

特殊文件与日志技术

鸣谢：黑马程序员



一、特殊文件

1. 分类

- properties属性文件：

```
1 name=张三
2 password=123456
3 address=北京
4 sex=男
5 ...
```

- xml文件：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <users>
3   <user id="1">
4     <name>张三</name>
5     <password>123456</password>
6     <address>北京</address>
7     <sex>男</sex>
8   </user>
9 
```

```
10     <user id="2">
11         <name>陈叶</name>
12         <password>888888</password>
13         <address>上海</address>
14         <sex>女</sex>
15     </user>
16
17     ...
18 </users>
```

- **适用场景：**

- 属性文件适用于存储单用户数据，简单方便；
 - XML文件适用于存储多用户数据，结构清晰。
-

2.properties属性文件

2.1 特点

- 只能存储键值对数据，且键不能重复。
- 文件后缀一般是.properties。

2.2 Properties

- `Properties` 是一个 `Map` 集合，但是我们一般不会把它当做集合来使用，它是用来代表属性文件的。
- **常用方法：**

序号	方法	说明
01	<code>void load(InputStream is)</code>	通过字节输入流，读取属性文件里的键值对数据。
02	<code>void load(Reader reader)</code>	通过字符输入流，读取属性文件里的键值对数据。
03	<code>String getProperty(String key)</code>	根据键获取值。
04	<code>Set<String> stringPropertyNames()</code>	获取全部键的集合。
05	<code>Object setProperty(String key, String value)</code>	将键值对数据保存到 <code>Properties</code> 对象中。
06	<code>void storeOutputStream os, String comments)</code>	通过字节输出流，将键值对数据保存到属性文件里， <code>comments</code> 表示附带注释。
07	<code>void storeWriter writer, String comments)</code>	通过字符输出流，将键值对数据保存到属性文件里。

2.3 读取数据

- `users.properties` :

```

1 # 以下内容都是用户名和密码
2 admin=123456
3 zsh=888
4 zjl=76543
5 zxj=$hit

```

- `Test1` :

```

1 import java.io.FileReader;
2 import java.util.Properties;
3

```

```
4 public class Test1 {
5     public static void main(String[] args) throws Exception {
6         Properties properties = new Properties();
7         properties.load(new
8             FileReader("test9\\src\\users.properties"));
9         System.out.println(properties);
10        System.out.println(properties.getProperty("admin"));
11        for (String name : properties.stringPropertyNames()) {
12            System.out.println(name + "=>" +
13                properties.getProperty(name));
14        }
15        System.out.println("-----");
16        // forEach
17        properties.forEach((name, password) -> {
18            System.out.println(name + "=>" + password);
19        });
20    }
21 }
```

- 控制台输出：

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\CS-Softwares\Intel
{admin=123456, zjl=76543, zxj=shit, zsh=888}
123456
admin=>123456
zjl=>76543
zxj=>shit
zsh=>888
-----
admin=>123456
zjl=>76543
zxj=>shit
zsh=>888

Process finished with exit code 0
```

2.4 存储数据

- Test2：

```
1 import java.io.FileWriter;
2 import java.util.Properties;
3
4 public class Test2 {
5     public static void main(String[] args) throws Exception {
6         Properties properties = new Properties();
7         properties.setProperty("lily", "love");
8         properties.setProperty("ghost", "hidden");
9         properties.setProperty("arthur", "goddamn_man");
10
11         properties.store(new
12             FileWriter("test9/src/users2.properties"), "save 3 users' information");
13     }
14 }
```

- 执行后查看users2.properties文件的内容：

```
1 #save 3 users' information
2 #Sat Nov 15 00:13:01 CST 2025
3 arthur=goddamn_man
4 ghost=hidden
5 lily=love
```

3.xml文件

Important

XML: Extensible Markup Language, 意思是可扩展标记性语言。

它本质上是一种特殊数据格式，可以用来存储复杂的数据结构和数据关系。

应用场景：经常用来作为系统的配置文件；或者作为一种特殊数据结构，在网络中进行传输。

3.1 特点

- xml的“”称为一个标签或一个元素，一般成对出现，标签名可自定义（这就是为什么叫“可扩展”），但必须要正确嵌套。
- xml只能有一个根标签。
- xml的标签可以有属性，比如。
- 文件后缀一般是.xml。

3.2 语法规则

- xml文件的第一行必须是如下文档抬头：
- xml中书写“<”、“&”等特殊符号时，可能会出现冲突导致报错，此时可以用下面的特殊字符替代：

冲突字符	替代字符（必须+分号）
<	<
>	>
&	&
'	'
"	"

- xml中可以写一个叫作CDATA的数据区：，里面的内容可以随便写。
-

3.3 读取数据（解析xml文件）

P1 `dom4j`

`dom4j`

`dom4j` is an open source framework for processing XML which is integrated with XPath and fully supports DOM, SAX, JAXP and the Java platform such as Java 2 Collections.

See <https://dom4j.github.io> for details.

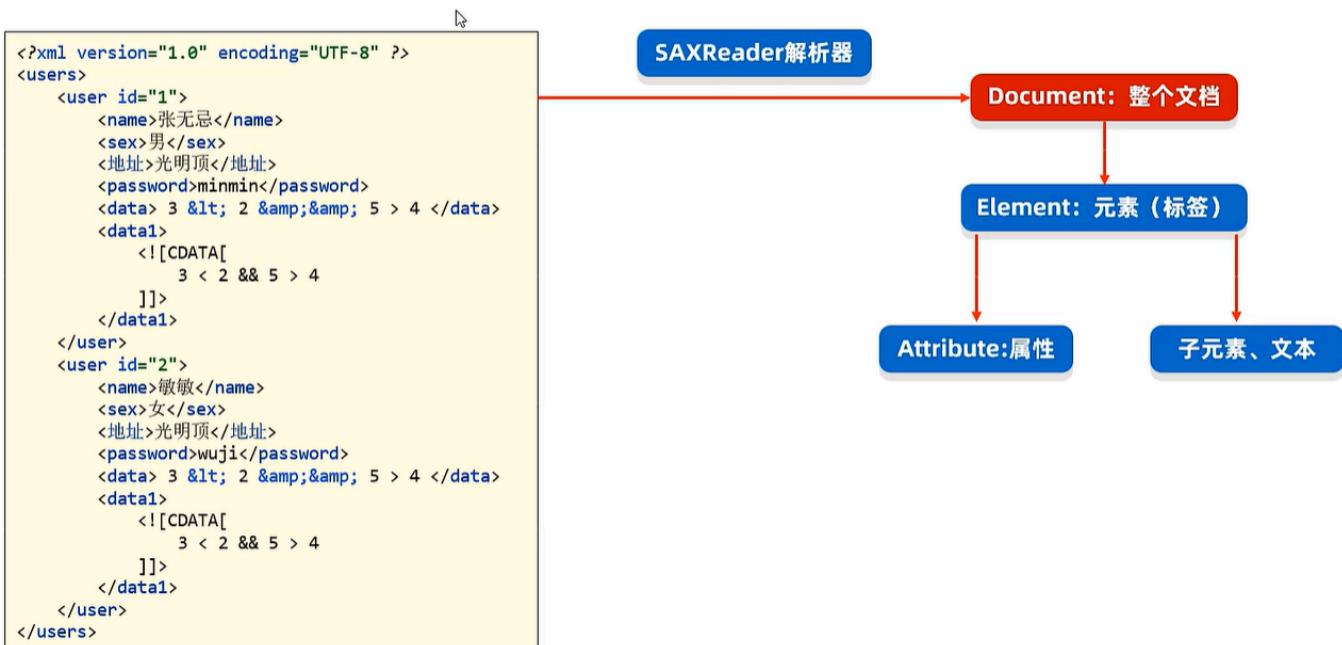
`dom4j` 是用于解析xml的最知名的第三方开源框架，十分便捷。

P2 `IDEA` 中使用 `dom4j`

1. 从官网下载 `dom4j` 框架。
2. 在项目中创建一个 `lib` 文件夹。
3. 将 `dom4j-x.y.z.jar` 复制到 `lib` 文件夹下。
4. 右键此 `jar` 文件，选择 `Add as Library`，在弹出的对话框中点击OK。
5. 在类中导包使用。

P3 dom4j 解析xml的核心思想：DOM（文档对象模型）

DOM4J解析XML文件的思想：文档对象模型



- SAXReader : dom4j 提供的解析器，可以认为它代表了整个框架。

构造器/方法	说明
public SAXReader()	创建 dom4j 的解析器对象。
Document read(String url)	将xml文件读取成文档对象。
Document read(InputStream is)	通过字节输入流，读取xml文件。

- Document :

方法	说明
Element getRootElement()	获得根元素对象。

- Element :

方法	说明
<code>String getName()</code>	获取元素名字。
<code>List<Element> elements()</code>	获取当前元素下的所有一级子元素。
<code>List<Element> elements(String name)</code>	获取当前元素下指定名字的所有一级子元素。
<code>Element element(String name)</code>	获取当前元素下指定名字的一级子元素，若多个则返回第一个。
<code>String attributeValue(String name)</code>	通过属性名获取属性值。
<code>String elementText(String childElementName)</code>	获取指定名字的一级子元素的文本。
<code>String getText()</code>	获取元素文本。

P4 代码演示

- `users.xml`:

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <users>
3      <user id="1">
4          <name>张三</name>
5          <password>123456</password>
6          <address>北京</address>
7          <sex>男</sex>
8      </user>
9
10     <user id="2">
11         <name>陈叶</name>
12         <password>888888</password>
13         <address>上海</address>
14         <sex>女</sex>
15     </user>
16

```

```
17  </users>
```

- Dom4jTest1:

```
1 import org.dom4j.*;
2 import org.dom4j.io.SAXReader;
3
4 import java.util.List;
5
6 public class Dom4jTest1 {
7     public static void main(String[] args) throws Exception {
8         SAXReader saxReader = new SAXReader();
9         Document document = saxReader.read("test9/src/users.xml");
10        Element root = document.getRootElement();
11        System.out.println("root: " + root.getName());
12
13        List<Element> elements = root.elements();
14        for (Element element : elements) {
15            System.out.println("level 1: " + element.getName());
16        }
17
18        Element user1 = root.element("user");
19        System.out.println("user1's name: " +
20 user1.elementText("name"));
21
22        System.out.println("id: " + user1.attributeValue("id"));
23    }
24 }
```

- 控制台输出：

```
D:\CS-Softwares\JDK\JDK21\bin\java.exe "-javaagent:D:\CS-S
root: users
level 1: user
level 1: user
张三
id: 1

Process finished with exit code 0
```

3.4 存储数据

Tip

不建议使用 `dom4j` 把数据写入xml文件中，因为要创建大量对象，复杂且不好维护。

推荐直接把数据拼接成xml格式，然后用IO流写入xml文件中。

- `Test`:

```
1 import java.io.BufferedWriter;
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class Test {
6     public static void main(String[] args) {
7         StringBuilder sb = new StringBuilder();
8         sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\" ?"
9             >\n");
10        sb.append("<book>\n");
11        sb.append("\t<name>").append("My Fighting").append(
12            "</name>\n");
13        sb.append("\t<author>").append("Hitler").append(
14            "</author>\n");
15        sb.append("\t<price>").append("666.66").append(
16            "</price>\n");
17        sb.append("</book>");
18
19        try {
20            BufferedWriter bw = new BufferedWriter(new
21                FileWriter("test9/src/book.xml"));
22        } {
23            bw.write(sb.toString());
24        } catch (IOException e) {
25            throw new RuntimeException(e);
26        }
27    }
28}
```

- 执行后查看book.xml文件的内容：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <book>
3   <name>My Fighting</name>
4   <author>Hitler</author>
5   <price>666.66</price>
6 </book>
```

3.5 约束xml文件的编写

💡 Tip

就是限制xml文件只能按照某种格式进行编写。

P1 约束文档

- 专门用来约束xml书写格式的文档，比如限制标签、属性应该怎么写。
- 分类：
 - DTD文档
 - Schema文档

P2 DTD文档(Document Type Definition: 文档类型定义)

⚠ Caution

无法约束具体数据类型。

比如程序员编写xml文档时写了很便宜，不会报错。

1. 编写DTD约束文档 `book_rule.dtd`，后缀必须是.dtd。

```
1 <!ELEMENT bookshelf (book+)>
2 <!ELEMENT book (name,author,price)>
3 <!ELEMENT name (#PCDATA)>
4 <!ELEMENT author (#PCDATA)>
5 <!ELEMENT price (#PCDATA)>
```

2. 在需要编写的xml文件中导入该DTD约束文档。

The screenshot shows an IDE interface with three tabs: 'Test.java' (selected), 'book.xml' (underlined), and 'book_rule.dtd'. The 'book.xml' tab contains the following XML code:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE bookshelf SYSTEM "book_rule.dtd">
3 <bookshelf>
4   <book>
5     <name>RDR2</name>
6     <author>Rockstar</author>
7     <price>很便宜</price>
8   </book>
9 </bookshelf>
10
```

3. 然后此xml文件就必须按照DTD约束文档规定的格式进行编写，否则会报错。

The screenshot shows an IDE interface with three tabs: 'Test.java' (selected), 'book.xml' (underlined), and 'book_rule.dtd'. The 'book.xml' tab contains the following XML code, with a validation error highlighted at line 4:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE bookshelf SYSTEM "book_rule.dtd">
3 <bookshelf>
4   <bo>
5     Element bo is not allowed here
6     <author>Rockstar</author>
7     <price>很便宜</price>
8   </bo>
9 </bookshelf>
10
```

A tooltip window appears over the 'bo' element at line 4, displaying the message: 'Element bo is not allowed here'.

P3 Schema文档

1. 编写Schema约束文档 `book_rule.xsd`，后缀必须是.xsd。

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!-- targetNamespace: 申明约束文档的地址（命名空间）-->
3 <!-- 注意xmlns的标准URI是http而非https -->
4 <schema xmlns="http://www.w3.org/2001/XMLSchema"
5         targetNamespace="http://zsh.com"
6         elementFormDefault="qualified">
7   <element name="bookshelf">
8     <complexType>
9       <!-- maxOccurs="unbounded"意思是bookshelf下面的子元素book
可以有无穷多个 -->
10    <sequence maxOccurs="unbounded">
11      <element name="book">
12        <complexType>
13          <sequence>
14            <element name="name" type="string"/>
15            <element name="author" type="string"/>
16            <element name="price" type="double"/>
17          </sequence>
18        </complexType>
19      </element>
20    </sequence>
21  </complexType>
22 </element>
23 </schema>
```

2. 在需要编写的xml文件中导入该Schema约束文档。

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!-- 注意xmlns:xsi的标准URI是http而非https -->
3 <bookshelf xmlns="http://zsh.com"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://zsh.com book_rule.xsd">
6
7     <book>
8         <name>RDR2</name>
9         <author>Rockstar</author>
10        <price>666.66</price>
11    </book>
12 </bookshelf>
13
```

3. 按照约束内容编写xml文件，否则报错。

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!-- 注意xmlns:xsi的标准URI是http而非https -->
3 <bookshelf xmlns="http://zsh.com"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://zsh.com book_rule.xsd">
6
7     <book>
8         <name>RDR2</name>
9         <author>Rockstar</author>
10        <price>很便宜</price>
11    </book>
12 </bookshelf>
13
```

二、日志技术

Important

把程序运行的信息存储到文件中，方便程序员定位bug，并了解程序的执行情况等。

1. 引入背景

- 直接输出到控制台的弊端：

- 日志会展示在控制台，一旦重新执行程序就会被清除掉。
- 不能更方便地将日志记录到其他位置，比如文件、数据库等。
- 若需要取消或修改日志，则必须修改源代码。

- 日志技术的优势：

- 可以将程序运行的信息，方便地记录到指定位置，比如控制台、文件、数据库。
- 可以随时以开关的形式控制日志的启停，无需修改源代码。

2. 日志技术的体系结构

- 日志接口：设计日志框架的一套标准，主流日志框架都需要实现这些接口，从而降低程序员学习和切换日志框架的压力。

- `JCL(java.commons.logging)`
- `slf4j(simple logging facade for java)`

- 主流日志框架：

- `JUL(java.util.logging)`：实现了 `JCL` 接口。
- `log4j`、`logback`：实现了 `slf4j` 接口。
- 现在最流行的日志框架是 `logback`。

3. logback

Tip

官网: <https://logback.qos.ch/index.html>

3.1 logback 的3大模块

- `logback-core`: 核心模块，为另外2个模块奠定了基础。
- `logback-classic`: 完整实现了 `slf4j` 接口。
- `logback-access`: 与Servlet容器（如Tomcat和Jetty）集成，提供HTTP访问日志功能。（可选）

注意：要想在项目中使用 `logback` 日志框架，至少要整合3个模块：`slf4j-api`、`logback-core`、`logback-classic`，然后还要在 `src` 目录下，自己创建 `logback` 日志框架的核心配置文件 `logback.xml`（也可以找别人写好的）。

3.2 代码演示

- `LogBackTest`:

```
1 import org.slf4j.Logger;
2 import org.slf4j.LoggerFactory;
3
4 public class LogBackTest {
5     private static final Logger LOGGER =
6         LoggerFactory.getLogger("LogBackTest");
7
8     public static void main(String[] args) {
9         try {
10             LOGGER.info("Method divide() starts.");
11             divide(10, 0);
12             LOGGER.info("Method divide() is executed
13             successfully.");
14         } catch (Exception e) {
15             LOGGER.error("Method divide() fails to execute! Please
16             check the input numbers!");
17     }
18 }
```

```

15     }
16
17     public static void divide(int a, int b) {
18         LOGGER.debug("param a: " + a + ", param b: " + b);
19         int c = a / b;
20         LOGGER.info("a/b = " + c);
21     }
22 }
```

- 控制台输出：

```

LogBackTest

23:05:17,176 |-INFO in c.q.l.core.rolling.SizeAndTimeBasedRollingPolicy@817348612 - Will use gz compression
23:05:17,179 |-INFO in c.q.l.core.rolling.SizeAndTimeBasedRollingPolicy@817348612 - Will use the pattern D:/ZSH-ComputerScienc
23:05:17,200 |-INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFileNamingAndTriggeringPolicy@79efed2d - The date pattern i
23:05:17,200 |-INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFileNamingAndTriggeringPolicy@79efed2d - Roll-over at midni
23:05:17,210 |-INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFileNamingAndTriggeringPolicy@79efed2d - Setting initial pe
23:05:17,217 |-INFO in ch.qos.logback.core.rolling.RollingFileAppender[FILE] - Active log file name: D:/ZSH-ComputerScience/L
23:05:17,217 |-INFO in ch.qos.logback.core.rolling.RollingFileAppender[FILE] - File property is set to [D:/ZSH-ComputerScience
23:05:17,219 |-INFO in ch.qos.logback.classic.model.processor.RootLoggerModelHandler - Setting level of ROOT logger to DEBUG
23:05:17,219 |-INFO in ch.qos.logback.core.model.processor.AppenderRefModelHandler - Attaching appender named [CONSOLE] to Log
23:05:17,220 |-INFO in ch.qos.logback.core.model.processor.AppenderRefModelHandler - Attaching appender named [FILE] to Logger
23:05:17,220 |-INFO in ch.qos.logback.core.model.processor.DefaultProcessor@2928854b - End of configuration.
23:05:17,220 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@27ae2fd0 - Registering current configuration as safe fal
23:05:17,221 |-INFO in ch.qos.logback.classic.util.ContextInitializer@d706f19 - ch.qos.logback.classic.util.DefaultJoranConfig

2025-11-15 23:05:17.225 [INFO ] LogBackTest [main] : Method divide() starts.
2025-11-15 23:05:17.227 [DEBUG] LogBackTest [main] : param a: 10, param b: 0
2025-11-15 23:05:17.228 [ERROR] LogBackTest [main] : Method divide() fails to execute! Please check the input numbers!

Process finished with exit code 0
|
```

- 日志文件：

```

D:\ZSH-ComputerScience\Log\JavaSE\zsh-data.log - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
zsh-data.log x
1 2025-11-15 23:05:17.225 [main] INFO LogBackTest - Method divide() starts.
2 2025-11-15 23:05:17.227 [main] DEBUG LogBackTest - param a: 10, param b: 0
3 2025-11-15 23:05:17.228 [main] ERROR LogBackTest - Method divide() fails to execute! Please check the input numbers!
4 |
```

3.3 核心配置文件 `logback.xml`

以下是黑马程序员提供的 `logback.xml`：

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <configuration>
3     <!-- CONSOLE: 将当前的日志信息输出到控制台 -->
4     <appender name="CONSOLE"
      class="ch.qos.logback.core.ConsoleAppender">
```

```
5      <!-- 输出流对象 默认 System.out 改为 System.err -->
6      <target>System.out</target>
7      <encoder>
8          <!--
9              格式化输出:
10             %d: 日期
11             %thread: 线程名
12             %-5level: 级别与左侧文字间隔5个字符宽度
13             %logger: 日志名称
14             %msg: 日志消息
15             %n: 换行符
16         -->
17         <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%-5level] %c
18     [<%thread>] : %msg%n</pattern>
19     </encoder>
20     </appender>
21
22     <!-- FILE: 将当前的日志信息输出到文件 -->
23     <appender name="FILE"
24         class="ch.qos.logback.core.rolling.RollingFileAppender">
25         <encoder>
26             <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level
27             %logger{36} - %msg%n</pattern>
28             <charset>utf-8</charset>
29         </encoder>
30         <!-- 指定日志输出到的文件路径 -->
31         <file>D:/ZSH-ComputerScience/Log/JavaSE/zsh-data.log</file>
32         <!-- 指定日志文件拆分和压缩规则, 避免单个日志文件过大 -->
33         <rollingPolicy
34             class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
35             <!-- 通过指定压缩文件名称, 来确定分割文件方式 -->
36             <fileNamePattern>D:/ZSH-ComputerScience/Log/JavaSE/zsh-
37             data%i-%d{yyyy-MM-dd}-.log.gz</fileNamePattern>
38             <!-- 文件拆分大小, 一旦单个日志文件大小超过1MB, 则会把前面1MB的部分
39             压缩成一个新文件, 以此类推 -->
40             <maxFileSize>1MB</maxFileSize>
41         </rollingPolicy>
42     </appender>
43
44     <!--
45         控制日志的输出情况: 开启日志(level="ALL"), 取消日志(level="OFF")。
46     -->
47     <root level="ALL">
48         <appender-ref ref="CONSOLE"/>
49         <appender-ref ref="FILE" />
```

```
44      </root>
45  </configuration>
```

3.4 日志级别

P1 分类

日志级别（从低到高）	说明	使用频率
trace	追踪，指明程序运行轨迹。	几乎不
debug	调试，实际应用中一般将其作为最低级别。	调试用
info	输出重要的运行信息，数据库连接、网络连接、IO操作等。	高
warn	警告，可能会在此处出现bug。	高
error	错误。	高

P2 设置日志级别

```
1  <root level="info"> // 这样配置后，只有级别>=info的日志才会输出到控制台和文件中
2    <appender-ref ref="CONSOLE"/>
3    <appender-ref ref="FILE" />
4  </root>
```