特殊文件与日志技术

> 鸣谢：黑马程序员
>
> 
>
> Java入门基础视频教程，java零基础自学就选黑马程序员Java...
> 黑马程序员 · 2021-9-22

一、特殊文件

1.分类

- properties属性文件：

```
name=张三
password=123456
address=北京
sex=男
...
```

- xml文件：

```
<?xml version="1.0" encoding="UTF-8" ?>
<users>
    <user id="1">
        <name>张三</name>
        <password>123456</password>
        <address>北京</address>
        <sex>男</sex>
    </user>

```

```xml
    <user id="2">
        <name>陈叶</name>
        <password>888888</password>
        <address>上海</address>
        <sex>女</sex>
    </user>

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

```
# 以下内容都是用户名和密码
admin=123456
zsh=888
zjl=76543
zxj=shit
```

- `Test1`：

```
import java.io.FileReader;
import java.util.Properties;

```

```java
public class Test1 {
    public static void main(String[] args) throws Exception {
        Properties properties = new Properties();
        properties.load(new
FileReader("test9\\src\\users.properties"));
        System.out.println(properties);
        System.out.println(properties.getProperty("admin"));
        for (String name : properties.stringPropertyNames()) {
            System.out.println(name + "==>" +
properties.getProperty(name));
        }
        System.out.println("------------------------------------");

        // forEach
        properties.forEach((name, password) -> {
            System.out.println(name + "==>" + password);
        });

    }
}
```

- 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`：

```java
import java.io.FileWriter;
import java.util.Properties;

public class Test2 {
    public static void main(String[] args) throws Exception {
        Properties properties = new Properties();
        properties.setProperty("lily", "love");
        properties.setProperty("ghost", "hidden");
        properties.setProperty("arthur", "goddamn_man");

        properties.store(new FileWriter("test9/src/users2.properties"), "save 3 users' information");
    }
}
```

- 执行后查看users2.properties文件的内容：

```properties
#save 3 users' information
#Sat Nov 15 00:13:01 CST 2025
arthur=goddamn_man
ghost=hidden
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（这就是为什么叫做“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

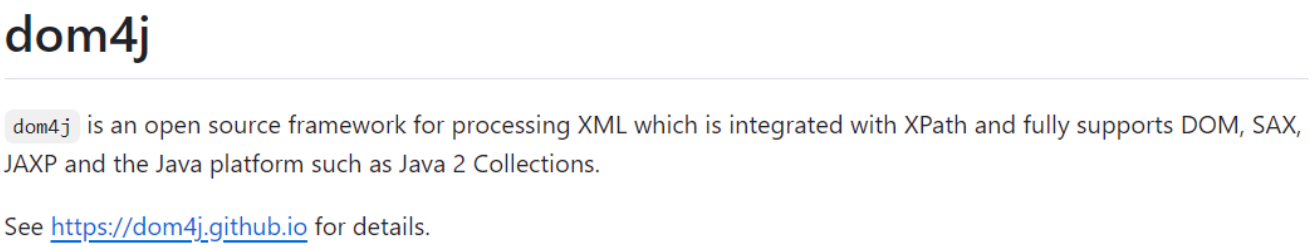
- xml文件的第一行必须是如下文档抬头：
- xml中书写“<”、“&”等特殊符号时，可能会出现冲突导致报错，此时可以用下面的特殊字符替代：

冲突字符	替代字符（必须+分号）
<	&lt;
>	&gt;
&	&amp;
'	&apos;
''	&quot;

- 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`是用于解析xml的最知名的第三方开源框架，十分便捷。

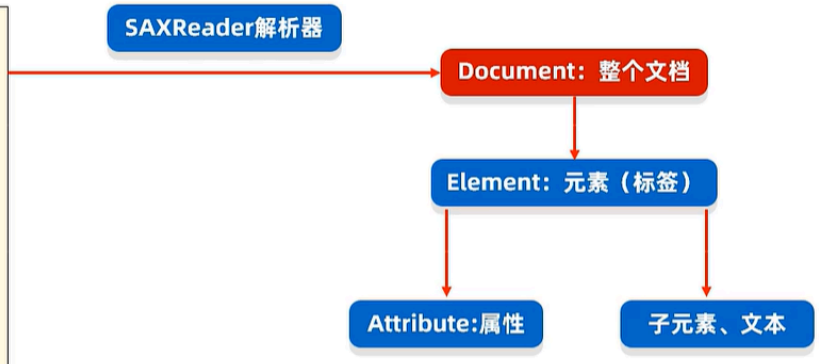
P2 `IDEA`中使用`dom4j`

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

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<users>
    <user id="1">
        <name>张无忌</name>
        <sex>男</sex>
        <地址>光明顶</地址>
        <password>minmin</password>
        <data> 3 &lt; 2 &amp;&amp; 5 > 4 </data>
        <data1>
            <![CDATA[
                3 < 2 && 5 > 4
            ]]>
        </data1>
    </user>
    <user id="2">
        <name>敏敏</name>
        <sex>女</sex>
        <地址>光明顶</地址>
        <password>wuji</password>
        <data> 3 &lt; 2 &amp;&amp; 5 > 4 </data>
        <data1>
            <![CDATA[
                3 < 2 && 5 > 4
            ]]>
        </data1>
    </user>
</users>
```



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

P4 代码演示

- `users.xml`:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<users>
    <user id="1">
        <name>张三</name>
        <password>123456</password>
        <address>北京</address>
        <sex>男</sex>
    </user>

    <user id="2">
        <name>陈叶</name>
        <password>888888</password>
        <address>上海</address>
        <sex>女</sex>
    </user>

```

```
</users>
```

- `Dom4jTest1`：

```
import org.dom4j.*;
import org.dom4j.io.SAXReader;

import java.util.List;

public class Dom4jTest1 {
    public static void main(String[] args) throws Exception {
        SAXReader saxReader = new SAXReader();
        Document document = saxReader.read("test9/src/users.xml");
        Element root = document.getRootElement();
        System.out.println("root: " + root.getName());

        List<Element> elements = root.elements();
        for (Element element : elements) {
            System.out.println("level 1: " + element.getName());
        }

        Element user1 = root.element("user");
        System.out.println("user1's name: " +
user1.elementText("name"));

        System.out.println("id: " + user1.attributeValue("id"));
    }
}
```

- 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`：

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"  ?>\n");
        sb.append("<book>\n");
        sb.append("\t<name>").append("My Fighting").append("</name>\n");
        sb.append("\t<author>").append("Hitler").append("</author>\n");
        sb.append("\t<price>").append("666.66").append("</price>\n");
        sb.append("</book>");

        try (
                BufferedWriter bw = new BufferedWriter(new FileWriter("test9/src/book.xml"));
        ) {
            bw.write(sb.toString());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    }
}
```

- 执行后查看book.xml文件的内容：

```xml
<?xml version="1.0" encoding="UTF-8"  ?>
<book>
    <name>My Fighting</name>
    <author>Hitler</author>
    <price>666.66</price>
</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。

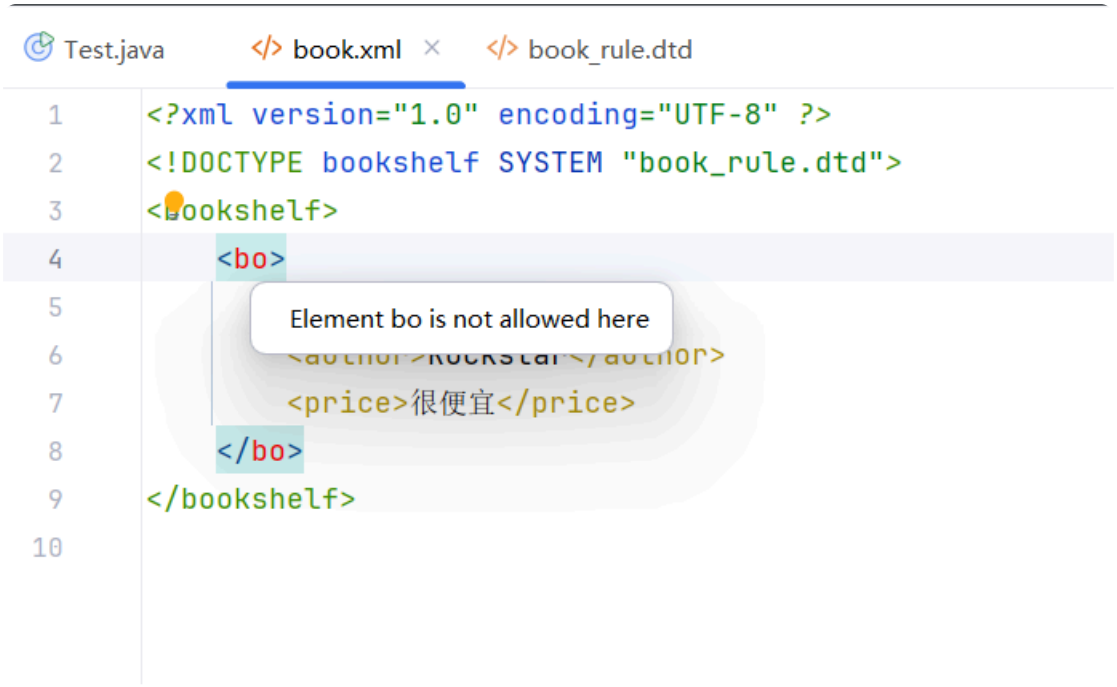
```
<!ELEMENT bookshelf (book+)>
<!ELEMENT book (name,author,price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

2. 在需要编写的xml文件中导入该DTD约束文档。

Test.java book.xml book_rule.dtd

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE bookshelf SYSTEM "book_rule.dtd">
<bookshelf>
    <book>
        <name>RDR2</name>
        <author>Rockstar</author>
        <price>很便宜</price>
    </book>
</bookshelf>
```

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。



P3 Schema文档

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

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- targetNameSpace: 申明约束文档的地址（命名空间）-->
<!-- 注意xmlns的标准URI是http而非https -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://zsh.com"
        elementFormDefault="qualified">
    <element name="bookshelf">
        <complexType>
            <!-- maxOccurs="unbounded"意思是bookshelf下面的子元素book可以有无穷多个 -->
            <sequence maxOccurs="unbounded">
                <element name="book">
                    <complexType>
                        <sequence>
                            <element name="name" type="string"/>
                            <element name="author" type="string"/>
                            <element name="price" type="double"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
</schema>
```

2. 在需要编写的xml文件中导入该Schema约束文档。

Test.java book.xml book_rule.xsd

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- 注意xmlns:xsi的标准URI是http而非https -->
<bookshelf xmlns="http://zsh.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://zsh.com book_rule.xsd">

    <book>
        <name>RDR2</name>
        <author>Rockstar</author>
        <price>666.66</price>
    </book>
</bookshelf>

```

3. 按照约束内容编写xml文件，否则报错。

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

- **直接输出到控制条的弊端**：
  - 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）。

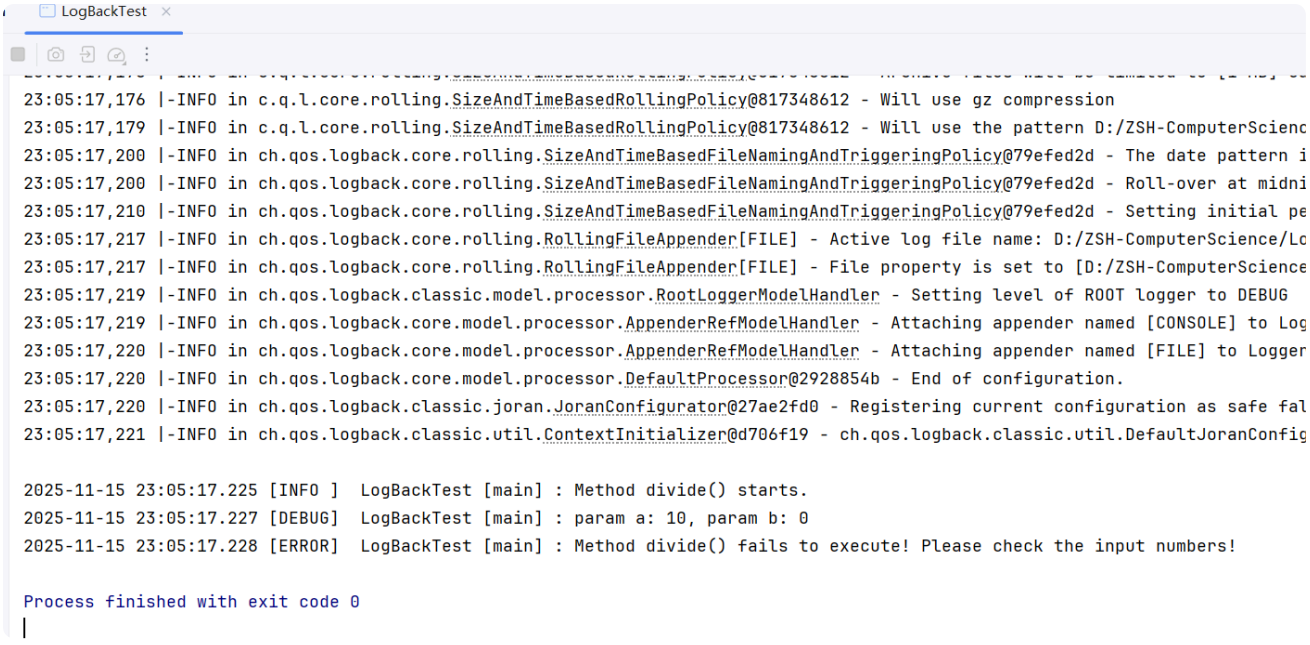
3.2 代码演示

- `LogBackTest`：

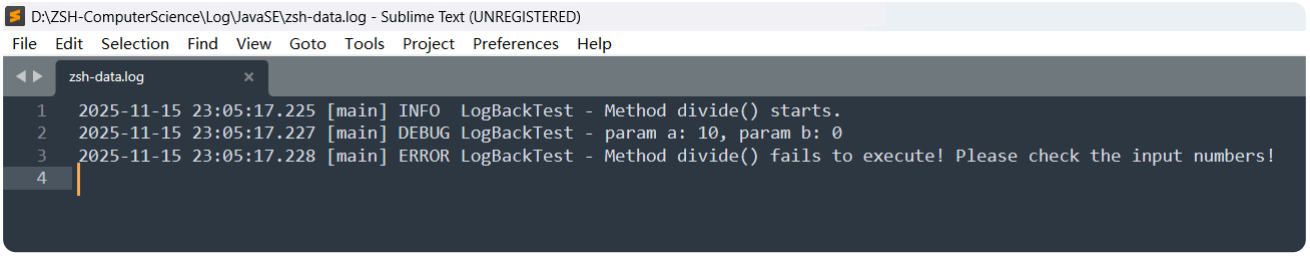
```
1  import org.slf4j.Logger;
2  import org.slf4j.LoggerFactory;
3
4  public class LogBackTest {
5      private static final Logger LOGGER =
   LoggerFactory.getLogger("LogBackTest");
6
7      public static void main(String[] args) {
8          try {
9              LOGGER.info("Method divide() starts.");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

```
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：



- 日志文件：



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
    [%thread] : %msg%n</pattern>
18          </encoder>
19      </appender>
20
21      <!-- FILE：将当前的日志信息输出到文件 -->
22      <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
23          <encoder>
24              <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level
    %logger{36} - %msg%n</pattern>
25              <charset>utf-8</charset>
26          </encoder>
27          <!-- 指定日志输出到的文件路径 -->
28          <file>D:/ZSH-ComputerScience/Log/JavaSE/zsh-data.log</file>
29          <!-- 指定日志文件拆分和压缩规则，避免单个日志文件过大 -->
30          <rollingPolicy
    class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
31              <!-- 通过指定压缩文件名称，来确定分割文件方式 -->
32              <fileNamePattern>D:/ZSH-ComputerScience/Log/JavaSE/zsh-
    data%i-%d{yyyy-MM-dd}-.log.gz</fileNamePattern>
33              <!-- 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
    分压缩成一个新文件，以此类推 -->
34              <maxFileSize>1MB</maxFileSize>
35          </rollingPolicy>
36      </appender>
37
38      <!--
39          控制日志的输出情况：开启日志(level="ALL")，取消日志(level="OFF")。
40      -->
41      <root level="ALL">
42          <appender-ref ref="CONSOLE"/>
43          <appender-ref ref="FILE" />
```

```
    </root>
</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

```
<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
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE" />
</root>
```