

## | 学习目标

- \* 能够掌握JUnit单元测试
  - \* 概念, Junit
  - \* @Test, @Before, @After, @BeforeClass, @AfterClass, @Ignore
  - \* TestAsset.equals();
- \* 能够掌握XML的概述
- \* 能够掌握XML的HelloWorld开发
- \* 能够熟悉XML的语法
  - <:&lt;
- \* 能够理解XML元素的概念
  - \* 节点, 元素
- \* 能够理解XML属性的概念
  - \* id
- \* 能够了解XML验证方式之DTD
- \* 能够了解XML验证方式之Schema
  - \* xml中能
- \* 能够掌握操作XML操作之DOM
  - \* DOM4J
  - \* Document, Node, Element, Attribute, Text
  - \* getRootElement
  - \* CRUD
- \* 能够掌握解析XML操作之SAX
- \* 能够了解解析XML操作之Pull

---

\* 面试岗位：中高级Java工程师：期望薪资：16-18K

\* 基础：

- \* ArrayList与LinkedList的区别

- \* 在JDK1.8的对HashMap做哪些优化

- \* JVM如何加装类，类加载过程

- \* 类加载过程：加载（ class-->java.lang.Class ） -->链接（ 验证->准备-->解析 ） -->类的初始化（ 父静态-子类静态代码块-父非静态代码块--父构造器-子类非静态代码块--子类构造器 ） --使用--卸载

- \* 框架

- \* SpringBoot加载原理

- \* Dubbo 的协议

- \* Mysql做分库，分表，那么Redis如何做分库，分表（ 可以按时间（ 月 ） ）

- \* 给出公司业务场景，问你如实现，实现思路是怎么样？

- \* 回顾

- \* 编码，解码，乱码，转换流（ InputStreamReader，OutputStreamWriter ）

- \* 编码表：ASCII（ 128 ） ， ISO-8859-1(256),GB2312(6763) ,GBK(21003),UNICODE(UTF-16,UTF-8)

- \* 打印流：PrintStream，PrintWriter

- \* SequenceInputStream

- \* ObjectInputStream，ObjectOutputStream

- \* GZIPInputStream，GZIPOutputStream

- \* tomcat:

- \* 能够掌握Junit单元测试

- \* 单元测试（ unit testing ） ，是指对软件中的最小可测试单元进行检查和验证。

- \* Junit是一个Java语言的单元测试框架（ 白盒测试 ） 。

- \* Junit单元测试简单使用截图

```

3
4 public class TestXml {
5
6     @Test
7     public void testHelloDom4j() {
8
9     }
10 }
11

```

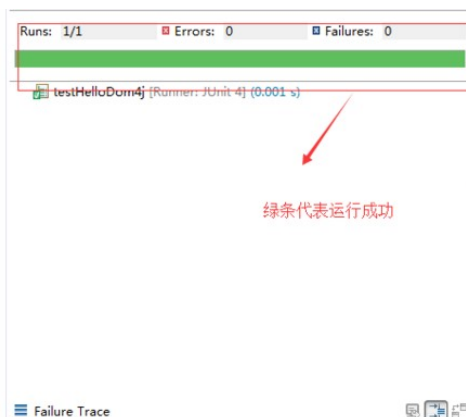
标注一个Test的注解

@Test

Test cannot be resolved to a type

6 quick fixes available:

- 1- Import 'Test' (org.junit)
  - Add JUnit 4 library to the build path
  - Add JUnit 5 library to the build path
  - @ Create annotation 'Test'
  - Change to 'TestedOn' (org.junit.experimental.theories.suppliers)
  - Fix project setup...
- 点击导入JUnit的包



```

3 import org.junit.Test;
4
5 public class TestXml {
6
7     @Test
8     public void testHelloDom4j() {
9         System.out.println("Hello");
10    }
11 }
12

```

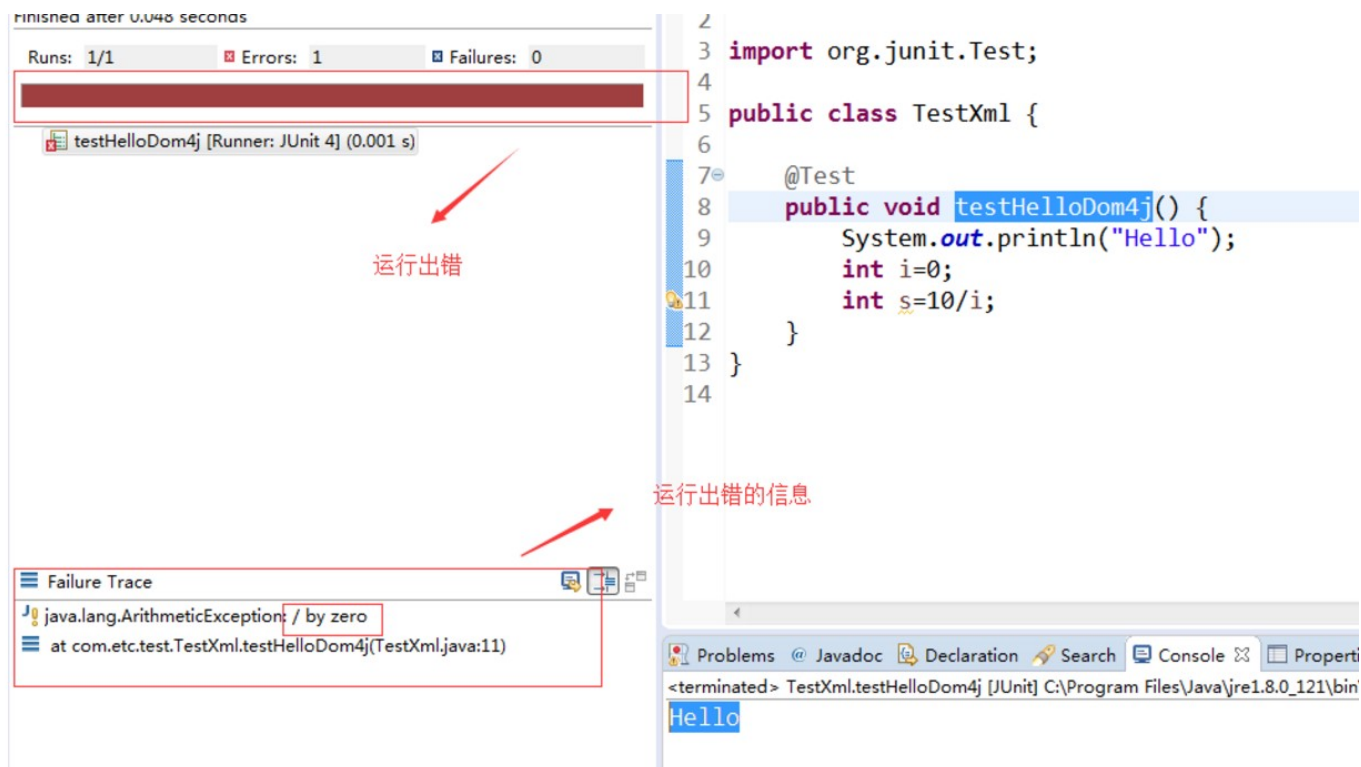
对准这个方法，右键-run-as-->JUnit

控制台输出的结果

```

<terminated> TestXml.testHelloDom4j [JUnit] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (2018年3月26日 下午5:06:26)
Hello

```



```
1 * Junit单元测试
2 * @Test: 测试方法
3 * @Ignore: 被忽略的测试方法: 加上之后, 暂时不运行此段代码
4 * @Before: 每一个测试方法之前运行
5 * @After: 每一个测试方法之后运行
6 * @BeforeClass: 方法必须要是静态方法 (static 声明), 所有测试开始之前运行
7 * @AfterClass: 方法必须要是静态方法 (static 声明), 所有测试结束之后运行
8
9 * 代码
10 public class Test1 {
11
12     @Test
13     public void test1() {
14         System.out.println("HelloWorld");
15     }
16
17     @Test()
18     public void testAdd() {
19         Calculator cal=new Calculator();
20         int sum=cal.add(2, 3);
```

```
21     TestCase.assertEquals(sum, 5);
22 }
23
24 @Before
25 public void testBefore() {
26     System.out.println("before...");
27 }
28
29 @After
30 public void testAfter() {
31     System.out.println("after...");
32 }
33
34 @BeforeClass
35 public static void testStaticBefore() {
36     System.out.println("static before...");
37 }
38
39 @AfterClass
40 public static void testStaticAfter() {
41     System.out.println("static after...");
42 }
43 }
```

45 结果:

46 \* 运行test1()方法

47 static before...

48 before...

49 HelloWorld

50 after...

51 static after...

52

53 \* 运行所有

54 static before...

55 before...

56 after...

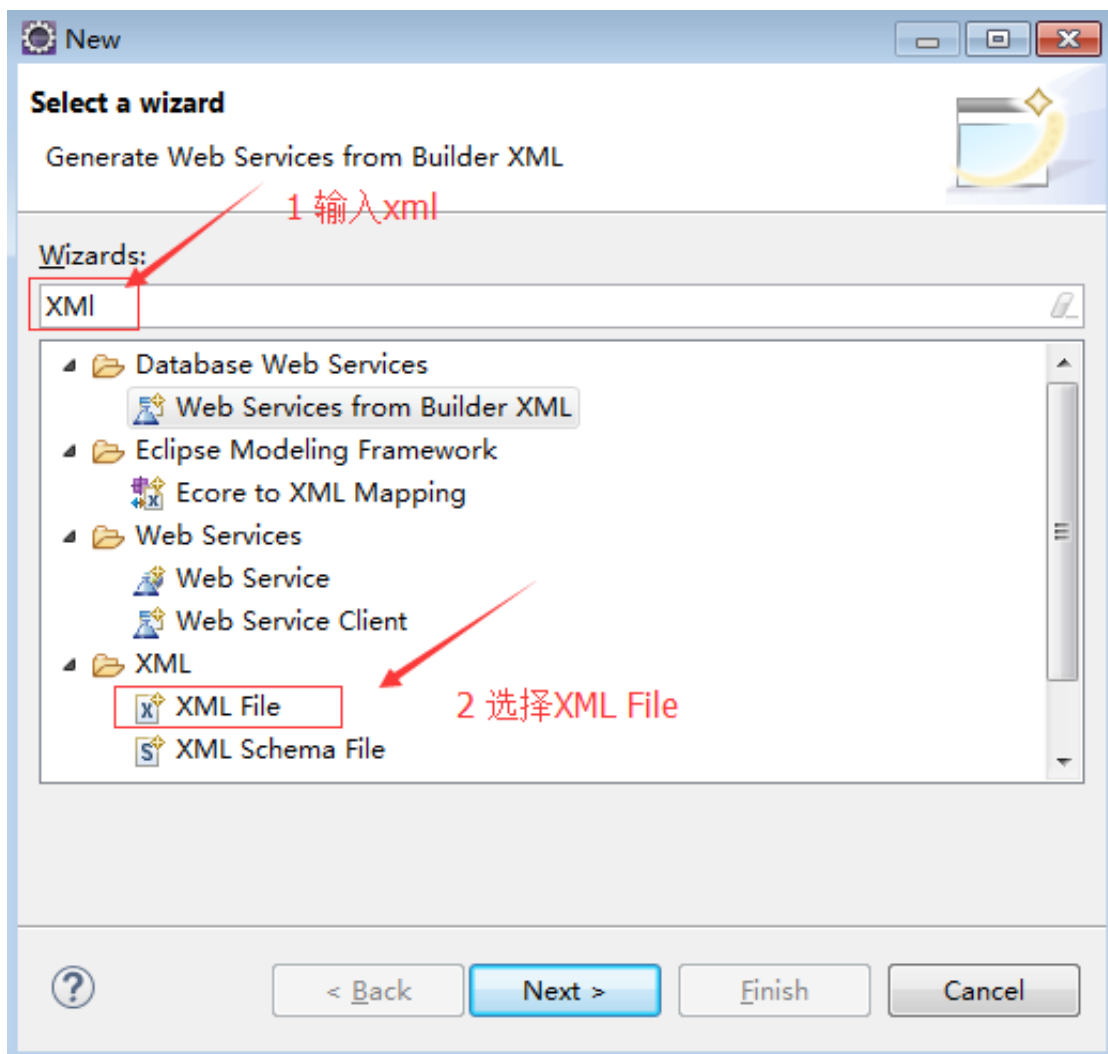
57 before...

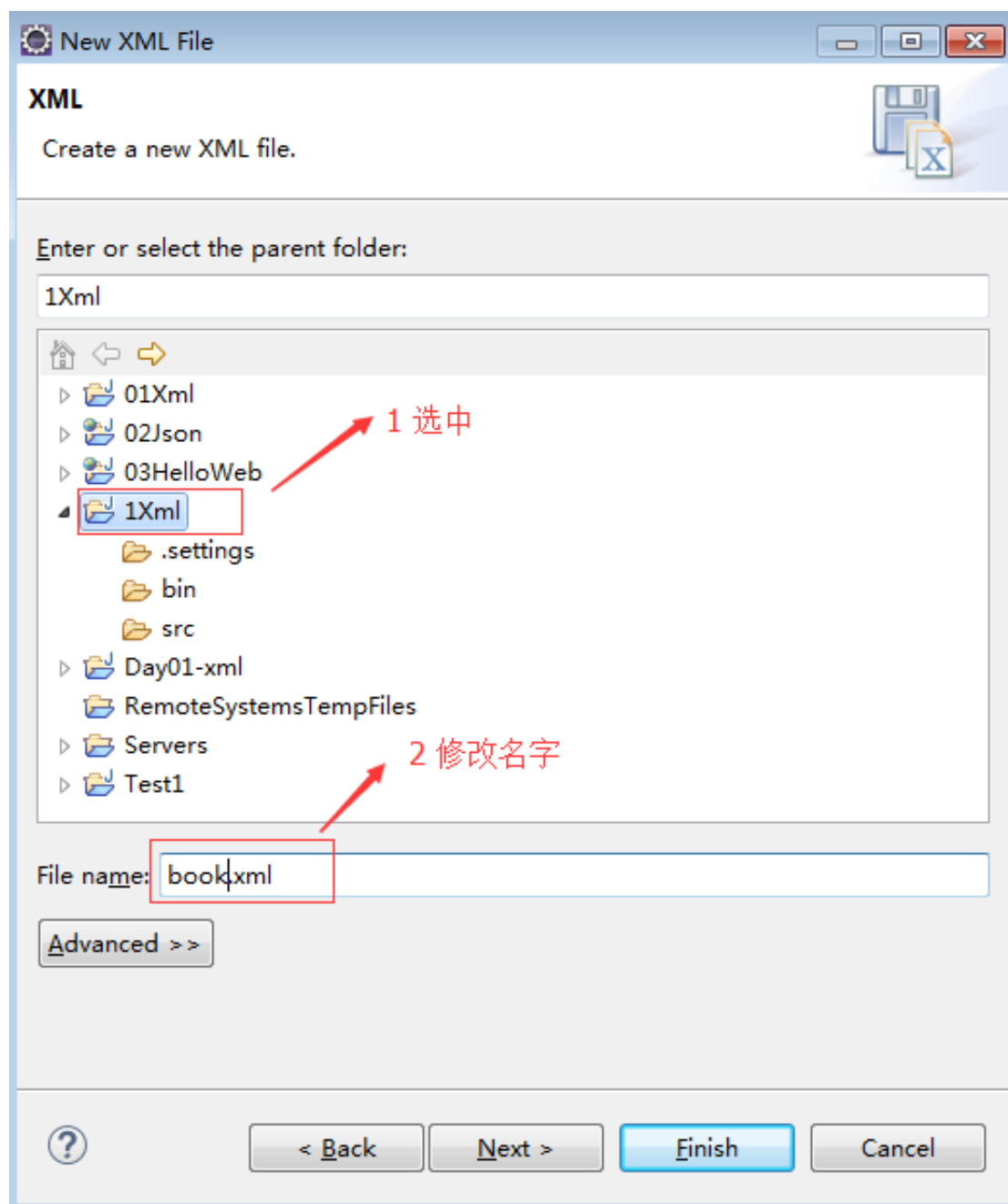
58 HelloWorld

59 after...

60 static after...

- \* 能够掌握XML的概述
  - \* XML 指可扩展标记语言 ( EXtensible Markup Language )
  - \* XML 是一种标记语言，很类似 HTML
  - \* XML 的设计宗旨是传输数据（也可作为配置文件），而非显示数据
  - \* XML 标签没有被预定义。您需要自行定义标签。
  - \* XML 被设计为具有自我描述性。
  - \* XML 是W3C 的推荐标准
- \* 能够掌握XML的HelloWorld开发
  - \* 简单步骤截图





```
1 <?xml version="1.0" encoding="UTF-8"?>
```

版本号

xml的编码

自动生成：也是xml第一句话

```
2 <book>
3   <id>1001</id>
4   <name>Java编程思想</name>
5   <price>88.70</price>
6   <author>Bruce Eckel</author>
7 </book>
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <book>
3     <id>1001</id>
4     <name>Java编程思想</name>
5     <price>88.70</price>
6     <author>Bruce Eckel</author>
7 </book>
8
9 * 温馨提示:
10 * 第一行是 XML 声明。它定义 XML 的版本 (1.0) 和所使用的编码 (UTF-8)
11 * book是根元素
12 * 接下来 4 行描述根的 4 个子元素
13 * 最后一行定义根元素的结尾(</book>)
```

### \* 能够熟悉XML的语法

```
1 * 所有 XML 元素都须有关闭标签
2 * XML 标签对大小写敏感
3 * XML 必须正确地嵌套
4 * XML 文档必须有根元素
5 * XML 的属性值须加引号
6 * 实体引用
7     * 在 XML 中, 有 5 个预定义的实体引用:
8     * &lt;    <    小于
9     * &gt;    >    大于
10    * &amp;   &    和号
11    * &apos;   '    单引号
12    * &quot;   "    引号
13 * XML 中的注释
14 * 在 XML 中, 空格会被保留
15     * 在浏览器里面是看不到效果, 但是解析的时候, 是有空格
```

### \* 语法部分截图



```
<book>
  <id>1001</id>
  <name>Java编程思想</name>
  <price>88.70</price>
  <author>Bruce Eckel</author>
</book>
```

```
<book>
  <Id>1001</id>
  <name>Java编程思想</name>
  <price>88.70</price>
  <author>Bruce Eckel</author>
</book>
```

`<t1><t2></t1></t2>` 错误嵌套

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <book>
3   <id>1001</id>
4   <name>Java编程思想</name>
5   <price>88.70</price>
6   <author>Bruce Eckel</author>
7 </book>
8
9 <book>
10   <id>1001</id>
11   <name>Java编程思想</name>
12   <price>88.70</price>
13   <author>Bruce Eckel</author>
14 </book>
```

文档必须有根元素

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <book id="1002">
3   <name>Java编程思想</name>
4   <price>88.70</price>
5   <author>Bruce Eckel</author>
6 </book>
```

属性必须有双引号

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <book id="1002">
3   <name>Java编程思想</name>
4   <price>88.70</price>
5   <author>Bruce Eckel</author>
6 </book>
7
8

```

小于号，解决方案：实体解决

```

<book id="1002">
  <name>Java编程思想<!--></name>
  <price>88.70</price>
  <author>Bruce Eckel</author>
</book>

```

&lt;相当于<

```

<!-- <author>Bruce Eckel</author> -->

```

\* 能够理解XML元素的概念

\* XML 元素指的是从（且包括）开始标签直到（且包括）结束标签的部分。

\* 元素可包含其他元素、文本或者标签两者的混合物。元素也可以拥有属性。

\* XML 命名规则

\* 名称可以含字母、数字以及其他的字符

\* 名称不能以数字或者标点符号开始

\* 名称不能以字符“xml”（或者 XML、Xml）开始

\* 实践是可以的，但是不建议

\* 名称不能包含空格

\* 可使用任何名称，没有保留的字词。

\* 最佳命名习惯：

\* 名称具有描述性。使用下划线的名称也很不错。

\* 名称应当比较简短，比如：<book\_title>

\* 部分效果截图

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <book id="1002">
3     <name>Java编程思想</name>
4     <price>88.70</price>
5     <author>Bruce Eckel</author>
6 </book>
7

```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <book id="1002">
3     <name>Java编程思想</name>
4     <price>88.70</price>
5     <author>Bruce Eckel</author>
6 </book>
7

```

### \* 能够理解XML属性的概念

- 1 \* XML 元素可以在开始标签中包含属性，类似 HTML。
- 2 \* 属性 (Attribute) 提供关于元素的额外（附加）信息。
- 3 \* 属性vs元素
- 4 \* 没有什么规矩可以告诉我们什么时候该使用属性，而什么时候该使用子元素。
- 5 \* 我的经验是在HTML中，属性用起来很便利，但是在 XML 中，您应该尽量避免使用属性。
- 6 \* 如果信息感觉起来很像数据，那么请使用子元素吧。
- 7 \* 避免XML属性
- 8 \* 属性无法包含多重的值（元素可以）
- 9 \* 属性无法描述树结构（元素可以）
- 10 \* 属性不易扩展（为未来的变化）
- 11 \* 属性难以阅读和维护
- 12
- 13 \* 针对元数据的 XML 属性
- 14 id name age salary
- 15 001 xiaohei 29 9999
- 16 002 xiaohei 30 10000
- 17 \* 在此我们极力向您传递的理念是：元数据（有关数据的数据）应当存储为属性，而数据本身应
- 18

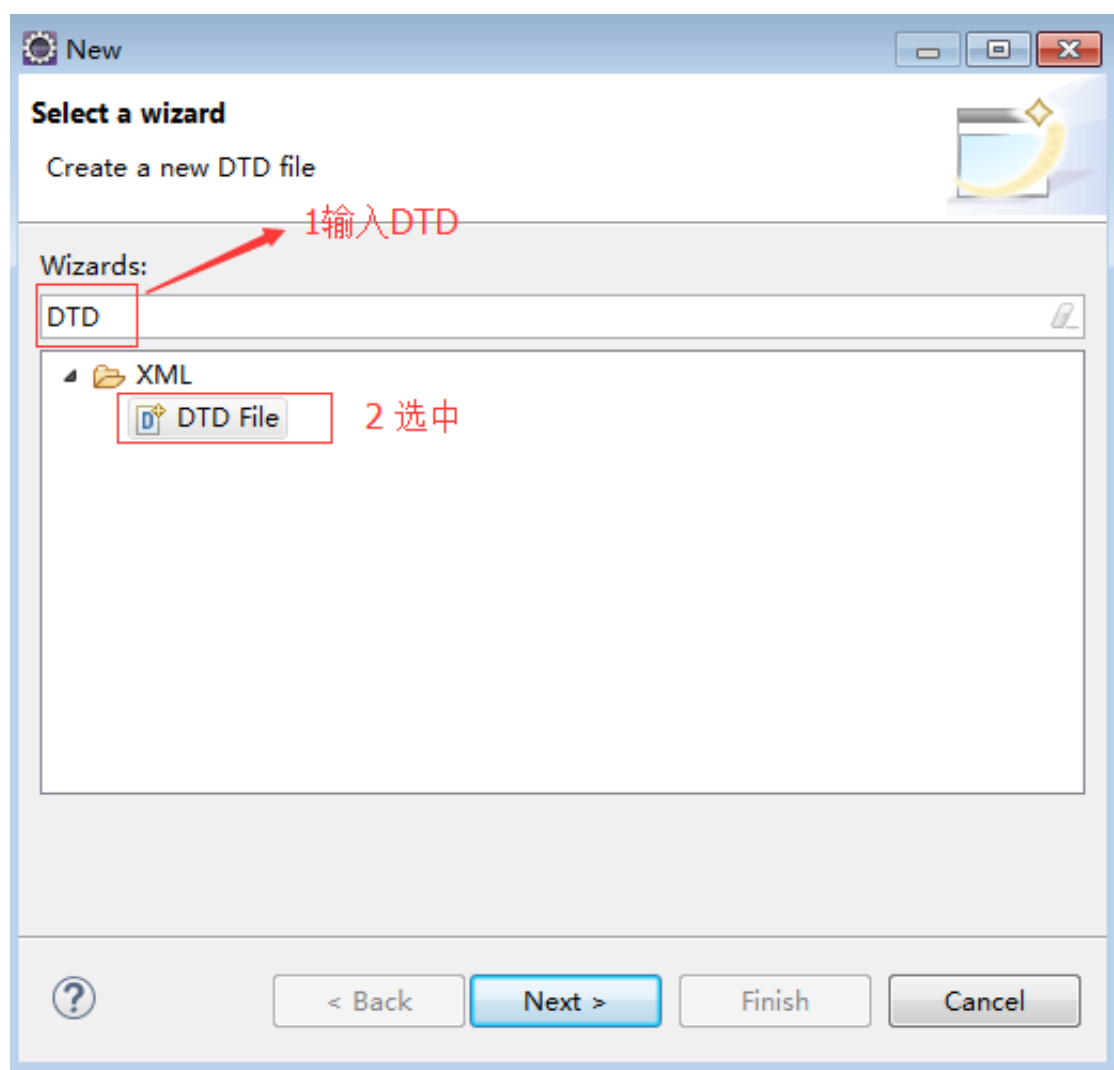
### \* 能够了解XML验证方式之DTD

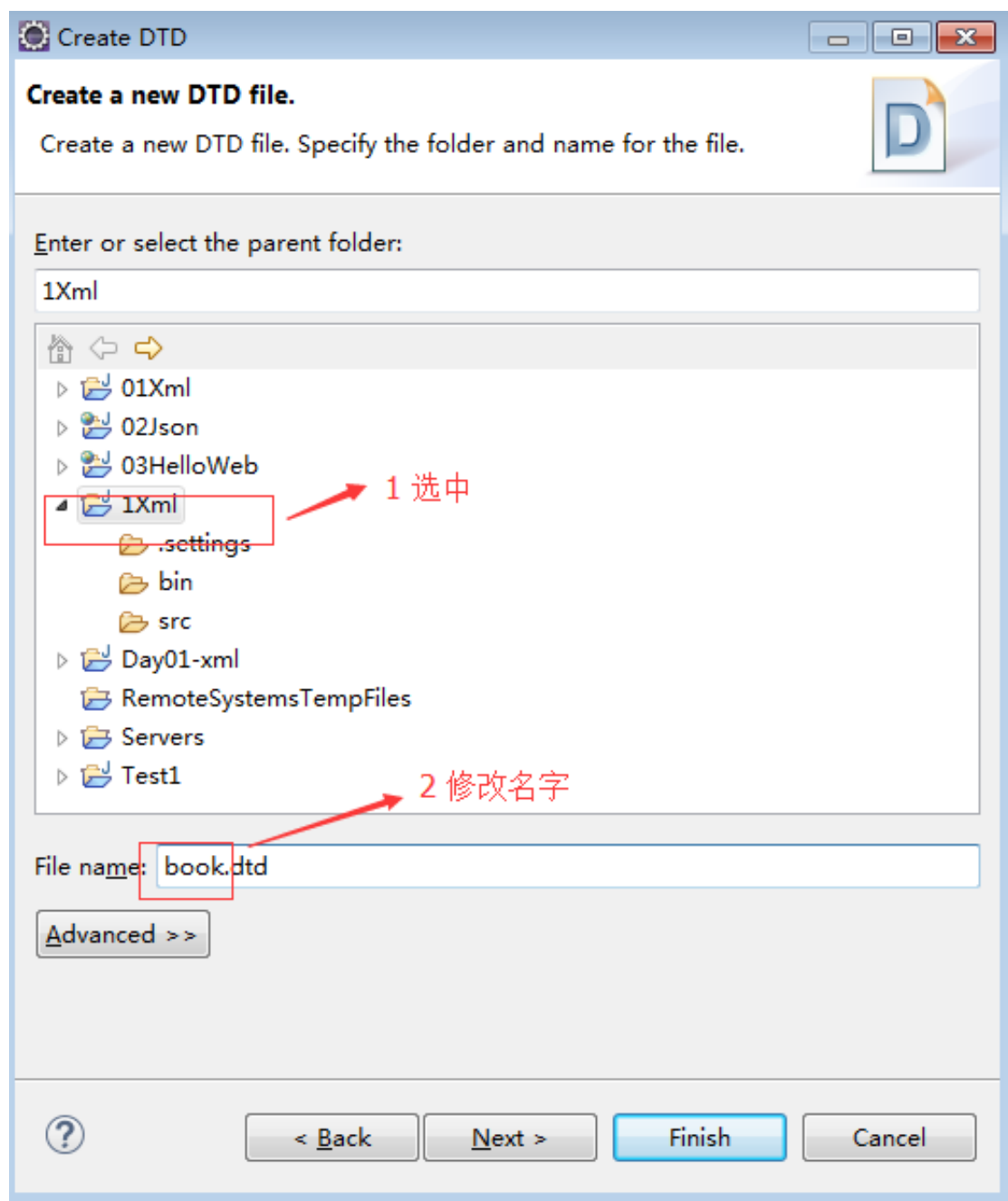
- \* DTD的作用是定义XML文档的结构。它使用一系列合法的元素来定义文档结构。

```
1 * 使用方式一
2 * 直接在xml文件中声明
3 <?xml version="1.0" encoding="UTF-8"?>
4 <!DOCTYPE book [
5     <!ELEMENT book (id,name,price,author)>
6     <!ELEMENT id      (#PCDATA)>
7     <!ELEMENT name     (#PCDATA)>
8     <!ELEMENT price    (#PCDATA)>
9     <!ELEMENT author   (#PCDATA)>
10 ]>
11 <book>
12     <id>1002</id>
13     <name>Java in thinking</name>
14     <price>90</price>
15     <author>xiaobai</author>
16 </book>
17 * 使用方式二
18 * xml导入dtd文件
19 * dtd 文件
20 <?xml version="1.0" encoding="UTF-8"?>
21 <!ELEMENT book (id,name,price,author)>
22 <!ELEMENT id      (#PCDATA)>
23 <!ELEMENT name     (#PCDATA)>
24 <!ELEMENT price    (#PCDATA)>
25 <!ELEMENT author   (#PCDATA)>
26 * 使用
27 <?xml version="1.0" encoding="UTF-8"?>
28 <!DOCTYPE book SYSTEM "book.dtd">
29 <book>
30     <id>1002</id>
31     <name>Java in thinking</name>
32     <price>90</price>
33     <author>xiaobai</author>
34 </book>
35 * 测试检验其合法性
36
37 * 使用方式三
38 * 添加到系统里共享
39 <?xml version="1.0" encoding="UTF-8"?>
```

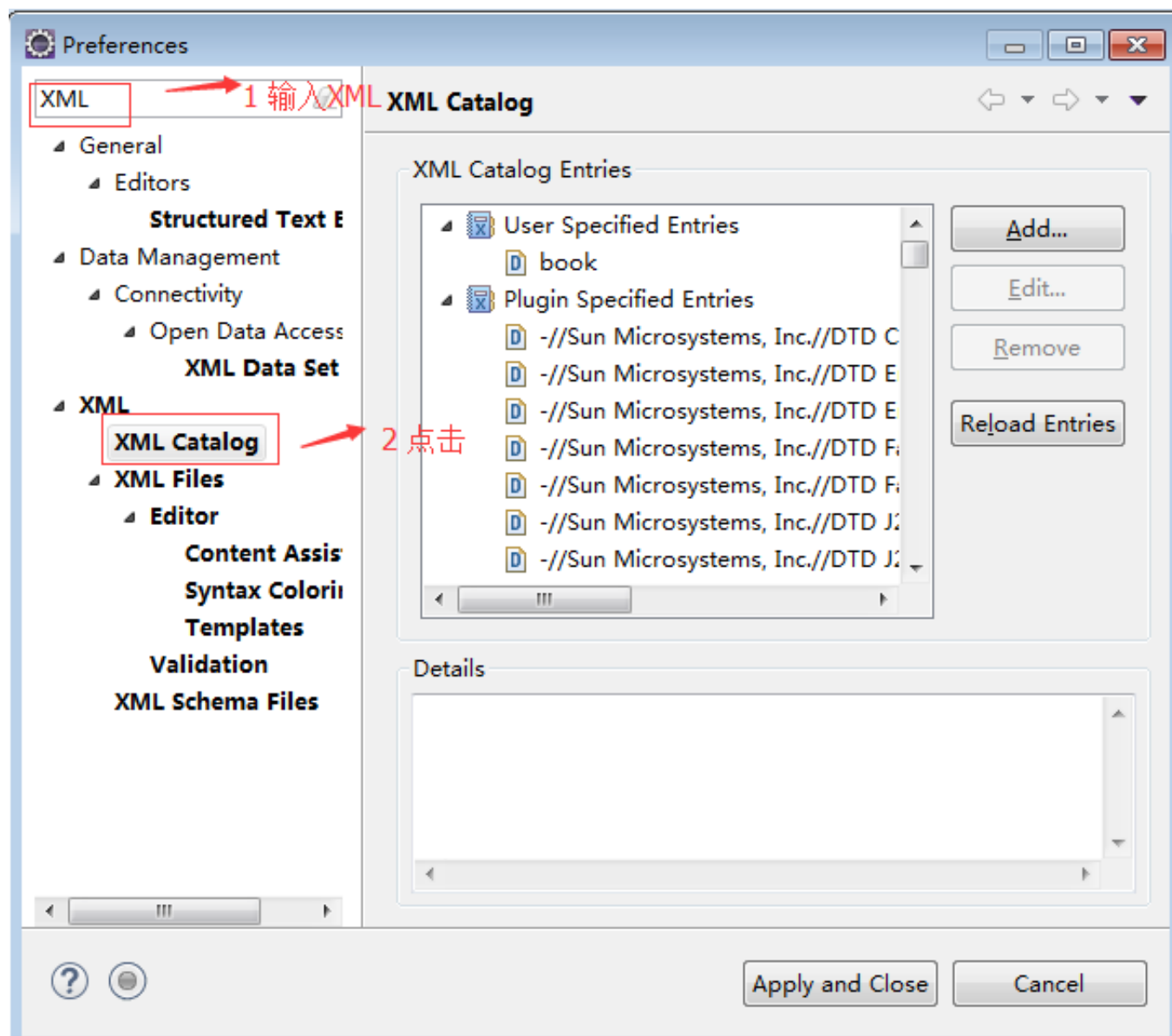
```
40 <!DOCTYPE book PUBLIC "book" "book.dtd">
41 <book>
42     <id>1002</id>
43     <name>Java in thinking</name>
44     <price>90</price>
45     <author>xiaobai</author>
46 </book>
47
```

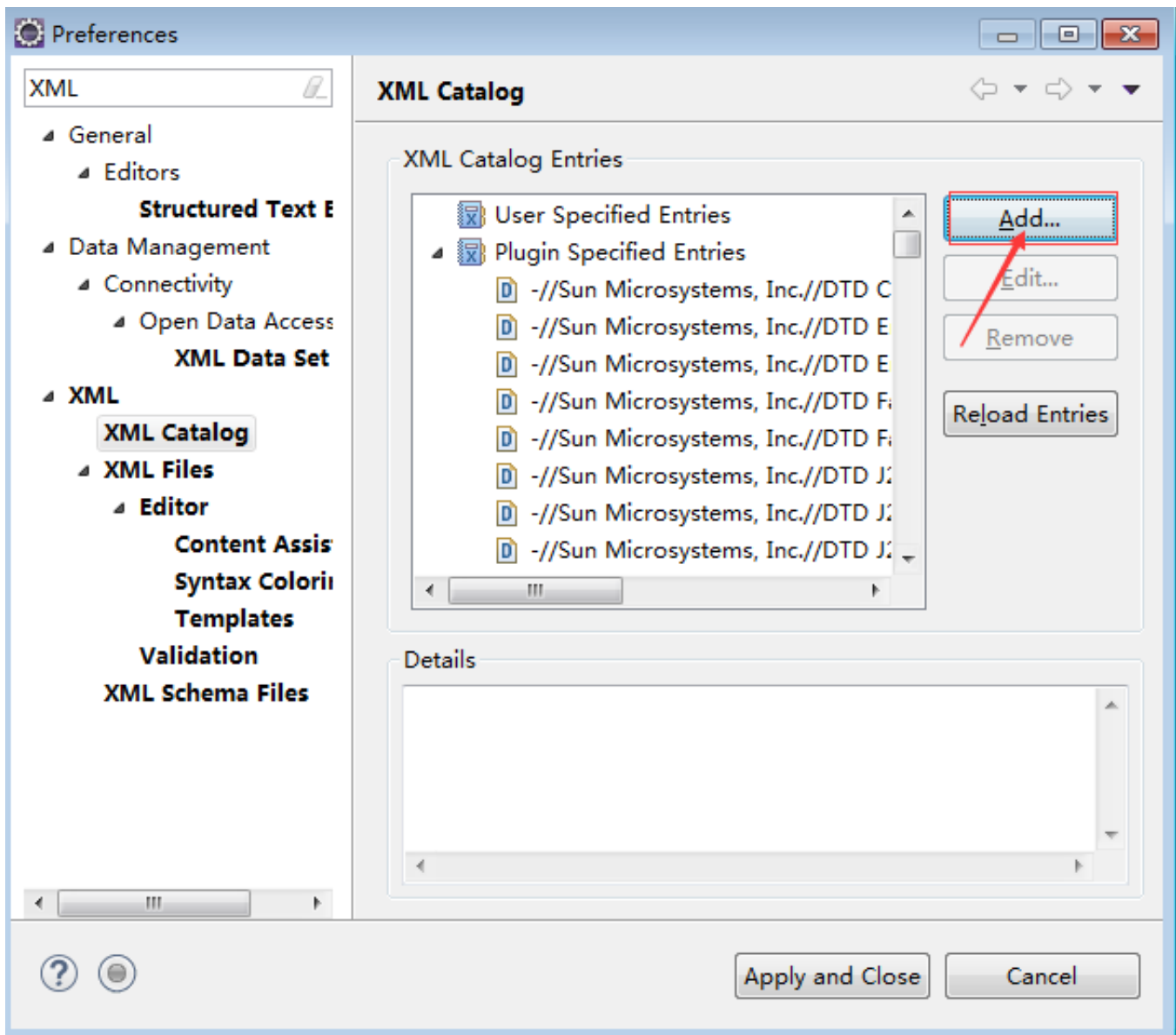
\* 创建DTD截图







\* 引入共享dtd截图











 Add XML Catalog Element ✕

  
Catalog Entry

  
Rewrite Entry

  
Suffix Entry

  
Next Catalog

  
Delegate Catalog

Location:

Workspace... File System...

Key type: Public ID

Key:

☐ Alternative web address:

OK Cancel

Add XML Catalog Element



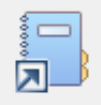
Catalog Entry



Rewrite Entry



Suffix Entry



Next Catalog



Delegate Catalog

Location: 1Xml/book.dtd

Workspace...

File System...

Key type: Public ID

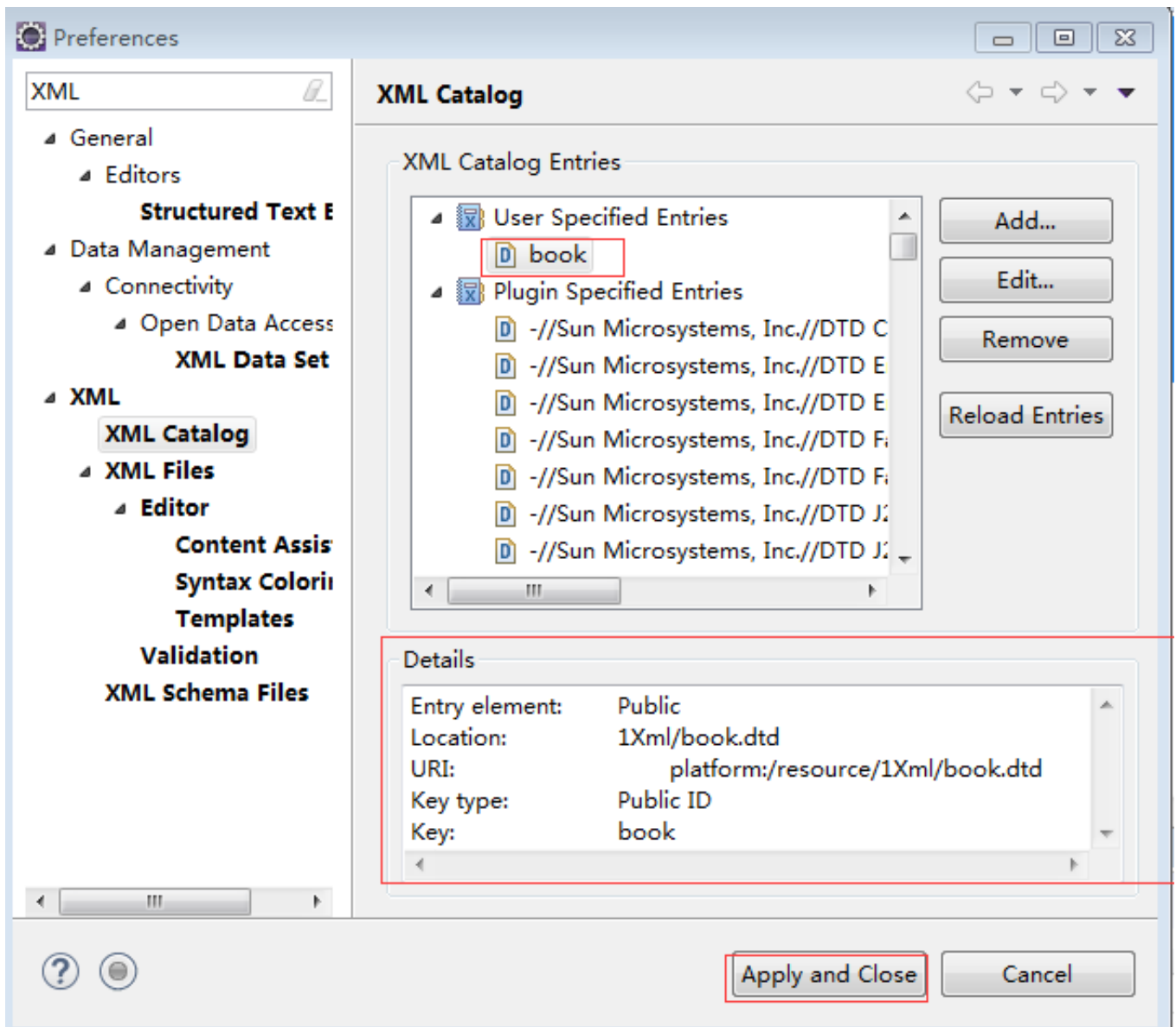
Key: book

☐ Alternative web address:

共享的key值

OK

Cancel



- \* 能够了解XML验证方式之Schema
- \* Schema是一种基于DTD的替代者。
- \* XML Schema 描述 XML 文档的结构。
- \* XML Schema 语言也称作 XML Schema 定义 ( XML Schema Definition , XSD ) 。

```
1 * schema 的文件
2 <?xml version="1.0" encoding="UTF-8"?>
3 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4     targetNamespace="http://www.lg168.org/book"
5     xmlns:tns="http://www.lg168.org/book"
6     elementFormDefault="qualified">
```

```

7      <xs:element name="book">
8          <xs:complexType>
9              <xs:sequence>
10                 <xs:element name="id" type="xs:string" />
11                 <xs:element name="name" type="xs:string" />
12                 <xs:element name="price" type="xs:string" />
13                 <xs:element name="author" type="xs:string" />
14             </xs:sequence>
15         </xs:complexType>
16     </xs:element>
17 </xs:schema>
18
19 * 使用
20 <?xml version="1.0" encoding="UTF-8"?>
21 <book xmlns="http://www.lg168.org/book" xmlns:xsi="http://www.w3.org/2001/XMLSchema
22     xsi:schemaLocation="http://www.lg168.org/book book.xsd">
23     <id>1002</id>
24     <name>Java in thinking</name>
25     <price>90</price>
26     <author>xiaobai</author>
27 </book>
28
29 * 测试校验的效果

```

\* 能够掌握操作XML操作之DOM(DOM4J)

\* 文档对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展标志语言的标准编程接口。

\* 在网页上，组织页面（或文档）的对象被组织在一个树形结构中，用来表示文档中对象的标准模型就称为DOM。

\* DOM解析原理

\* XML解析器一次性把整个xml文档加载进内存，然后在内存中构建一颗Document的对象树，

通过Document对象，得到树上的节点对象，通过节点对象访问（操作）到xml文档的内容。

\* DOM的优点

- \* 整个文档树在内存中，便于操作；
  - \* 可以修改，删除、重新排列XML；
  - \* 可以随机访问任何一个节点，访问效率高
- \* DOM的缺点
- \* 占用内存大，占用资源多(贵)
  - \* 解析速度慢
- \* 适用场景：
- \* 需多次访问这些数据；
  - \* 对解析效率要求不高；
  - \* 硬件资源充足（内存、CPU）
- \* 常见的API
- \* Document:代表整个xml文档
  - \* Node:节点
  - \* Element：标签节点
  - \* Attribute：属性节点
  - \* Text：文本节点

```
1 * 准备bookstore的xml文件
2 <?xml version="1.0" encoding="UTF-8"?>
3 <bookstore>
4     <book id="1001">
5         <name>Java in thinking</name>
6         <price>90</price>
7         <author>xiaobai</author>
8     </book>
9     <book id="1002">
10        <name>Java 从入门到奔溃</name>
11        <price>1</price>
12        <author>无名氏</author>
13    </book>
14 </bookstore>
```

```
15
16 * 开发步骤
17 * 下载:https://dom4j.github.io/
18 * 导入jar包
19 * 编写代码测试
20 @Test
21 public void testHelloDom4j() throws Exception {
22     // 1 获取bookstore.xml 的Document对象
23     SAXReader reader=new SAXReader();
24     Document document = reader.read("./bookstore.xml");
25     // 2 获取Document对象的根节点
26     Element root = document.getRootElement();
27     // 3 打印出根节点标签的名字
28     System.out.println(root.getName());
29     // 4 根节点下所有子元素
30     List<Element> elements = root.elements();
31     // 5 迭代打印出根节点下所有子元素标签的名字
32     for(Element element:elements) {
33         System.out.println(element.getName());
34     }
35 }
36 * 结果:
37     bookstore
38     book
39     book
40
41 // 遍历: 获取所有book的属性和子标签的文本
42 @Test
43 public void test1() throws Exception {
44     // 1 获取Document文档对象
45     SAXReader reader=new SAXReader();
46     Document doc = reader.read("./bookstore.xml");
47     // 2 获取根元素
48     Element root = doc.getRootElement();
49     // 3 从根元素获取所有book元素
50     List<Element> elements = root.elements("book");
51     // 4 迭代book元素的集合
52     for(Element element:elements) {
53         // 5 在迭代中获取book元素下, name, price, author元素的文本元素, 还有id
54         String id=element.attributeValue("id");
```

```

55         String name=element.element("name").getText();
56         String price=element.element("price").getText();
57         String author=element.element("author").getText();
58         System.out.println("id:"+id);
59         System.out.println("name:"+name);
60         System.out.println("price:"+price);
61         System.out.println("author:"+author);
62         System.out.println("-----");
63     }
64 }
65 * 结果:
66 id:1001
67 name:Java in thinking
68 price:90
69 author:xiaobai
70 -----
71 id:1002
72 name:Java 从入门到崩溃
73 price:1
74 author:无名氏
75
76 // 增: 在bookstore添加book标签
77 @Test
78 public void test2() throws Exception {
79     // 1 获取Document对象
80     SAXReader reader=new SAXReader();
81     Document doc = reader.read("./bookstore2.xml");
82     // 2 获取根节点
83     Element root = doc.getRootElement();
84     // 3 向根节点添加book元素
85     Element bookElement = root.addElement("book");
86     // 4 设置book元素id属性
87     bookElement.addAttribute("id", "1003");
88     // 5 给book元素添加name, price,author的元素, 并且设置文本内容
89     bookElement.addElement("name").setText("葫芦娃");
90     bookElement.addElement("price").setText("10$");
91     bookElement.addElement("author").setText("蛇精");
92     // 6 更新文档
93     // 1 构建输出格式(设置缩进格式, 设置添加是否换行)
94     // OutputFormat format=new OutputFormat("\t",true);

```

```

95     OutputFormat format=OutputFormat.createPrettyPrint();// 格式比较好看
96     // 2 构建XmlWriter
97     XMLWriter writer=new XMLWriter(new FileOutputStream("./bookstore2.xml"),
98     // 3 通过XmlWriter写出文档（从内存中写到磁盘）
99     writer.write(doc);
100 }
101
102 结果:
103 <?xml version="1.0" encoding="UTF-8"?>
104 <bookstore>
105     <book id="1001">
106         <name>Java in thinking</name>
107         <price>90</price>
108         <author>xiaobai</author>
109     </book>
110     <book id="1002">
111         <name>Java 从入门到奔溃</name>
112         <price>1</price>
113         <author>无名氏</author>
114     </book>
115     <book id="1003">
116         <name>葫芦娃</name>
117         <price>10$</price>
118         <author>蛇精</author>
119     </book>
120 </bookstore>
121
122 //删除第三个book标签(id="1003")
123 @Test
124 public void test4() throws Exception {
125     // 1 获取Document对象
126     SAXReader reader=new SAXReader();
127     Document doc = reader.read("./bookstore2.xml");
128     // 2 获取根节点
129     Element root = doc.getRootElement();
130     // 3 获取book的所有标签
131     List<Element> elements = root.elements("book");
132     // 4 迭代根标签
133     // 5 在迭代中判断id是否等于1003, 假如是的话, 修改价格
134     for(Element element:elements) {

```



```

135     String id = element.attributeValue("id");
136     if("1003".equals(id)) {
137         // 删除自己：自杀的权利
138         element.getParent().remove(element);
139         // 6 更新文档
140         // 1 构建输出格式(设置缩进格式，设置添加是否换行)
141         OutputFormat format=OutputFormat.createPrettyPrint();// 格式比较
142         // 2 构建XmlWriter
143         XMLWriter writer=new XMLWriter(new FileOutputStream("./bookstore2.xml"));
144         // 3 通过XmlWriter写出文档（从内存中写到磁盘）
145         writer.write(doc);
146     }
147 }
148
149 }
150

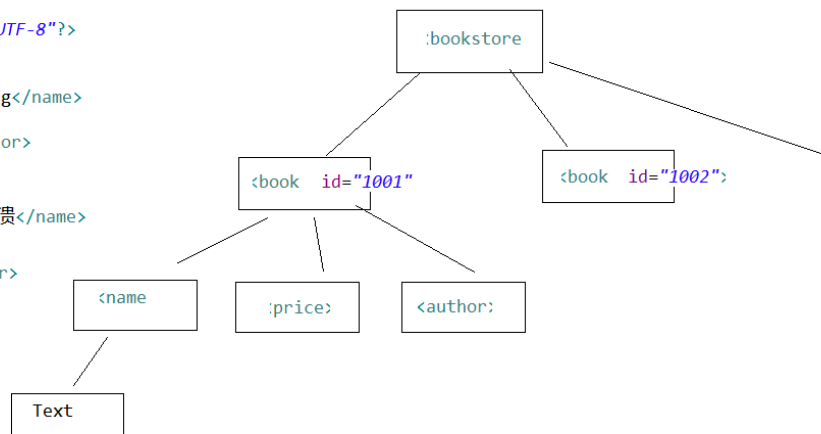
```

## \* 分析bookstore.xml截图

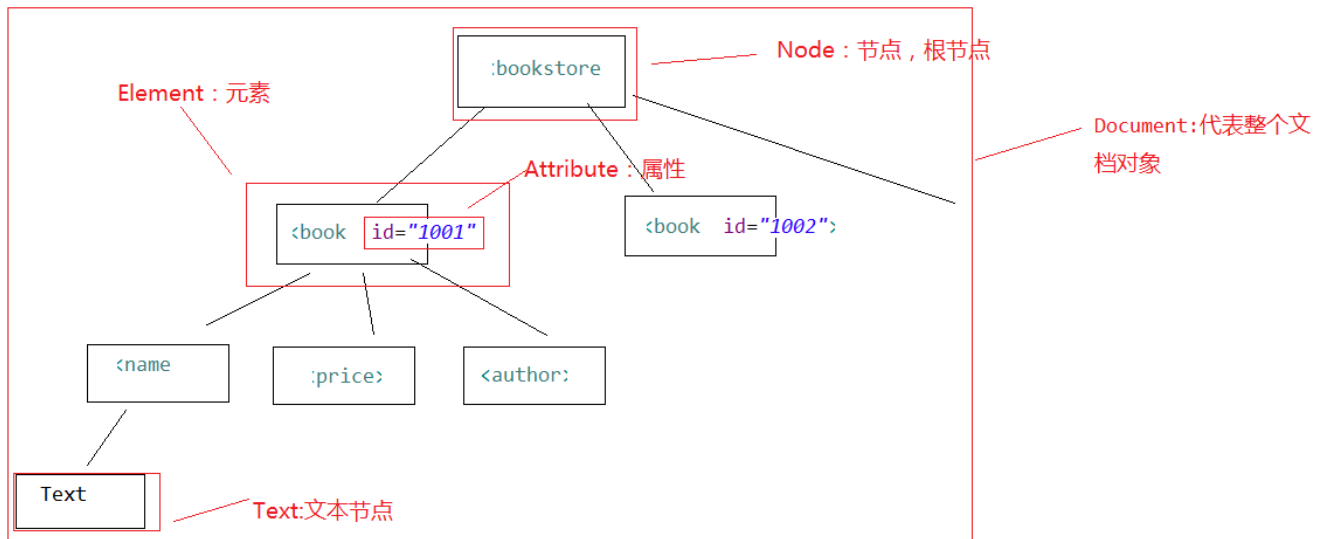
```

<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book id="1001">
    <name>Java in thinking</name>
    <price>90</price>
    <author>xiaobai</author>
  </book>
  <book id="1002">
    <name>Java 从入门到崩溃</name>
    <price>1</price>
    <author>无名氏</author>
  </book>
</bookstore>

```



## \* API解释图



\* 能够掌握解析XML操作之SAX

\* 能够了解解析XML操作之Pull