17 Spring IoC容器如何读取多个属性文件或者配置文件？

更新时间：2020-06-22 10:10:35

> 什么是路？就是从没路的地方践踏出来的，从只有荆棘的地方开辟出来的。 —— 鲁迅

背景

在 Spring 项目中，你可能需要从 properties 文件中读入配置注入到 bean 中，例如数据库连接信息，redis server 的地址端口信息等，此时，Properties 配置文件不止一个，需要在系统启动时同时加载多个 Properties 文件，该如何做呢？

Spring 加载多个属性文件

- Spring 老版本 XML 配置方式：

在 applicationContext.xml 使用 context:property-placeholder/，注意，需要增加命名空间：xmlns:context="http://www.springframework.org/schema/context"

示例：

```
<context:property-placeholder location="classpath:foo.properties, classpath:bar.properties"/>
```

或者在 applicationContext.xml 使用util:properties，注意，需要用到命名空间：xmlns:util="http://www.springframework.org/schema/util" 和 schema：http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd

示例：

```
<util:properties id="fileA" location="classpath:META-INF/properties/a.properties"/>
<util:properties id="fileB" location="classpath:META-INF/properties/b.properties"/>
```

然后使用方式如下：

```
@Resource(name="fileA")
private Properties propertyA;

@Resource(name="fileB")
private Properties propertyB;
```

- spring4.x 以上版本，使用注解或者配置。

注解方式，spring boot 同样适用：

```
@PropertySources( {
 @PropertySource("classpath:config001.properties"),
 @PropertySource("classpath:config002.properties")
} )
```

也可采用配置方式：

```
@Bean
public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
 PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer
         = new PropertySourcesPlaceholderConfigurer();
 Resource[] resources = new ClassPathResource[]{
   new ClassPathResource("app.properties")
 };
 propertySourcesPlaceholderConfigurer.setLocations(resources);
 propertySourcesPlaceholderConfigurer.setIgnoreUnresolvablePlaceholders(true);
 return propertySourcesPlaceholderConfigurer;
}
```

注意：

1. 使用 XML 文件属性 `<property-placeholder>` 定义属性文件时，如果用 **@Value** 注解时

若属性文件定义在父上下文时：

@Value 不能在子上下文中获取到值；

@Value 可以从父上下文获取到值。

若属性文件定义在子上下文：

@Value 可以在子上下文中获取到值；

@Value 不能从父上下文获取到值。

不能使用 environment.getProperty 从上下文获取到值，原因：`<property-placeholder>` 不对 Enviroment 暴漏属性。

2. 使用 @PropertySource 注解定义属性文件时，

若属性文件定义在父上下文时:

@Value 可以从子上下文获取到值；

@Value 可以从父上下文获取到值；

可以使用 environment.getProperty 在子上下文中获取到属性值；

可以使用 environment.getProperty 在父上下文中获取到属性值。

若属性文件定义在子上下文：

@Value 可以从子上下文获取到值；

@Value 不能从父上下文获取到值；

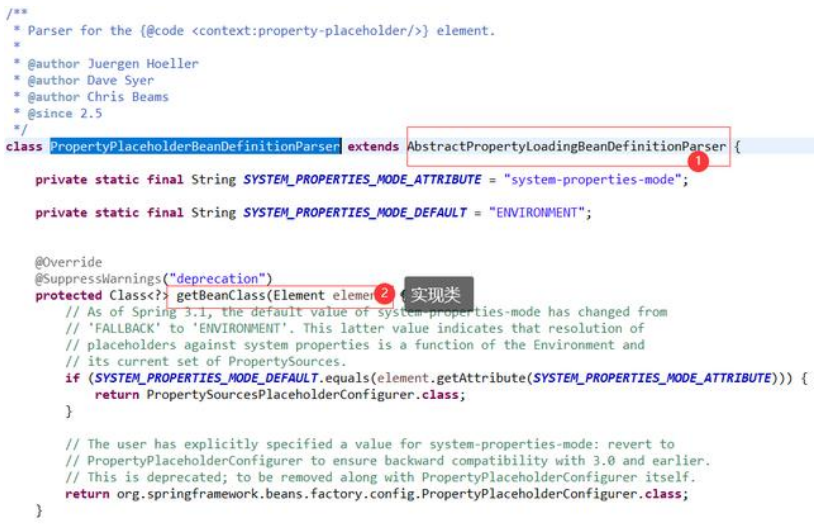
可以使用 environment.getProperty 从子上下文获取到值；

不能使用 environment.getProperty 从父上下文获取到值。

Spring 读取多个属性文件的原理

- <context:property-placeholder/>深入原理。

<context:property-placeholder/> 由 PropertyPlaceholderBeanDefinitionParser 类来解读：



解析 location 在 doParse 方法：

```java
@Override
protected void doParse(Element element, ParserContext parserContext, BeanDefinitionBuilder builder) {
    String location = element.getAttribute("location");
    if (StringUtils.hasLength(location)) {
        location = parserContext.getReaderContext().getEnvironment().resolvePlaceholders(location);
        String[] locations = StringUtils.commaDelimitedListToStringArray(location);
        builder.addPropertyValue("locations", locations);
    }

    String propertiesRef = element.getAttribute("properties-ref");
    if (StringUtils.hasLength(propertiesRef)) {
        builder.addPropertyReference("properties", propertiesRef);
    }

    String fileEncoding = element.getAttribute("file-encoding");
    if (StringUtils.hasLength(fileEncoding)) {
        builder.addPropertyValue("fileEncoding", fileEncoding);
    }

    String order = element.getAttribute("order");
    if (StringUtils.hasLength(order)) {
        builder.addPropertyValue("order", Integer.valueOf(order));
    }

    builder.addPropertyValue("ignoreResourceNotFound",
            Boolean.valueOf(element.getAttribute("ignore-resource-not-found")));

    builder.addPropertyValue("localOverride",
            Boolean.valueOf(element.getAttribute("local-override")));

    builder.setRole(BeanDefinition.ROLE_INFRASTRUCTURE);
}
```

location 解读

加载多个属性文件使用 PropertyResourceConfigurer#postProcessBeanFactory ：

```java
/**
 * {@linkplain #mergeProperties Merge}, {@linkplain #convertProperties convert} and
 * {@linkplain #processProperties process} properties against the given bean factory.
 * @throws BeanInitializationException if any properties cannot be loaded
 */
@Override
public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException {
    try {
        Properties mergedProps = mergeProperties();

        // Convert the merged properties, if necessary.
        convertProperties(mergedProps);

        // Let the subclass process the properties.
        processProperties(beanFactory, mergedProps);
    }
    catch (IOException ex) {
        throw new BeanInitializationException("Could not load properties", ex);
    }
}
```

最后调用 PropertiesLoaderUtils.fillProperties() 方法，支持 XML 和属性文件：

```java
/**
 * Actually load properties from the given EncodedResource into the given Properties instance.
 * @param props the Properties instance to load into
 * @param resource the resource to load from
 * @param persister the PropertiesPersister to use
 * @throws IOException in case of I/O errors
 */
static void fillProperties(Properties props, EncodedResource resource, PropertiesPersister persister)
        throws IOException {

    InputStream stream = null;
    Reader reader = null;
    try {
        String filename = resource.getResource().getFilename();
        if (filename != null && filename.endsWith(XML_FILE_EXTENSION)) {
            stream = resource.getInputStream();
            persister.loadFromXml(props, stream);
        }
        else if (resource.requiresReader()) {
            reader = resource.getReader();
            persister.load(props, reader);
        }
        else {
            stream = resource.getInputStream();
            persister.load(props, stream);
        }
    }
    finally {
        if (stream != null) {
            stream.close();
        }
        if (reader != null) {
            reader.close();
        }
    }
}
```

- <util:properties> 深入原理。

UtilNamespaceHandler 解析 <util> 标签，包含多个子标签：

```java
/**
 * {@link NamespaceHandler} for the {@code util} namespace.
 *
 * @author Rob Harrop
 * @author Juergen Hoeller
 * @since 2.0
 */
public class UtilNamespaceHandler extends NamespaceHandlerSupport {

    private static final String SCOPE_ATTRIBUTE = "scope";


    @Override
    public void init() {
        registerBeanDefinitionParser("constant", new ConstantBeanDefinitionParser());
        registerBeanDefinitionParser("property-path", new PropertyPathBeanDefinitionParser());
        registerBeanDefinitionParser("list", new ListBeanDefinitionParser());
        registerBeanDefinitionParser("set", new SetBeanDefinitionParser());
        registerBeanDefinitionParser("map", new MapBeanDefinitionParser());
        registerBeanDefinitionParser("properties", new PropertiesBeanDefinitionParser());
    }
```

<util：properties> 的解析则调用 UtilNamespaceHandler 解析，使用 PropertiesFactoryBean 来将 classpath 下的属性文件加载到内存中，多个文件都加载时会进行 merge 处理。

```java
private static class PropertiesBeanDefinitionParser extends AbstractSingleBeanDefinitionParser {

    @Override
    protected Class<?> getBeanClass(Element element) {
        return PropertiesFactoryBean.class;
    }

    @Override
    protected void doParse(Element element, ParserContext parserContext, BeanDefinitionBuilder builder) {
        Properties parsedProps = parserContext.getDelegate().parsePropsElement(element);
        builder.addPropertyValue("properties", parsedProps);

        String location = element.getAttribute("location");
        if (StringUtils.hasLength(location)) {
            location = parserContext.getReaderContext().getEnvironment().resolvePlaceholders(location);
            String[] locations = StringUtils.commaDelimitedListToStringArray(location);
            builder.addPropertyValue("locations", locations);
        }

        builder.addPropertyValue("ignoreResourceNotFound",
                Boolean.valueOf(element.getAttribute("ignore-resource-not-found")));

        builder.addPropertyValue("localOverride",
                Boolean.valueOf(element.getAttribute("local-override")));

        String scope = element.getAttribute(SCOPE_ATTRIBUTE);
        if (StringUtils.hasLength(scope)) {
            builder.setScope(scope);
        }
    }
}
```

最后 PropertiesFactoryBean 调用 PropertiesLoaderUtils.fillProperties() 方法，支持 XML 和属性文件.

- PropertySource 注解深入原理。

  从 spring4.3 开始，使用 PropertySourceFactory 的工厂方法来加载属性文件：

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Repeatable(PropertySources.class)
public @interface PropertySource {

    /**
     * Indicate the name of this property source. If omitted, the {@link #factory()}
     * will generate a name based on the underlying resource (in the case of
     * {@link org.springframework.core.io.support.DefaultPropertySourceFactory}:
     * derived from the resource description through a corresponding name-less
     * {@link org.springframework.core.io.support.ResourcePropertySource} constructor).
     * @see org.springframework.core.env.PropertySource#getName()
     * @see org.springframework.core.io.Resource#getDescription()
     */
    String name() default "";

    /**
     * Indicate the resource location(s) of the properties file to be loaded.
     * <p>Both traditional and XML-based properties file formats are supported
     * &mdash; for example, {@code "classpath:/com/myco/app.properties"}
     * or {@code "file:/path/to/file.xml"}.
     * <p>Resource location wildcards (e.g. *&#42;/*.properties) are not permitted;
     * each location must evaluate to exactly one {@code .properties} resource.
     * <p>${...} placeholders will be resolved against any/all property sources already
     * registered with the {@code Environment}. See {@linkplain PropertySource above}
     * for examples.
     * <p>Each location will be added to the enclosing {@code Environment} as its own
     * property source, and in the order declared.
     */
    String[] value();

    /**
     * Indicate if failure to find the a {@link #value() property resource} should be
     * ignored.
     * <p>{@code true} is appropriate if the properties file is completely optional.
     * Default is {@code false}.
     * @since 4.0
     */
    boolean ignoreResourceNotFound() default false;

    /**
     * A specific character encoding for the given resources, e.g. "UTF-8".
     * @since 4.3
     */
    String encoding() default "";

    /**
     * Specify a custom {@link PropertySourceFactory}, if any.
     * <p>By default, a default factory for standard resource files will be used.
     * @since 4.3
     * @see org.springframework.core.io.support.DefaultPropertySourceFactory
     * @see org.springframework.core.io.support.ResourcePropertySource
     */
    Class<? extends PropertySourceFactory> factory() default PropertySourceFactory.class;

}
```

PropertySourceFactory 的默认实现 DefaultPropertySourceFactory 使用 ResourcePropertySource 将属性文件资源化。

```java
/**
The default implementation for {@link PropertySourceFactory},
wrapping every resource in a {@link ResourcePropertySource}.
@author Juergen Hoeller
@since 4.3
@see PropertySourceFactory
@see ResourcePropertySource  */
public class DefaultPropertySourceFactory implements PropertySourceFactory {
   @Override
   public PropertySource<?> createPropertySource(@Nullable String name, EncodedResource resource) throws IOException {
   return (name != null ? new ResourcePropertySource(name, resource) :
         new ResourcePropertySource(resource));
 }
}
```

最后调用 PropertiesLoaderUtils.fillProperties() 方法 来加载资源。

总结

通过阅读代码发现，不管何种形式支持多个属性文件的读取，实现的代码是一致的，都是调用 PropertiesLoaderUtils.fillProperties() 方法来加载资源，底层的实现都是 Properties 的 load() 方法。

扩展：Spring MVC 中如何加载多个 XML 文件

方式一：使用**contextConfigLocation**。

```
<servlet-name>springMVC</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                WEB-INF/spring-core-context.xml,
                WEB-INF/spring-DAO.xml,
                WEB-INF/spring-Services.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

方式二：使用 import 注解。

```
<beans>
    <import resource="spring-core-context.xml"/>
    <import resource="spring-DAO.xml"/>
    <import resource="spring-services.xml"/>

    ... //other configurations

</beans>
```

参考 ResourceUtils，具体大家可以自己尝试分析。

}

16 Resource的前生今世

18 Spring IoC容器如何读取应用外部的xml，txt，图形或者属性文件？