Resource接口

Spring 的 Resource 接口位于 org.springframework.core.io 中。 用于抽象对低级资源的访问。

getInputStream(): 找到并打开资源，返回一个InputStream以从资源中读取。预计每次调用都会返回一个新的InputStream()，调用者有责任关闭每个流

exists(): 返回一个布尔值，表明某个资源是否以物理形式存在

isOpen: 返回一个布尔值，指示此资源是否具有开放流的句柄。如果为true，InputStream就不能够多次读取，只能够读取一次并且及时关闭以避免内存泄漏。对于所有常规资源实现，返回false，但是InputStreamResource除外。

getDescription(): 返回资源的描述，用来输出错误的日志。这通常是完全限定的文件名或资源的实际URL

Resource的实现类

- UrlResource访问网络资源
- ClassPathResource 访问类路径下资源
- FileSystemResource 访问文件系统资源
- InputStreamResource 输入流(InputStream)的Resource实现

ResourceLoader 接口

Spring 提供如下两个标志性接口：

（1）ResourceLoader： 通过ApplicationContext 该接口实现类的实例可以获得一个Resource实例。

（2） ResourceLoaderAware： 通过自动注入， 该接口实现类的实例将获得一个ResourceLoader的引用。

```
Resource res = ctx.getResource("classpath:bean.xml");
Resrouce res = ctx.getResource("file:bean.xml");
Resource res = ctx.getResource("http://192.168.18.13:4567/beans.xml");
//Okhttp 有丰富的 配置接口
//证书 服务端和客户端 双向加密
```

i18n概述

国际化也称作i18n，其来源是英文单词 internationalization的首末字符i和n，18为中间的字符数。由于软件发行可能面向多个国家，对于不同国家的用户，软件显示不同语言的过程就是国际化。通常来

讲，软件中的国际化是通过配置文件来实现的，假设要支撑两种语言，那么就需要两个版本的配置文件

- Java自身是支持国际化的，java.util.Locale用于指定当前用户所属的语言环境等信息，java.util.ResourceBundle用于查找绑定对应的资源文件。Locale包含了language信息和country信息
- 配置文件命名规则：basename_language_country.properties， 必须遵循命名规则，java才会识别。其中，basename是必须的，语言和国家是可选的。
- 如果同时提供了context.properties和context_zh_CN.propertes两个配置文件，如果提供的locale符合en_CN，那么优先查找messages_en_CN.propertes配置文件，如果没查找到，再查找messages.properties配置文件。所有的配置文件必须放在classpath中，一般放在resources目录下。

```
ResourceBundle.getBundle()  //获取国际化配置档，
//由于默认文件编码原因， 需要转码
new String(message.getString("").getBytes("ISO-8859-1"), "UTF-8");
```

MessageSource 国际化

Spring 框架提供的 多语言解决方案

- 初始化 ResourceBundleMessageSource类
- 配置 basenames 即文件名
- 配置 defaultEncoding 解决中文乱码问题
- 需要注意 使用 springConfig类初始化 I18n的情况下 bean对象名字 必须是 messageSource

整合Junit

- **配置Junit4**
- **整合SpringTest**

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.1.9.RELEASE</version>
    <scope>test</scope>
```

```
</dependency>
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = SpringConfig.class)
public class UserBuyBookServiceImplTest {

}
```

整合 Log4J

Log4j主要有三个组件：

- Logger：负责供客户端代码调用，执行debug(Object msg)、info(Object msg)、warn(Object msg)、error(Object msg)等方法。
- Appender：负责日志的输出，Log4j已经实现了多种不同目标的输出方式，可以向文件输出日志、向控制台输出日志、向Socket输出日志等。
- Layout：负责日志信息的格式化。

log4j的日志级别

一共七种，从高到低依次为

- OFF 最高日志级别，即关闭日志
- FATAL　导致应用程序退出的错误
- ERROR 运行发生错误，但不影响程序继续运行
- WARN
- INFO
- DEBUG
- ALL

注意文件命名 slf4j.xml

依赖包

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.32</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
```

```
    <version>1.2.10</version>
</dependency>
```

配置档

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <property name="log.path" value="./" />
    <property name="log.pattern" value="%d{HH:mm:ss.SSS} [%thread] %-5level %logger{20} - [%method,%line] - %msg%n" />
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
    </appender>
    <appender name="file_info" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.path}/sys-info.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${log.path}/sys-info.%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>60</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
    </appender>
    <appender name="file_error" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.path}/sys-error.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${log.path}/sys-error.%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>60</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>${log.pattern}</pattern>
        </encoder>
    </appender>
    <logger name="com.lovecoding" level="debug" additivity="false">
        <appender-ref ref="console"/>
        <appender-ref ref="file_info"/>
    </logger>
```

```
34 </configuration>
```

使用方法

```
1 private static final Logger logger = LoggerFactory.getLogger(Application.class);
2 logger.info( "项目启动-----------------");
```