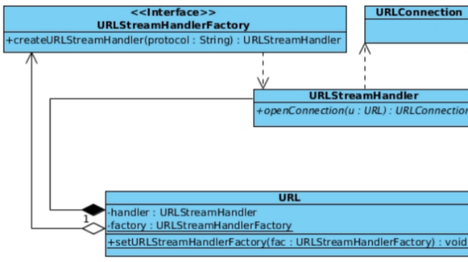
4、训练营期间：BeanDefinition的基础——资源

2020/11/24　　09:52

1、资源的定位

使用Java URL，有了JAVA，资源的定位和资源就分开了，资源可以位于文件系统，也可以位于网络中。



java.net.URLStreamHandler的具体实现有如下几种：

协议	实现类
file	sun.net.www.protocol.file.Handler
ftp	sun.net.www.protocol.ftp.Handler
http	sun.net.www.protocol.http.Handler
https	sun.net.www.protocol.https.Handler
jar	sun.net.www.protocol.jar.Handler
mailto	sun.net.www.protocol.mailto.Handler
netdoc	sun.net.www.protocol.netdoc.Handler

2、Spring资源接口

org.springframework.core.io.Resource。实现类如下：

- 可写资源接口
  - org.springframework.core.io.WritableResource
  - org.springframework.core.io.FileSystemResource
  - org.springframework.core.io.FileUrlResource（@since 5.0.2）
  - org.springframework.core.io.PathResource（@since 4.0 & @Deprecated）
- 编码资源接口
  - org.springframework.core.io.support.EncodedResource

直接使用Resource加载资源：

```
String currentJavaFilePath = System.getProperty("user.dir") +
    "/thinking-in-
spring/resource/src/main/java/org/geekbang/thinking/in/spring/resource/EncodedFileSyst
emResourceDemo.java";
// FileSystemResource => WritableResource => Resource
FileSystemResource fileSystemResource =
    new FileSystemResource(currentJavaFilePath);
EncodedResource encodedResource =
    new EncodedResource(fileSystemResource, "UTF-8");
// 字符输入流
// 字符输入流
try (Reader reader = encodedResource.getReader()) {
    System.out.println(IOUtils.toString(reader));
}
```

3、资源的加载器——ResourceLoader

Resource 加载器
- org.springframework.core.io.ResourceLoader
  - org.springframework.core.io.DefaultResourceLoader
    - org.springframework.core.io.FileSystemResourceLoader
    - org.springframework.core.io.ClassRelativeResourceLoader
    - org.springframework.context.support.AbstractApplicationContext

使用ResourceLoader加载资源

```
String currentJavaFilePath = "/" +
    System.getProperty("user.dir") +
    "/thinking-in-
spring/resource/src/main/java/org/geekbang/thinking/in/spring/resource/EncodedFileSyst
emResourceLoaderDemo.java";
// 新建一个 FileSystemResourceLoader 对象
FileSystemResourceLoader resourceLoader = new FileSystemResourceLoader();
// FileSystemResource => WritableResource => Resource
Resource resource = resourceLoader.getResource(currentJavaFilePath);
EncodedResource encodedResource = new EncodedResource(resource, "UTF-8");
// 字符输入流
try (Reader reader = encodedResource.getReader()) {
    System.out.println(IOUtils.toString(reader));
}
```

此外，还有一个通配符资源加载器。

4、使用注解注入资源

基于 @Value 实现， 如：
@Value("classpath:/...")
private Resource resource;

@Value不仅可以注入外部化配置，也可以用于注入资源，注入资源时也支持模式匹配。
@Value和@Autowired都依赖AutowiredAnnotationBeanPostProcessor。

@Value注入发生在@PostConstruct之前。

```
public class InjectingResourceDemo {

  @Value("classpath:/META-INF/default.properties")
  private Resource defaultPropertiesResource;

  @Value("classpath*:/META-INF/*.properties") //模式匹配要使用classpath*
  private Resource[] propertiesResources;

  @Value("${user.dir}")
  private String currentProjectRootPath;

  @PostConstruct
  public void init() {
    System.out.println(ResourceUtils.getContent(defaultPropertiesResource));
    System.out.println("================");
    Stream.of(propertiesResources).map(ResourceUtils::getContent).forEach(System.out::println);
    System.out.println("================");
    System.out.println(currentProjectRootPath);
  }

  public static void main(String[] args) {

    AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();
    // 注册当前类作为 Configuration Class
    context.register(InjectingResourceDemo.class);
    // 启动 Spring 应用上下文
    context.refresh();
    // 关闭 Spring 应用上下文
    context.close();

  }
}
```

5、还可以注入Resource Loader

- 方法一：实现 ResourceLoaderAware回调，覆盖setResourceLoader方法，参数就是框架解析完成的AbstractApplicationContext （DefaultResourceLoader的子类），原理参见《IOC依赖来源》。
- 方法二：@Autowired注入 ResourceLoader
- 方法三：注入 ApplicationContext 作为 ResourceLoader。

上面三个方法对应的是同一个BEAN实例。