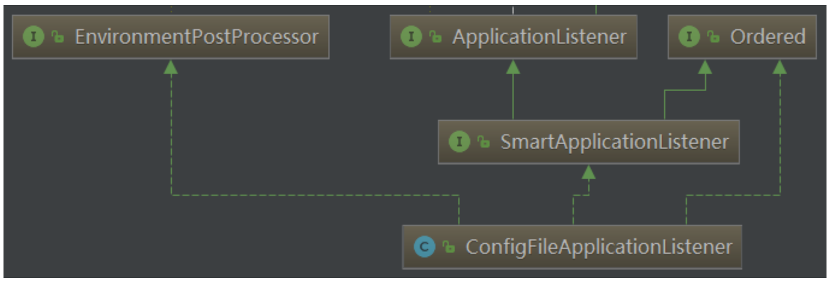
2.3、 `listeners.environmentPrepared` 方法

会按优先级顺序遍历执行 `SpringApplicationRunListener#environmentPrepared`，比如 `EventPublishingRunListener` 和 自定义的 `SpringApplicationRunListener`

- `EventPublishingRunListener` 发布 `ApplicationEnvironmentPreparedEvent` 事件
    - `ConfigFileApplicationListener` 监听 `ApplicationEvent` 事件 、处理 `ApplicationEnvironmentPreparedEvent` 事件，加载所有 `EnvironmentPostProcessor` 包括自己，然后按照顺序进行方法回调

        - `ConfigFileApplicationListener#postProcessEnvironment` 方法回调，然后 `addPropertySources` 方法调用 `RandomValuePropertySource#addToEnvironment`，在 systemEnvironment 后面添加 random，然后添加配置文件的属性源（详见源码 `ConfigFileApplicationListener.Loader#load()`
- 扩展点
    - 自定义 `SpringApplicationRunListener`，重写 `environmentPrepared` 方法
    - 自定义 `EnvironmentPostProcessor`
    - 自定义 `ApplicationListener` 监听 `ApplicationEnvironmentPreparedEvent` 事件

`ConfigFileApplicationListener`，即是 `EnvironmentPostProcessor`，又是 `ApplicationListener`，类的层次结构如下



```java
@Override
public void onApplicationEvent(ApplicationEvent event) {
    // 处理 ApplicationEnvironmentPreparedEvent 事件
    if (event instanceof ApplicationEnvironmentPreparedEvent) {
        onApplicationEnvironmentPreparedEvent(
            (ApplicationEnvironmentPreparedEvent) event);
    }
    // 处理 ApplicationPreparedEvent 事件
    if (event instanceof ApplicationPreparedEvent) {
        onApplicationPreparedEvent(event);
    }
}

private void onApplicationEnvironmentPreparedEvent(
    ApplicationEnvironmentPreparedEvent event) {
    // 加载 META-INF/spring.factories 中配置的 EnvironmentPostProcessor
    List<EnvironmentPostProcessor> postProcessors = loadPostProcessors();
    // 加载自己 ConfigFileApplicationListener
    postProcessors.add(this);
    // 按照 Ordered 进行优先级排序
    AnnotationAwareOrderComparator.sort(postProcessors);
    // 回调 EnvironmentPostProcessor
    for (EnvironmentPostProcessor postProcessor : postProcessors) {
        postProcessor.postProcessEnvironment(event.getEnvironment(),
                                             event.getSpringApplication());
    }
}

List<EnvironmentPostProcessor> loadPostProcessors() {
    return SpringFactoriesLoader.loadFactories(EnvironmentPostProcessor.class,
                                               getClass().getClassLoader());
}
```

```java
@Override
public void postProcessEnvironment(ConfigurableEnvironment environment,
                                   SpringApplication application) {
    addPropertySources(environment, application.getResourceLoader());
}

/**
  * Add config file property sources to the specified environment.
  * @param environment the environment to add source to
  * @param resourceLoader the resource loader
  * @see #addPostProcessors(ConfigurableApplicationContext)
  */
protected void addPropertySources(ConfigurableEnvironment environment,
                                  ResourceLoader resourceLoader) {
    RandomValuePropertySource.addToEnvironment(environment);
    // 添加配置文件的属性源
    new Loader(environment, resourceLoader).load();
}
```

`RandomValuePropertySource`

```java
public static void addToEnvironment(ConfigurableEnvironment environment) {
    // 在 systemEnvironment 后面添加 random
    environment.getPropertySources().addAfter(
        StandardEnvironment.SYSTEM_ENVIRONMENT_PROPERTY_SOURCE_NAME,
        new RandomValuePropertySource(RANDOM_PROPERTY_SOURCE_NAME));
    logger.trace("RandomValuePropertySource add to Environment");
}
```

> 添加配置文件的属性源：
>
> 执行 `new Loader(environment, resourceLoader).load();` ， 调用 `load(Profile, DocumentFilterFactory, DocumentConsumer)` (getSearchLocations() 获取配置文件位置，可以指定通过 spring.config.additional-location 、spring.config.location 、spring.config.name 参数或者使用默认值) ， 然后调用 `addLoadedPropertySources -> addLoadedPropertySource` (加载 查找出来的 `PropertySource` 到 `PropertySources` ，并确保放置到 defaultProperties 的前面)
>
> 默认的查找位置，配置为 `"classpath:/,classpath:/config/,file:./,file:./config/"` ， 查找顺序从后向前

`PropertySources` 顺序：

1. commandLineArgs
2. servletConfigInitParams
3. servletContextInitParams
4. jndiProperties
5. systemProperties
6. systemEnvironment
7. random
8. application.properties ...
9. defaultProperties

3、`SpringApplication#prepareContext`

`prepareContext` 方法中，主要的三步如下

```
private void prepareContext(ConfigurableApplicationContext context,
                            ConfigurableEnvironment environment,
                            SpringApplicationRunListeners listeners,
                            ApplicationArguments applicationArguments,
                            Banner printedBanner) {
    ...
    applyInitializers(context); // 3.1
    listeners.contextPrepared(context); //3.2
    ...
    listeners.contextLoaded(context); // 3.3
}
```

3.1、`applyInitializers` 方法

会遍历执行所有的 `ApplicationContextInitializer#initialize`

- 扩展点
  - 自定义 `ApplicationContextInitializer`

3.2、`listeners.contextPrepared` 方法

会按优先级顺序遍历执行 `SpringApplicationRunListener#contextPrepared`，比如 `EventPublishingRunListener` 和 自定义的 `SpringApplicationRunListener`

- 扩展点
  - 自定义 `SpringApplicationRunListener` ，重写 `contextPrepared` 方法

3.3、`listeners.contextLoaded` 方法

会按优先级顺序遍历执行 `SpringApplicationRunListener#contextLoaded`，比如 `EventPublishingRunListener` 和 自定义的 `SpringApplicationRunListener`

- `EventPublishingRunListener` 发布 `ApplicationPreparedEvent` 事件
  - `ConfigFileApplicationListener` 监听 `ApplicationEvent` 事件 处理 `ApplicationPreparedEvent` 事件
- 扩展点
  - 自定义 `SpringApplicationRunListener` ，重写 `contextLoaded` 方法
  - 自定义 `ApplicationListener` ，监听 `ApplicationPreparedEvent` 事件

`ConfigFileApplicationListener`

```
@Override
public void onApplicationEvent(ApplicationEvent event) {
    // 处理 ApplicationEnvironmentPreparedEvent 事件
    if (event instanceof ApplicationEnvironmentPreparedEvent) {
        onApplicationEnvironmentPreparedEvent(
            (ApplicationEnvironmentPreparedEvent) event);
    }
    // 处理 ApplicationPreparedEvent 事件
    if (event instanceof ApplicationPreparedEvent) {
        onApplicationPreparedEvent(event);
    }
}

private void onApplicationPreparedEvent(ApplicationEvent event) {
    this.logger.replayTo(ConfigFileApplicationListener.class);
    addPostProcessors(((ApplicationPreparedEvent) event).getApplicationContext());
}

// 添加 PropertySourceOrderingPostProcessor 处理器，配置 PropertySources
protected void addPostProcessors(ConfigurableApplicationContext context) {
    context.addBeanFactoryPostProcessor(
        new PropertySourceOrderingPostProcessor(context));
}
```

`PropertySourceOrderingPostProcessor`

```
// 回调处理（在配置类属性源解析）
@Override
public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory)
    throws BeansException {
    reorderSources(this.context.getEnvironment());
}

// 调整 PropertySources 顺序，先删除 defaultProperties， 再把 defaultProperties 添加到最后
private void reorderSources(ConfigurableEnvironment environment) {
    PropertySource<?> defaultProperties = environment.getPropertySources()
        .remove(DEFAULT_PROPERTIES);
    if (defaultProperties != null) {
        environment.getPropertySources().addLast(defaultProperties);
    }
}
```

> `PropertySourceOrderingPostProcessor` 是 `BeanFactoryPostProcessor`

4、`SpringApplication#refreshContext`

会进行 `@Configuration` 配置类属性源解析，处理 `@PropertySource` annotations on your `@Configuration` classes，但顺序是在 defaultProperties 之后，下面会把 defaultProperties 调整到最后

`AbstractApplicationContext#refresh` 调用 `invokeBeanFactoryPostProcessors` (`PostProcessorRegistrationDelegate#invokeBeanFactoryPostProcessors`)， 然后进行 `BeanFactoryPostProcessor` 的回调处理 ，比如 `PropertySourceOrderingPostProcessor` 的回调（源码见上文）

`PropertySources` 顺序：

1. commandLineArgs
2. servletConfigInitParams
3. servletContextInitParams
4. jndiProperties
5. systemProperties
6. systemEnvironment
7. random
8. application.properties ...
9. `@PropertySource` annotations on your `@Configuration` classes
10. defaultProperties

> 不推荐使用这种方式，推荐使用在 refreshContext 之前准备好，`@PropertySource` 加载太晚，不会对自动配置产生任何影响

二、扩展外部化配置属性源

1、基于 `EnvironmentPostProcessor` 扩展

```java
public class CustomEnvironmentPostProcessor implements EnvironmentPostProcessor
```

2、基于 `ApplicationEnvironmentPreparedEvent` 扩展

```java
public class ApplicationEnvironmentPreparedEventListener implements
ApplicationListener<ApplicationEnvironmentPreparedEvent>
```

3、基于 `SpringApplicationRunListener` 扩展

```java
public class CustomSpringApplicationRunListener implements SpringApplicationRunListener, Ordered
```

> 可以重写方法 environmentPrepared、contextPrepared、contextLoaded 进行扩展

4、基于 `ApplicationContextInitializer` 扩展

```java
public class CustomApplicationContextInitializer implements ApplicationContextInitializer
```

> 关于与 Spring Cloud Config Client 整合，对外部化配置加载的扩展（绑定到Config Server，使用远端的 property sources 初始化 `Environment`），参考源码 `PropertySourceBootstrapConfiguration`（是对 `ApplicationContextInitializer` 的扩展）、`ConfigServicePropertySourceLocator#locate`
>
> 获取远端的property sources是 `RestTemplate` 通过向 http://{spring.cloud.config.uri}/{spring.application.name}/{spring.cloud.config.profile}/{spring.cloud.config.label} 发送 GET 请求方式获取的

5、基于 `ApplicationPreparedEvent` 扩展

```java
public class ApplicationPreparedEventListener implements
ApplicationListener<ApplicationPreparedEvent>
```

6、扩展实战

6.1、扩展配置

在 classpath 下添加配置文件 `META-INF/spring.factories`， 内容如下

```
# Spring Application Run Listeners
org.springframework.boot.SpringApplicationRunListener=\
springboot.propertysource.extend.listener.CustomSpringApplicationRunListener

# Application Context Initializers
org.springframework.context.ApplicationContextInitializer=\
springboot.propertysource.extend.initializer.CustomApplicationContextInitializer

# Application Listeners
org.springframework.context.ApplicationListener=\
springboot.propertysource.extend.event.listener.ApplicationEnvironmentPreparedEventListener,\
springboot.propertysource.extend.event.listener.ApplicationPreparedEventListener

# Environment Post Processors
org.springframework.boot.env.EnvironmentPostProcessor=\
springboot.propertysource.extend.processor.CustomEnvironmentPostProcessor
```

> 以上的扩展可以选取其中一种进行扩展，只是属性源的加载时机不太一样

6.2、扩展实例代码

https://github.com/shijw823/springboot-externalized-configuration-extend.git

`PropertySources` 顺序：

```
propertySourceName: [ApplicationPreparedEventListener], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [CustomSpringApplicationRunListener-contextLoaded], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [CustomSpringApplicationRunListener-contextPrepared],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [CustomApplicationContextInitializer], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [bootstrapProperties], propertySourceClassName: [CompositePropertySource]

propertySourceName: [configurationProperties], propertySourceClassName:
[ConfigurationPropertySourcesPropertySource]

propertySourceName: [CustomSpringApplicationRunListener-environmentPrepared],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [CustomEnvironmentPostProcessor-dev-application], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [ApplicationEnvironmentPreparedEventListener], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [commandLineArgs], propertySourceClassName:
[SimpleCommandLinePropertySource]

propertySourceName: [servletConfigInitParams], propertySourceClassName: [StubPropertySource]

propertySourceName: [servletContextInitParams], propertySourceClassName:
[ServletContextPropertySource]

propertySourceName: [systemProperties], propertySourceClassName: [MapPropertySource]

propertySourceName: [systemEnvironment], propertySourceClassName:
[OriginAwareSystemEnvironmentPropertySource]

propertySourceName: [random], propertySourceClassName: [RandomValuePropertySource]

propertySourceName: [applicationConfig:
[classpath:/extend/config/springApplicationRunListener.properties]], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [applicationConfig:
[classpath:/extend/config/applicationListener.properties]], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [applicationConfig:
[classpath:/extend/config/applicationContextInitializer.properties]], propertySourceClassName:
[OriginTrackedMapPropertySource]
```

```
propertySourceName: [applicationConfig:
[classpath:/extend/config/environmentPostProcessor.properties]], propertySourceClassName:
[OriginTrackedMapPropertySource]

propertySourceName: [applicationConfig: [classpath:/extend/config/application.properties]],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [applicationConfig: [classpath:/extend/config/config.properties]],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [applicationConfig: [classpath:/application.properties]],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [springCloudClientHostInfo], propertySourceClassName: [MapPropertySource]

propertySourceName: [applicationConfig: [classpath:/bootstrap.properties]],
propertySourceClassName: [OriginTrackedMapPropertySource]

propertySourceName: [propertySourceConfig], propertySourceClassName: [ResourcePropertySource]

propertySourceName: [defaultProperties], propertySourceClassName: [MapPropertySource]
```

> bootstrapProperties 是 获取远端（config-server）的 property sources
>
> 加载顺序也可参考 http://{host}:{port}/actuator/env

`PropertySources` 单元测试顺序：

```
@TestPropertySource#properties
@SpringBootTest#properties
@TestPropertySource#locations
```

三、参考资料

https://docs.spring.io/spring-boot/docs/2.0.5.RELEASE/reference/htmlsingle/#boot-features-external-config