/\*

\* Copyright 2002-2008 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package org.springframework.core.io;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import org.springframework.util.Assert;

import org.springframework.util.ClassUtils;

import org.springframework.util.ObjectUtils;

import org.springframework.util.ResourceUtils;

import org.springframework.util.StringUtils;

/\*\*

\* {@link Resource} implementation for class path resources.

\* Uses either a given ClassLoader or a given Class for loading resources.

\*

\*

Supports resolution as java.io.File if the class path

\* resource resides in the file system, but not for resources in a JAR.

\* Always supports resolution as URL.

\*

\* @author Juergen Hoeller

\* @since 28.12.2003

\* @see java.lang.ClassLoader#getResourceAsStream(String)

\* @see java.lang.Class#getResourceAsStream(String)

\*/

public class ClassPathResource extends AbstractResource {

private final String path;

private ClassLoader classLoader;

private Class clazz;

/\*\*

\* Create a new ClassPathResource for ClassLoader usage.

\* A leading slash will be removed, as the ClassLoader

\* resource access methods will not accept it.

\*

The thread context class loader will be used for

\* loading the resource.

\* @param path the absolute path within the class path

\* @see java.lang.ClassLoader#getResourceAsStream(String)

\* @see org.springframework.util.ClassUtils#getDefaultClassLoader()

\*/

public ClassPathResource(String path) {

this(path, (ClassLoader) null);

}

/\*\*

\* Create a new ClassPathResource for ClassLoader usage.

\* A leading slash will be removed, as the ClassLoader

\* resource access methods will not accept it.

\* @param path the absolute path within the classpath

\* @param classLoader the class loader to load the resource with,

\* or null for the thread context class loader

\* @see java.lang.ClassLoader#getResourceAsStream(String)

\*/

public ClassPathResource(String path, ClassLoader classLoader) {

Assert.notNull(path, "Path must not be null");

if (path.startsWith("/")) {

path = path.substring(1);

}

this.path = StringUtils.cleanPath(path);

this.classLoader = (classLoader != null ? classLoader : ClassUtils.getDefaultClassLoader());

}

/\*\*

\* Create a new ClassPathResource for Class usage.

\* The path can be relative to the given class,

\* or absolute within the classpath via a leading slash.

\* @param path relative or absolute path within the class path

\* @param clazz the class to load resources with

\* @see java.lang.Class#getResourceAsStream

\*/

public ClassPathResource(String path, Class clazz) {

Assert.notNull(path, "Path must not be null");

this.path = StringUtils.cleanPath(path);

this.clazz = clazz;

}

/\*\*

\* Create a new ClassPathResource with optional ClassLoader and Class.

\* Only for internal usage.

\* @param path relative or absolute path within the classpath

\* @param classLoader the class loader to load the resource with, if any

\* @param clazz the class to load resources with, if any

\*/

protected ClassPathResource(String path, ClassLoader classLoader, Class clazz) {

this.path = path;

this.classLoader = classLoader;

this.clazz = clazz;

}

/\*\*

\* Return the path for this resource (as resource path within the class path).

\*/

public final String getPath() {

return this.path;

}

/\*\*

\* Return the ClassLoader that this resource will be obtained from.

\*/

public final ClassLoader getClassLoader() {

return (this.classLoader != null ? this.classLoader : this.clazz.getClassLoader());

}

/\*\*

\* This implementation opens an InputStream for the given class path resource.

\* @see java.lang.ClassLoader#getResourceAsStream(String)

\* @see java.lang.Class#getResourceAsStream(String)

\*/

public InputStream getInputStream() throws IOException {

InputStream is = null;

if (this.clazz != null) {

is = this.clazz.getResourceAsStream(this.path);

}

else {

is = this.classLoader.getResourceAsStream(this.path);

}

if (is == null) {

throw new FileNotFoundException(

getDescription() + " cannot be opened because it does not exist");

}

return is;

}

/\*\*

\* This implementation returns a URL for the underlying class path resource.

\* @see java.lang.ClassLoader#getResource(String)

\* @see java.lang.Class#getResource(String)

\*/

public URL getURL() throws IOException {

URL url = null;

if (this.clazz != null) {

url = this.clazz.getResource(this.path);

}

else {

url = this.classLoader.getResource(this.path);

}

if (url == null) {

throw new FileNotFoundException(

getDescription() + " cannot be resolved to URL because it does not exist");

}

return url;

}

/\*\*

\* This implementation returns a File reference for the underlying class path

\* resource, provided that it refers to a file in the file system.

\* @see org.springframework.util.ResourceUtils#getFile(java.net.URL, String)

\*/

public File getFile() throws IOException {

return ResourceUtils.getFile(getURL(), getDescription());

}

/\*\*

\* This implementation determines the underlying File

\* (or jar file, in case of a resource in a jar/zip).

\*/

protected File getFileForLastModifiedCheck() throws IOException {

URL url = getURL();

if (ResourceUtils.isJarURL(url)) {

URL actualUrl = ResourceUtils.extractJarFileURL(url);

return ResourceUtils.getFile(actualUrl);

}

else {

return ResourceUtils.getFile(url, getDescription());

}

}

/\*\*

\* This implementation creates a ClassPathResource, applying the given path

\* relative to the path of the underlying resource of this descriptor.

\* @see org.springframework.util.StringUtils#applyRelativePath(String, String)

\*/

public Resource createRelative(String relativePath) {

String pathToUse = StringUtils.applyRelativePath(this.path, relativePath);

return new ClassPathResource(pathToUse, this.classLoader, this.clazz);

}

/\*\*

\* This implementation returns the name of the file that this class path

\* resource refers to.

\* @see org.springframework.util.StringUtils#getFilename(String)

\*/

public String getFilename() {

return StringUtils.getFilename(this.path);

}

/\*\*

\* This implementation returns a description that includes the class path location.

\*/

public String getDescription() {

return "class path resource [" + this.path + "]";

}

/\*\*

\* This implementation compares the underlying class path locations.

\*/

public boolean equals(Object obj) {

if (obj == this) {

return true;

}

if (obj instanceof ClassPathResource) {

ClassPathResource otherRes = (ClassPathResource) obj;

return (this.path.equals(otherRes.path) &&

ObjectUtils.nullSafeEquals(this.classLoader, otherRes.classLoader) &&

ObjectUtils.nullSafeEquals(this.clazz, otherRes.clazz));

}

return false;

}

/\*\*

\* This implementation returns the hash code of the underlying

\* class path location.

\*/

public int hashCode() {

return this.path.hashCode();

}

}

<https://jar-download.com/artifacts/org.springframework/spring-core/2.5.4/source-code/org/springframework/core/io/ClassPathResource.java>

### 4.7.1 Constructing application contexts

An application context constructor (for a specific application context type) generally takes a string or array of strings as the location path(s) of the resource(s) such as XML files that make up the definition of the context.

When such a location path doesn't have a prefix, the specific Resource type built from that path and used to load the bean definitions, depends on and is appropriate to the specific application context. For example, if you create a ClassPathXmlApplicationContext as follows:

ApplicationContext ctx = **new** ClassPathXmlApplicationContext("conf/appContext.xml");

The bean definitions will be loaded from the classpath, as a ClassPathResource will be used. But if you create a FileSystemXmlApplicationContext as follows:

ApplicationContext ctx =

**new** FileSystemXmlApplicationContext("conf/appContext.xml");

The bean definition will be loaded from a filesystem location, in this case relative to the current working directory.

Note that the use of the special classpath prefix or a standard URL prefix on the location path will override the default type of Resource created to load the definition. So this FileSystemXmlApplicationContext...

ApplicationContext ctx =

**new** FileSystemXmlApplicationContext("classpath:conf/appContext.xml");

... will actually load its bean definitions from the classpath(*ClassPathResource will be created in this case*). However, it is still a FileSystemXmlApplicationContext. If it is subsequently used as a ResourceLoader, any unprefixed paths will still be treated as filesystem paths.

<https://docs.spring.io/spring/docs/3.0.0.RC3/spring-framework-reference/html/ch04s07.html>

Class.getResource can take a "relative" resource name, which is treated relative to the class's package. Alternatively you can specify an "absolute" resource name by using a leading slash. Classloader resource paths are always deemed to be absolute.

So the following are basically equivalent:

foo.bar.Baz.class.getResource("xyz.txt");

foo.bar.Baz.class.getClassLoader().getResource("foo/bar/xyz.txt");

And so are these (but they're different from the above):

foo.bar.Baz.class.getResource("/data/xyz.txt");

foo.bar.Baz.class.getClassLoader().getResource("data/xyz.txt");

<https://stackoverflow.com/questions/6608795/what-is-the-difference-between-class-getresource-and-classloader-getresource>

## **4.4. The ResourceLoader**

The ResourceLoader interface is meant to be implemented by objects that can return (i.e. load) Resource instances.

public interface ResourceLoader {

Resource getResource(String location);

}

All application contexts implement the ResourceLoader interface, and therefore all application contexts may be used to obtain Resource instances.

When you call getResource() on a specific application context, and the location path specified doesn't have a specific prefix, you will get back a Resource type that is appropriate to that particular application context. For example, assume the following snippet of code was executed against a ClassPathXmlApplicationContext instance:

Resource template = ctx.getResource("some/resource/path/myTemplate.txt);

What would be returned would be a ClassPathResource; if the same method was executed against a FileSystemXmlApplicationContext instance, you'd get back a FileSystemResource. For a WebApplicationContext, you'd get back a ServletContextResource, and so on.

As such, you can load resources in a fashion appropriate to the particular application context.

On the other hand, you may also force ClassPathResource to be used, regardless of the application context type, by specifying the special classpath: prefix:

Resource template = ctx.getResource("classpath:some/resource/path/myTemplate.txt);

Similarly, one can force a UrlResource to be used by specifying any of the standard java.net.URL prefixes:

Resource template = ctx.getResource("file:/some/resource/path/myTemplate.txt);

Resource template = ctx.getResource("http://myhost.com/resource/path/myTemplate.txt);

The following table summarizes the strategy for converting Strings to Resources:

**Table 4.1. Resource strings**

| **Prefix** | **Example** | **Explanation** |
| --- | --- | --- |
| classpath: | classpath:com/myapp/config.xml | Loaded from the classpath. |
| file: | file:/data/config.xml | Loaded as a URL, from the filesystem. [[a](https://docs.spring.io/spring/docs/2.5.x/reference/resources.html" \l "ftn.d0e8268)] |
| http: | http://myserver/logo.png | Loaded as a URL. |
| (none) | /data/config.xml | Depends on the underlying ApplicationContext. |
| [[a](https://docs.spring.io/spring/docs/2.5.x/reference/resources.html" \l "d0e8268)]But see also the section entitled [Section 4.7.3, “FileSystemResource caveats”](https://docs.spring.io/spring/docs/2.5.x/reference/resources.html" \l "resources-filesystemresource-caveats" \o "4.7.3. FileSystemResource caveats). | | |

**.....**

## **4.7. Application contexts and Resource paths**

### **4.7.1. Constructing application contexts**

An application context constructor (for a specific application context type) generally takes a string or array of strings as the location path(s) of the resource(s) such as XML files that make up the definition of the context.

When such a location path doesn't have a prefix, the specific Resource type built from that path and used to load the bean definitions, depends on and is appropriate to the specific application context. For example, if you create a ClassPathXmlApplicationContext as follows:

ApplicationContext ctx = new ClassPathXmlApplicationContext("conf/appContext.xml");

The bean definitions will be loaded from the classpath, as a ClassPathResource will be used. But if you create a FileSystemXmlApplicationContext as follows:

ApplicationContext ctx =

new FileSystemXmlApplicationContext("conf/appContext.xml");

The bean definition will be loaded from a filesystem location, in this case relative to the current working directory.

Note that the use of the special classpath prefix or a standard URL prefix on the location path will override the default type of Resource created to load the definition. So this FileSystemXmlApplicationContext...

ApplicationContext ctx =

new FileSystemXmlApplicationContext("classpath:conf/appContext.xml");

... will actually load its bean definitions from the classpath. However, it is still a FileSystemXmlApplicationContext. If it is subsequently used as a ResourceLoader, any unprefixed paths will still be treated as filesystem paths.

#### **4.7.1.1. Constructing ClassPathXmlApplicationContext instances - shortcuts**

The ClassPathXmlApplicationContext exposes a number of constructors to enable convenient instantiation. The basic idea is that one supplies merely a string array containing just the filenames of the XML files themselves (without the leading path information), and one also supplies a Class; the ClassPathXmlApplicationContext will derive the path information from the supplied class.

An example will hopefully make this clear. Consider a directory layout that looks like this:

com/

foo/

services.xml

daos.xml

MessengerService.class

A ClassPathXmlApplicationContext instance composed of the beans defined in the 'services.xml' and 'daos.xml' could be instantiated like so...

ApplicationContext ctx = new ClassPathXmlApplicationContext(

new String[] {"services.xml", "daos.xml"}, MessengerService.class);

Please do consult the Javadocs for the ClassPathXmlApplicationContext class for details of the various constructors.

### **4.7.2. Wildcards in application context constructor resource paths**

The resource paths in application context constructor values may be a simple path (as **shown above**) which has a one-to-one mapping to a target Resource, or alternately may contain the special "classpath\*:" prefix and/or internal Ant-style regular expressions (matched using Spring's PathMatcher utility). Both of the latter are effectively wildcards

One use for this mechanism is when doing component-style application assembly. All components can 'publish' context definition fragments to a well-known location path, and when the final application context is created using the same path prefixed via classpath\*:, all component fragments will be picked up automatically.

Note that this wildcarding is specific to use of resource paths in application context constructors (or when using the PathMatcher utility class hierarchy directly), and is resolved at construction time. It has nothing to do with the Resource type itself. It's not possible to use the classpath\*: prefix to construct an actual Resource, as a resource points to just one resource at a time.

<https://docs.spring.io/spring/docs/2.5.x/reference/resources.html>