- 在 Springboot 源码分析—— 总纲 中介绍了 `Springboot` 启动的总过程，本文将详细解析其中的 `prepareEnvironment()` 方法。
- `prepareEnvironment()` 方法主要是为了查找并设置配置文件信息到 `environment` 中。

1 prepareEnvironment()：查找并设置配置文件信息

- 根据应用类型，创建应用环境：如得到系统的参数、`JVM` 及 `Servlet` 等参数，等
- 将 `defaultProperties`、`commandLine` 及 `active-prifiles` 属性加载到环境中
- 将配置文件加载到环境中

```
//SpringApplication：
private ConfigurableEnvironment prepareEnvironment(SpringApplicationRunListeners
listeners, ApplicationArguments applicationArguments) {
    //根据应用类型，创建应用环境：如得到系统的参数、JVM及Servlet等参数，等
    ConfigurableEnvironment environment = this.getOrCreateEnvironment();
    //将 defaultProperties、commandLine及active-prifiles 属性加载到环境中
    //commandLine 在 args 中配置
    //其它参数可在如下4个路径中配置：servletConfigInitParams、servletContextInitParams、
systemProperties、systemEnvironment
    this.configureEnvironment((ConfigurableEnvironment)environment,
applicationArguments.getSourceArgs());
    //1.1 将 spirng.config 配置文件加载到环境中
    listeners.environmentPrepared((ConfigurableEnvironment)environment);
    this.bindToSpringApplication((ConfigurableEnvironment)environment);
    if (!this.isCustomEnvironment) {
        environment = (new
EnvironmentConverter(this.getClassLoader())).convertEnvironmentIfNecessary((Configurabl
eEnvironment)environment, this.deduceEnvironmentClass());
    }
    //加一个ConfigurationPropertySources
    ConfigurationPropertySources.attach((Environment)environment);
    return (ConfigurableEnvironment)environment;
}
```

1.1 environmentPrepared()：运行一些7个监听器，主要是ConfigFileApplicationListener

- 运行7监听器中的环境配置类，关键的是 `ConfigFileApplicationListener`，来读取配置文件，其它六个为：
- `AnsiOutputApplicationListener`：不知道干啥
- `LoggingApplicationListener`：不知道干啥
- `ClasspathLoggingApplicationListener`：打日志
- `BackgroundPreinitializer`：不知道干啥
- `DelegatingApplicationListener`：不知道干啥
- `FileEncodingApplicationListener`：判断编解码是否强制

```
//SpringApplicationRunListeners：
//这里之后，environment中的PropertySources中已经包含了所有的配置文件了
//这里的listeners就一个：EventPublishingRunListener
public void environmentPrepared(ConfigurableEnvironment environment) {
    Iterator var2 = this.listeners.iterator();

    while(var2.hasNext()) {
        SpringApplicationRunListener listener =
(SpringApplicationRunListener)var2.next();
        listener.environmentPrepared(environment);
    }
}
//EventPublishingRunListener：
public void environmentPrepared(ConfigurableEnvironment environment) {
    //initialMulticaster 为 SimpleApplicationEventMulticaster 实例
    //一般有7个监听事件：ConfigFileAppLst、AnsiOutputAppLst、LoggingAppLst、
ClasspathLoggingAppLst、BackgroundPreinitializer、DelegatingAppLst、FileEncodingAppLst
    //1.1.1 关键是通过调用
ConfigFileApplicationListener#onApplicationEnvironmentPreparedEvent() 方法来读取配置文件！
    this.initialMulticaster.multicastEvent(new
ApplicationEnvironmentPreparedEvent(this.application, this.args, environment));
}
```

1.1.1 ConfigFileAppLst#onApplicationEnvironmentPreparedEvent()：调用4个后置处理器

- `ConfigFileApplicationListener`：读取配置文件----------下文就是分析这个！
- `SystemEnvironmentPropertySourceEnvironmentPostProcessor`：重新读取设置系统参数，`Java_HOME` 等
- `SpringApplicationJsonEnvironmentPostProcessor`：解析系统环境中的 `Json` 配置
- `CloudFoundryVvcapEnvironmentPostProcessor`：不知道干啥

```
//ConfigFileApplicationListener：
private void onApplicationEnvironmentPreparedEvent(ApplicationEnvironmentPreparedEvent
event) {
```

```
    //查找spring.factories中EnvironmentPostProcessor的类，这里有3个
    //1 SystemEnvironmentPropertySourceEnvironmentPostProcessor：重新读取设置系统参数，
Java_HOME等
    //2 SpringApplicationJsonEnvironmentPostProcessor：解析系统环境中的Json配置
    //3 CloudFoundryVvcapEnvironmentPostProcessor：不知道干啥
    List<EnvironmentPostProcessor> postProcessors = this.loadPostProcessors();
    //加上ConfigFileApplicationListener：关键是这个！！！
    postProcessors.add(this);
    //按照AnnotationAwareOrderComparator类排序
    AnnotationAwareOrderComparator.sort(postProcessors);
    Iterator var3 = postProcessors.iterator();
    while(var3.hasNext()) {
        EnvironmentPostProcessor postProcessor = (EnvironmentPostProcessor)var3.next();
        //1.2 关键是调用 ConfigFileApplicationListener 类中的 postProcessEnvironment()：加
载配置文件
        postProcessor.postProcessEnvironment(event.getEnvironment(),
event.getSpringApplication());
    }
}
```

1.2 ConfigFileAppLst#postProcessEnvironment()：调用这个后置处理器来预加载配置文件

- 初始化文件加载器——包括 `placeholder、resourceLoader、propertyLoader`
- 通过文件加载器加载配置文件

```
//ConfigFileApplicationListener，
public void postProcessEnvironment(ConfigurableEnvironment environment,
SpringApplication application) {
    this.addPropertySources(environment, application.getResourceLoader());
}
//ConfigFileApplicationListener：
protected void addPropertySources(ConfigurableEnvironment environment, ResourceLoader
resourceLoader) {
    RandomValuePropertySource.addToEnvironment(environment);
    //2 通过load()方法加载配置文件----------------------
    (new ConfigFileApplicationListener.Loader(environment, resourceLoader)).load();
}
//新建文件加载器
Loader(ConfigurableEnvironment environment, ResourceLoader resourceLoader) {
    this.logger = ConfigFileApplicationListener.this.logger;
    this.loadDocumentsCache = new HashMap();
    this.environment = environment;
    //新建各种解析器和加载器，并将环境设置进去
    this.placeholdersResolver = new
PropertySourcesPlaceholdersResolver(this.environment);
    this.resourceLoader = (ResourceLoader)(resourceLoader != null ? resourceLoader :
new DefaultResourceLoader());
    //配置文件加载器，各个包都有自己的加载器来加载自己的文件，接口实现分离！
    //基本有两种，PropertiesPropertySourceLoader和YamlPropertySourceLoader！
```

```
    this.propertySourceLoaders =
SpringFactoriesLoader.loadFactories(PropertySourceLoader.class,
this.getClass().getClassLoader());
}
```

2 加载配置文件核心方法——load()

- 读取配置文件默认的路径：`classpath:/,classpath:/config/,file:./,file:./config/`
- 如果设置了 `spring.config.location`，则指定扫描的配置文件路径，`commaDelimitedListToStringArray`（用逗号隔开可以表示多个路径）
- 如果配置了 `spring.config.additional-location` 默认路径加上它
- 通过3次 `load(3参数)` 调用，基本上加载了所有配置文件了

```
public void load() {
    //2.1 初始化environment中配置的 active-profiles 和 spring.profiles.* 包含的所有profiles
    this.initializeProfiles();
    //遍历profiles，并解析：这里的profiles至少有两个呀，null和default或null或自定义的
    while(!this.profiles.isEmpty()) {
        Profile profile = this.profiles.poll();
        //如果profile是自定义的则加入
        if (profile != null && !profile.isDefaultProfile()) {
            //如果系统中定义了profiles属性，加入
            this.addProfileToEnvironment(profile.getName());
        }
        //2.2 第一次profile=null，第二次profile=default，开始解析
        this.load(profile, this::getPositiveProfileFilter,
this.addToLoaded(MutablePropertySources::addLast, false));
        //存已经处理过的 profile
        this.processedProfiles.add(profile);
    }
    //将系统变量中自定义的profiles重新加入到环境中
    this.resetEnvironmentProfiles(this.processedProfiles);
    // 第3次 profile=null
    this.load((ConfigFileApplicationListener.Profile)null,
this::getNegativeProfileFilter, this.addToLoaded(MutablePropertySources::addFirst,
true));
    //加入PropertySources，到这里，environment中就多了项目中的配置文件了
    this.addLoadedPropertySources();
}
```

2.1 initializeProfiles()：初始化所有 profiles

- 查找系统变量中的 `spring.profiles.active` 和 `spring.profiles.include` 属性对应的 `Profile`
- 如果找到了则返回，如果没找到，则查找系统变量中的 `spring.profiles.default` 属性
- 如果没找到的话 `profiles.add('default')`

```
private void initializeProfiles() {
```

```
    // The default profile for these purposes is represented as null. We add it
    // first so that it is processed first and has lowest priority.
    this.profiles.add(null);
    //找到PropertySource中的spring.profiles.active和spring.profiles.include属性对应的
Profile
    Set<Profile> activatedViaProperty = getProfilesActivatedViaProperty();
    //这一步不知道它的意图，感觉addAll的一定是null啊
    this.profiles.addAll(getOtherActiveProfiles(activatedViaProperty));
    addActiveProfiles(activatedViaProperty);
    if (this.profiles.size() == 1) { // only has null profile
        //AbstractEnvironment 中有默认的default属性
        for (String defaultProfileName : this.environment.getDefaultProfiles()) {
            Profile defaultProfile = new Profile(defaultProfileName, true);
            //如果到这的话，且系统属性没有spring.profiles.default的话，defaultProfile=
'default'
            //如果系统属性有spring.profiles.default的话，设置到环境中的
this.setDefaultProfiles
            this.profiles.add(defaultProfile);
        }
    }
}
```

2.2 load(3个参数)：加载配置文件

- 在 `load()` 方法中总共调用了此方法3次，参数分别是：
- `profile:null,default,null`
- `filterFactory:2*getPositiveProfileFilter，getNegativeProfileFilter`
- `consumer：2*addToLoaded(MutablePropertySources::addFirst, true)，addToLoaded(MutablePropertySources::addLast, false)`

```
private void load(Profile profile, DocumentFilterFactory filterFactory,
        DocumentConsumer consumer) {
    getSearchLocations().forEach((location) -> {
        boolean isFolder = location.endsWith("/");
        Set<String> names = isFolder ? getSearchNames() : NO_SEARCH_NAMES;
        //正式加载配置配置文件：后面就有点乱来，到这差不多行啦...
        //可以简单参考：https://github.com/wangkang09/wk-
notes/blob/master/spring/springboot/4.2.1.1.4.2%20加载配置文件.md
        names.forEach(
                (name) -> load(location, name, profile, filterFactory, consumer));
    });
}
```

2.2.1 getSearchLocations()：获取在哪个路径中查找配置文件

- 如果设置了 `spring.config.location`，则指定扫描的配置文件路径，`commaDelimitedListToStringArray`（用逗号隔开可以表示多个路径）
- 如果配置了 `spring.config.additional-location`，则添加此路径，在加上默认路径
- 默认路径为：`classpath:/,classpath:/config/,file:./,file:./config/`

```
private Set<String> getSearchLocations() {
    if (this.environment.containsProperty("spring.config.location")) {
        return this.getSearchLocations("spring.config.location");
    } else {
        Set<String> locations = this.getSearchLocations("spring.config.additional-
location");

 locations.addAll(this.asResolvedSet(ConfigFileApplicationListener.this.searchLocations
, "classpath:/,classpath:/config/,file:./,file:./config/"));
        return locations;
    }
}
```

2.2.2 load(5个参数)：正式加载,后面就有点乱来，到这差不多行啦