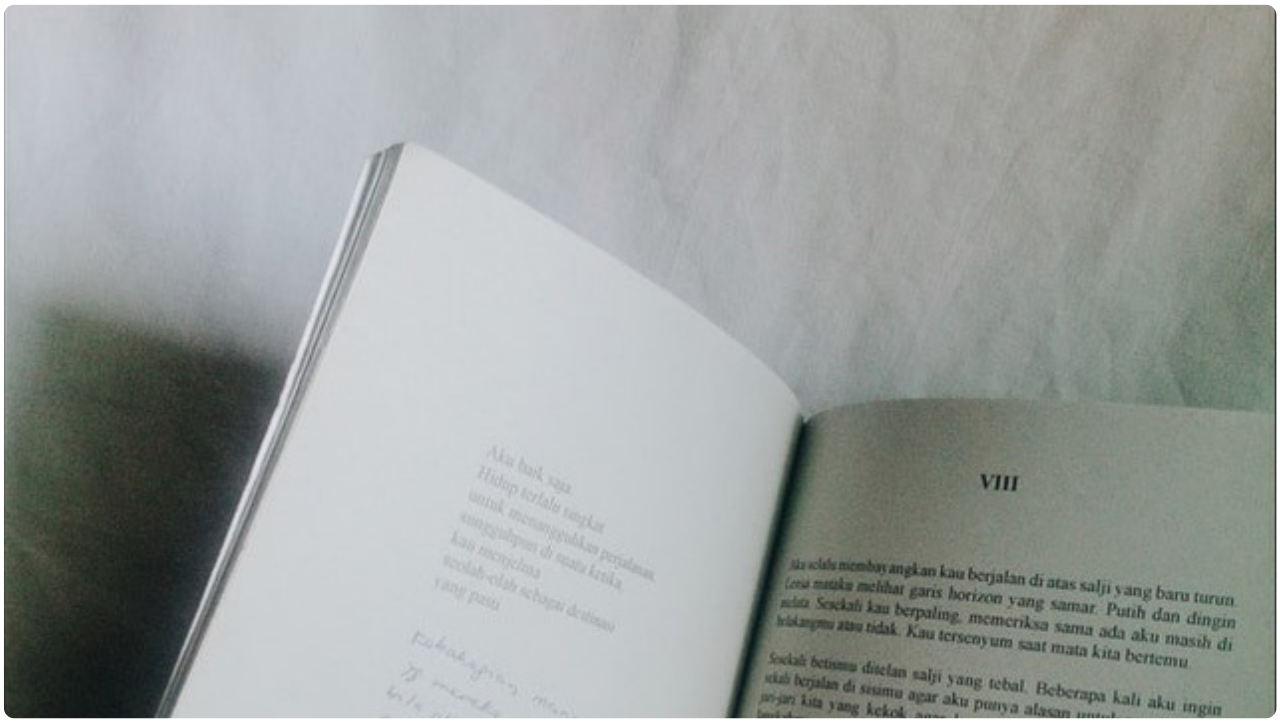
18 Spring IoC容器如何读取应用外部的xml，txt，图形或者属性文件？

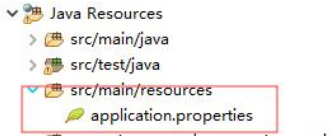
更新时间：2020-08-04 17:51:21



> 不要问你的国家能够为你做些什么，而要问你可以为国家做些什么。——林肯

背景

我们使用 Spring IoC 容器的时候，配置文件一般是放到到应用内部，如下所示：



我们往往使用 ClassPathXmlApplicationContext 来根据相对的位置来读取这些内部文件。如果我们想要读取外部的 XML，txt 等文件，或者甚至是别的网站上的文件，该如何处理呢？

Spring 读取外部文件实例

Spring 同时也提供了一个 FileSystemXmlApplicationContext 可以读取外部的文件，甚至 URL 路径的文件，我们来看一个示例：

javaBean

```java
package com.davidwang456.test;
public class Employee {
    private int age;
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

外部配置文件

E:\tmp\SpringBeans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="bean1" class="com.davidwang456.test.Employee">
    <property name="name" value="Rakesh" />
    <property name="age" value="20" />
  </bean>
</beans>
```

测试类

```java
package com.davidwang456.test;

import org.springframework.context.ApplicationContext;
//import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class ApplicationContextTest {
    public static void main(String[] args) {
        @SuppressWarnings("resource")
        //ApplicationContext context = new ClassPathXmlApplicationContext("SpringBeans.xml");
        ApplicationContext context = new FileSystemXmlApplicationContext
            ("E:/tmp/SpringBeans.xml");
        Employee emp = (Employee) context.getBean("bean1");
        System.out.println(emp.getAge());
    }
}
```

容器 ApplicationContext 启动时从应用外部的磁盘空间读取 SpringBeans.xml 文件，然后获取定义的 Bean，最后输出 Bean 的方法。

Spring 读取外部文件工作原理

查找关键点，打印出调用链

根据 ApplicationContext 的继承关系，我们知道 AbstractApplicationContext 继承了 DefaultResourceLoader，并实现了 getResources()，因此我们在 AbstractApplicationContext 的 getResources() 打印出调用调用链。

```java
@Override
public Resource[] getResources(String locationPattern) throws IOException {
  PrintStackUtil.printStack();
  return this.resourcePatternResolver.getResources(locationPattern);
}
```

打印出调用链如下：

```
打印序列号 1 调用类和方法：
com.davidwang456.test.ApplicationContextTest$main
打印序列号 2 调用类和方法：
org.springframework.context.support.FileSystemXmlApplicationContext$init
打印序列号 3 调用类和方法：
org.springframework.context.support.FileSystemXmlApplicationContext$init
打印序列号 4 调用类和方法：
org.springframework.context.support.AbstractApplicationContext$refresh
打印序列号 5 调用类和方法：
org.springframework.context.support.AbstractApplicationContext$obtainFreshBeanFactory
打印序列号 6 调用类和方法：
org.springframework.context.support.AbstractRefreshableApplicationContext$refreshBeanFactor
打印序列号 7 调用类和方法：
org.springframework.context.support.AbstractXmlApplicationContext$loadBeanDefinitions
打印序列号 8 调用类和方法：
org.springframework.context.support.AbstractXmlApplicationContext$loadBeanDefinitions
打印序列号 9 调用类和方法：
org.springframework.beans.factory.support.AbstractBeanDefinitionReader$loadBeanDefinitions
打印序列号 10 调用类和方法：
org.springframework.beans.factory.support.AbstractBeanDefinitionReader$loadBeanDefinitions
打印序列号 11 调用类和方法：
org.springframework.beans.factory.support.AbstractBeanDefinitionReader$loadBeanDefinitions
打印序列号 12 调用类和方法：
org.springframework.context.support.AbstractApplicationContext$getResources
打印序列号 13 调用类和方法：
com.davidwang456.test.util.PrintStackUtil$printStack
打印序列号 14 调用类和方法：
java.lang.Thread$getAllStackTraces
打印序列号 15 调用类和方法：
java.lang.Thread$dumpThreads
```

分析链路源码，并深入原理

1. 读取 XML 文件 AbstractBeanDefinitionReader.java：

```
public int loadBeanDefinitions(String location, @Nullable Set<Resource> actualResources) throws BeanDefinitionStoreException {
    ResourceLoader resourceLoader = getResourceLoader();
    if (resourceLoader == null) {
        throw new BeanDefinitionStoreException(
                "Cannot load bean definitions from location [" + location + "]: no ResourceLoader available");
    }

    if (resourceLoader instanceof ResourcePatternResolver) {
        // Resource pattern matching available.
        try {
            Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(location);
            int count = loadBeanDefinitions(resources);
            if (actualResources != null) {
                Collections.addAll(actualResources, resources);
            }
            if (logger.isTraceEnabled()) {
                logger.trace("Loaded " + count + " bean definitions from location pattern [" + location + "]");
            }
            return count;
        }
        catch (IOException ex) {
            throw new BeanDefinitionStoreException(
                    "Could not resolve bean definition resource pattern [" + location + "]", ex);
        }
    }
    else {
        // Can only load single resources by absolute URL.
        Resource resource = resourceLoader.getResource(location);
        int count = loadBeanDefinitions(resource);
        if (actualResources != null) {
            actualResources.add(resource);
        }
        if (logger.isTraceEnabled()) {
            logger.trace("Loaded " + count + " bean definitions from location [" + location + "]");
        }
        return count;
    }
}
```

其中 resourceLoader 即 FileSystemXmlApplicationContext。

调用 PathMatchingResourcePatternResolver 的 getResources 方法：

```
@Override
public Resource[] getResources(String locationPattern) throws IOException {
    Assert.notNull(locationPattern, "Location pattern must not be null");
    if (locationPattern.startsWith(CLASSPATH_ALL_URL_PREFIX)) {
        // a class path resource (multiple resources for same name possible)
        if (getPathMatcher().isPattern(locationPattern.substring(CLASSPATH_ALL_URL_PREFIX.length()))) {
            // a class path resource pattern
            return findPathMatchingResources(locationPattern);
        }
        else {
            // all class path resources with the given name
            return findAllClassPathResources(locationPattern.substring(CLASSPATH_ALL_URL_PREFIX.length()));
        }
    }
    else {
        // Generally only look for a pattern after a prefix here,
        // and on Tomcat only after the "*/" separator for its "war:" protocol.
        int prefixEnd = (locationPattern.startsWith("war:") ? locationPattern.indexOf("*/") + 1 :
                locationPattern.indexOf(':') + 1);
        if (getPathMatcher().isPattern(locationPattern.substring(prefixEnd))) {
            // a file pattern
            return findPathMatchingResources(locationPattern);
        }
        else {
            // a single resource with the given name
            return new Resource[] {getResourceLoader().getResource(locationPattern)};
        }
    }
}
```

调用 DefaultResourceLoader 的 getResource 方法：

```
@Override
public Resource getResource(String location) {
    Assert.notNull(location, "Location must not be null");

    for (ProtocolResolver protocolResolver : getProtocolResolvers()) {
        Resource resource = protocolResolver.resolve(location, this);
        if (resource != null) {
            return resource;
        }
    }

    if (location.startsWith("/")) {
        return getResourceByPath(location);
    }
    else if (location.startsWith(CLASSPATH_URL_PREFIX)) {
        return new ClassPathResource(location.substring(CLASSPATH_URL_PREFIX.length()), getClassLoader());
    }
    else {
        try {
            // Try to parse the location as a URL...
            URL url = new URL(location);
            return (ResourceUtils.isFileURL(url) ? new FileUrlResource(url) : new UrlResource(url));
        }
        catch (MalformedURLException ex) {
            // No URL -> resolve as resource path.
            return getResourceByPath(location);
        }
    }
}
```

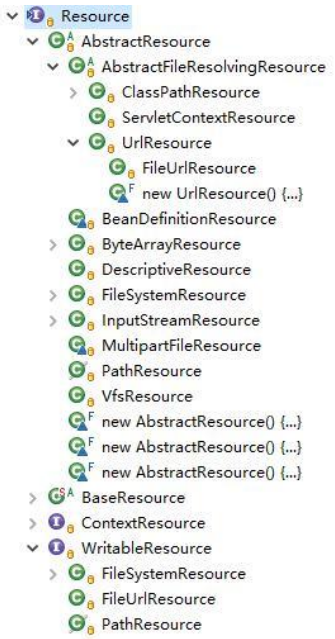
再调用 FileSystemXmlApplicationContext 的 getResourceByPath：

```
/**
 * Resolve resource paths as file system paths.
 * <p>Note: Even if a given path starts with a slash, it will get
 * interpreted as relative to the current VM working directory.
 * This is consistent with the semantics in a Servlet container.
 * @param path path to the resource
 * @return the Resource handle
 * @see org.springframework.web.context.support.XmlWebApplicationContext#getResourceByPath
 */
@Override
protected Resource getResourceByPath(String path) {
    if (path.startsWith("/")) {
        path = path.substring(1);
    }
    return new FileSystemResource(path);
}
```

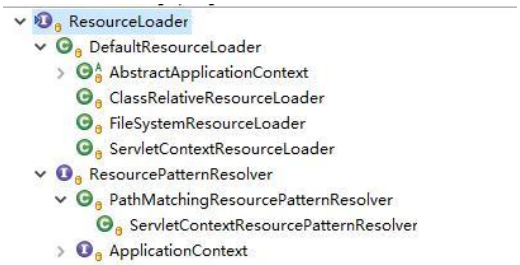
最终将配置文件转换为 FileSystemResource。

总结

在 Spring 中，一切皆资源，Spring 封装了各种 Resource 的实现：



ResourcePatternResolver 实现了 ResourceLoader 解析各种资源，其实现类有：



比如 PathMatchingResourcePatternResolver 支持五种格式的资源：

```
@Override
public Resource getResource(String location) {
    Assert.notNull(location, "Location must not be null");
    for (ProtocolResolver protocolResolver : getProtocolResolvers()) {
        Resource resource = protocolResolver.resolve(location, this);
        if (resource != null) {
            return resource;
        }
    }
    if (location.startsWith("/")) {
        return getResourceByPath(location);
    }
    else if (location.startsWith(CLASSPATH_URL_PREFIX)) {
        return new ClassPathResource(location.substring(CLASSPATH_URL_PREFIX.length()), getClassLoader());
    }
    else {
        try {
            // Try to parse the location as a URL...
            URL url = new URL(location);
            return (ResourceUtils.isFileURL(url) ? new FileUrlResource(url) : new UrlResource(url));
        }
        catch (MalformedURLException ex) {
            // No URL -> resolve as resource path.
            return getResourceByPath(location);
        }
    }
}
```

Bean 定义的文件，不管内部的还是外部的 XML，txt，图形或者属性文件等等，都可以使用 ResourcePatternResolver 来转化为 Resource，利用 BeanDefinitionReader 来读取。

}

17 Spring IoC容器如何读取多个属性文件或者配置文件？

19 Spring MVC应用拓展Resource应用于文件下载