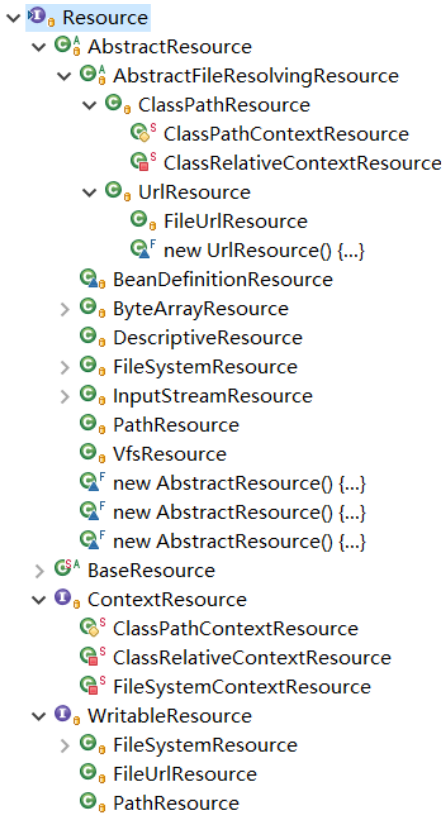
16 Resource的前生今世

更新时间：2020-08-06 13:39:21

> 自信和希望是青年的特权。——大仲马

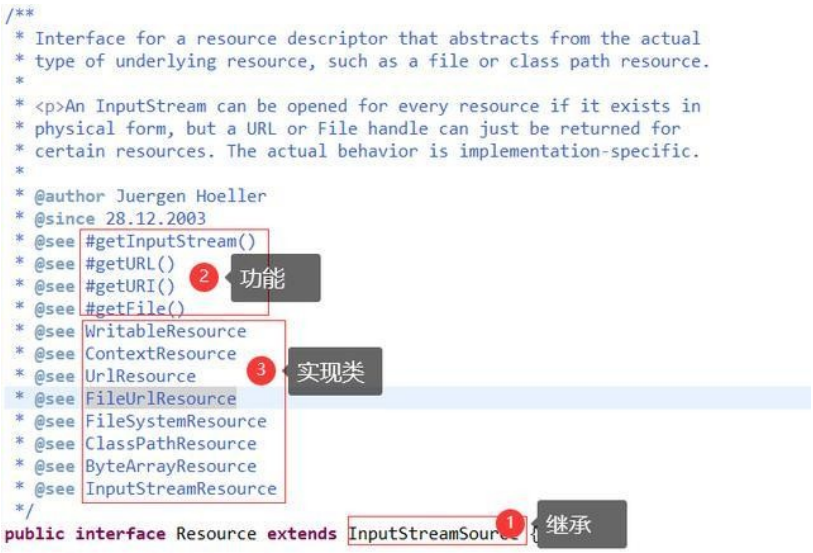
背景

Spring 把其资源做了一个抽象，底层使用统一的资源访问接口来访问 Spring 的所有资源。也就是说，不管什么格式的文件，也不管文件在哪里，到 Spring 底层，都只有一个访问接口，Resource。整个 Spring 的资源定义在 spring-core 的包中。spring core 定义了资源的访问方式，它使用一个抽象接口 Resource 封装了各种可能的资源类型，也就是对使用者来说屏蔽了文件类型的不同。这种把所有资源都抽象成一个接口的方式很值得在以后的设计中拿来学习。



Resource 概述

Spring 定义了一个 org.springframework.core.io.Resource 接口，Resource 接口是为了统一各种类型不同的资源而定义的，Spring 提供了若干 Resource 接口的实现类，这些实现类可以轻松地加载不同类型的底层资源，并提供了获取文件名、URL 地址以及资源内容的操作方法。



继承了接口 InputStreamSource

```java
/**
 * Simple interface for objects that are sources for an {@link InputStream}.
 *
 * <p>This is the base interface for Spring's more extensive {@link Resource} interface.
 *
 * <p>For single-use streams, {@link InputStreamResource} can be used for any
 * given {@code InputStream}. Spring's {@link ByteArrayResource} or any
 * file-based {@code Resource} implementation can be used as a concrete
 * instance, allowing one to read the underlying content stream multiple times.
 * This makes this interface useful as an abstract content source for mail
 * attachments, for example.
 *
 * @author Juergen Hoeller
 * @since 20.01.2004
 * @see java.io.InputStream
 * @see Resource
 * @see InputStreamResource
 * @see ByteArrayResource
 */
public interface InputStreamSource {

    /**
     * Return an {@link InputStream} for the content of an underlying resource.
     * <p>It is expected that each call creates a <i>fresh</i> stream.
     * <p>This requirement is particularly important when you consider an API such
     * as JavaMail, which needs to be able to read the stream multiple times when
     * creating mail attachments. For such a use case, it is <i>required</i>
     * that each {@code getInputStream()} call returns a fresh stream.
     * @return the input stream for the underlying resource (must not be {@code null})
     * @throws java.io.FileNotFoundException if the underlying resource doesn't exist
     * @throws IOException if the content stream could not be opened
     */
    InputStream getInputStream() throws IOException;

}
```

假设有一个文件地位于 Web 应用的类路径下，您可以通过以下方式对这个文件资源进行访问：

通过 FileSystemResource 以文件系统绝对路径的方式进行访问；

通过 ClassPathResource 以类路径的方式进行访问；

通过 ServletContextResource 以相对于 Web 应用根目录的方式进行访问。

相比于通过 JDK 的 File 类访问文件资源的方式，Spring 的 Resource 实现类无疑提供了更加灵活的操作方式，您可以根据情况选择适合的 Resource 实现类访问资源。

WritableResource：相较于 Resource 的读功能 getInputStream()，WritableResource 增加了写功能 getOutputStream()；

ContextResource：增加了上下文功能，例如可以从 ServletContextResource 读取一个资源；

UrlResource：支持 URL 和 file 方式的资源访问；

FileUrlResource：UrlResource 的实现类，侧重于 file 访问的方式；

FileSystemResource：实现了 WritableResource，同时支持 url 和 file 方式的资源访问；

ClassPathResource：以 classpath 为基准路径的资源访问方式；

ByteArrayResource：字节数组资源访问方式；

InputStreamResource：InputStreamSource 的加强版。注意两者的拼写不同。

Resource 应用

下面，我们分别通过 FileSystemResource 和 ClassPathResource 访问同一个文件资源：

```
package com.davidwang456.test;

import java.io.IOException;
import java.io.InputStream;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;

public class FileSourceExample {
    public static void main(String[] args) {
        try {
        String filePath = "D:/tmp/test/webapp/WEB-INF/classes/conf/file1.txt"; //使用系统文件路径方式加载文件
        Resource res1 = new FileSystemResource(filePath); //使用类路径方式加载文件
        Resource res2 = new ClassPathResource("conf/file1.txt");
        InputStream ins1 = res1.getInputStream();
        InputStream ins2 = res2.getInputStream();
        System.out.println("res1:"+res1.getFilename());
        System.out.println("res2:"+res2.getFilename());
        } catch (IOException e) {
            e.printStackTrace();
            }
        }
    }
```

在获取资源后，就可以通过 Resource 接口定义的多个方法访问文件的数据和其它的信息：比如可以通过 getFileName() 获取文件名，通过 getFile() 获取资源对应的 File 对象，通过 getInputStream() 直接获取文件的输入流。此外，还可以通过 createRelative(String relativePath) 在资源相对地址上创建新的资源。

在 Web 应用中，您还可以通过 ServletContextResource 以相对于 Web 应用根目录的方式访问文件资源，如下所示：

```
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
```

对于位于远程服务器（Web 服务器或 FTP 服务器）的文件资源，您则可以方便地通过 UrlResource 进行访问。

为了方便访问不同类型的资源，您必须使用相应的 Resource 实现类，是否可以在不显式使用 Resource 实现类的情况下，仅根据带特殊前缀的资源地址直接加载文件资源呢？ Spring 提供了一个 ResourceUtils 工具类，它支持 “classpath:” 和 “file:” 的地址前缀，它能够从指定的地址加载文件资源。

```java
/**
 * Utility methods for resolving resource locations to files in the
 * file system. Mainly for internal use within the framework.
 *
 * <p>Consider using Spring's Resource abstraction in the core package
 * for handling all kinds of file resources in a uniform manner.
 * {@link org.springframework.core.io.ResourceLoader}'s {@code getResource()}
 * method can resolve any location to a {@link org.springframework.core.io.Resource}
 * object, which in turn allows one to obtain a {@code java.io.File} in the
 * file system through its {@code getFile()} method.
 *
 * @author Juergen Hoeller
 * @since 1.1.5
 * @see org.springframework.core.io.Resource
 * @see org.springframework.core.io.ClassPathResource
 * @see org.springframework.core.io.FileSystemResource
 * @see org.springframework.core.io.UrlResource
 * @see org.springframework.core.io.ResourceLoader
 */
public abstract class ResourceUtils {

    /** Pseudo URL prefix for loading from the class path: "classpath:". */
    public static final String CLASSPATH_URL_PREFIX = "classpath:";

    /** URL prefix for loading from the file system: "file:". */
    public static final String FILE_URL_PREFIX = "file:";

    /** URL prefix for loading from a jar file: "jar:". */
    public static final String JAR_URL_PREFIX = "jar:";

    /** URL prefix for loading from a war file on Tomcat: "war:". */
    public static final String WAR_URL_PREFIX = "war:";

    /** URL protocol for a file in the file system: "file". */
    public static final String URL_PROTOCOL_FILE = "file";

    /** URL protocol for an entry from a jar file: "jar". */
    public static final String URL_PROTOCOL_JAR = "jar";

    /** URL protocol for an entry from a war file: "war". */
    public static final String URL_PROTOCOL_WAR = "war";

    /** URL protocol for an entry from a zip file: "zip". */
```

请看下面例子：

ResourceUtilsExample

```java
package com.davidwang456.test;


import java.io.File;
import org.springframework.util.ResourceUtils;
public class ResourceUtilsExample {
    public static void main(String[] args) throws Throwable{
        File clsFile = ResourceUtils.getFile("classpath:conf/file1.txt");
        System.out.println(clsFile.isFile());
        String httpFilePath = "file:D:/masterSpring/chapter23/src/conf/file1.txt";
        File httpFile = ResourceUtils.getFile(httpFilePath);
        System.out.println(httpFile.isFile());
        }
    }
```

ResourceUtils 的 getFile(String resourceLocation) 方法支持带特殊前缀的资源地址，这样，我们就可以在不和 Resource 实现类打交道的情况下使用 Spring 文件资源加载的功能了。

深入原理：如何将不同类型的资源加载进 ApplicationContext

- ResourceLoader

Spring 定义了一个接口 ResourceLoader，它提供了加载不同资源的策略。而 ApplicationContext 继承了实现类或者接口：ResourceLoader 和 ResourcePatternResolver。我们以 DefaultResourceLoader 来看：

```
@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);
        }
    }
}
```

1 协议解析

2 路径解析

3 classpath解析

4 url解析

- ResourceEditor

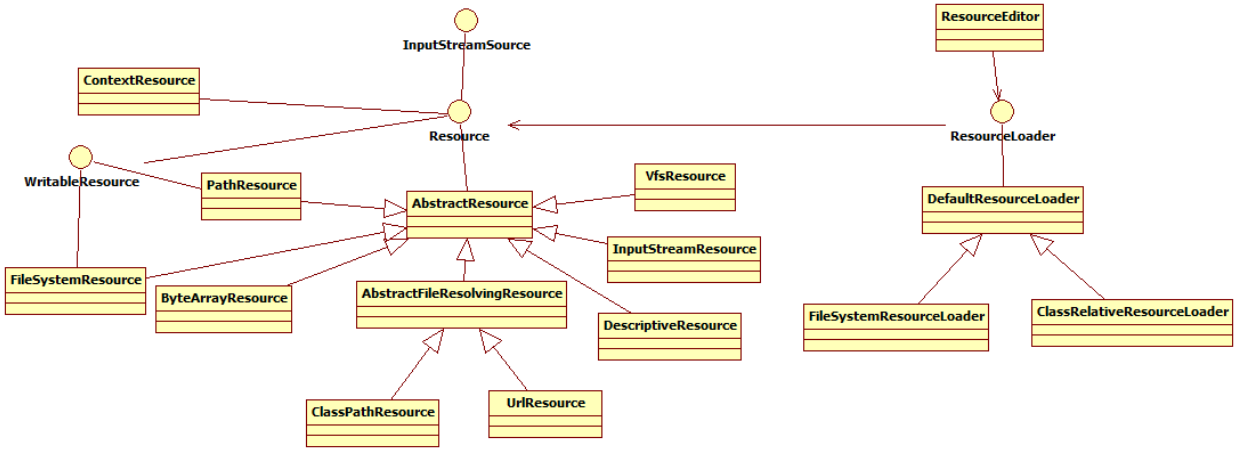
ResourceEditor 代理了 ResourceLoader，默认使用 DefaultResourceLoader 来解析路径。

总结

Resource 接口封装了各种可能的资源类型，也就是对使用者来说屏蔽了文件类型的不同。如下：

```
public static final String CLASSPATH_URL_PREFIX = "classpath:";

/** URL prefix for loading from the file system: "file:". */
public static final String FILE_URL_PREFIX = "file:";

/** URL prefix for loading from a jar file: "jar:". */
public static final String JAR_URL_PREFIX = "jar:";

/** URL prefix for loading from a war file on Tomcat: "war:". */
public static final String WAR_URL_PREFIX = "war:";

/** URL protocol for a file in the file system: "file". */
public static final String URL_PROTOCOL_FILE = "file";

/** URL protocol for an entry from a jar file: "jar". */
public static final String URL_PROTOCOL_JAR = "jar";

/** URL protocol for an entry from a war file: "war". */
public static final String URL_PROTOCOL_WAR = "war";

/** URL protocol for an entry from a zip file: "zip". */
public static final String URL_PROTOCOL_ZIP = "zip";

/** URL protocol for an entry from a WebSphere jar file: "wsjar". */
public static final String URL_PROTOCOL_WSJAR = "wsjar";

/** URL protocol for an entry from a JBoss jar file: "vfszip". */
public static final String URL_PROTOCOL_VFSZIP = "vfszip";

/** URL protocol for a JBoss file system resource: "vfsfile". */
public static final String URL_PROTOCOL_VFSFILE = "vfsfile";

/** URL protocol for a general JBoss VFS resource: "vfs". */
public static final String URL_PROTOCOL_VFS = "vfs";

/** File extension for a regular jar file: ".jar". */
public static final String JAR_FILE_EXTENSION = ".jar";

/** Separator between JAR URL and file path within the JAR: "!/". */
public static final String JAR_URL_SEPARATOR = "!/";

/** Special separator between WAR URL and jar part on Tomcat. */
public static final String WAR_URL_SEPARATOR = "*/";
```

抓住主要点 Resource、ResourceLoader 和 ResourceEditor



}

15 Spring IoC总结及热点面试题集萃

17 Spring IoC容器如何读取多个属性文件或者配置文件？