物联网中间件

第五章 WEB 基础——HTML、XML、JSON

参考资料

- https://www.runoob.com/xml/xml-tutorial.html
- http://www.ecma-international.org/publications/standards/Ecma-404.htm

W3C

- W3C 指万维网联盟（*World Wide Web Consortium*）
- W3C 创建于*1994年10月*
- W3C 由 *Tim Berners-Lee* 创建
- W3C 是一个*会员组织*
- W3C 的工作是*对 web 进行标准化*
- W3C 创建并维护 *WWW 标准*
- W3C 标准被称为 *W3C 推荐（W3C 规范）*

W3C 是如何创建的?

- 万维网（World Wide Web）是作为欧洲核子研究组织的一个项目发展起来的，在那里 Tim Berners-Lee 开发出万维网的雏形。
- Tim Berners-Lee - 万维网的发明人 - 目前是万维网联盟的主任。
- W3C 在 1994 年被创建的目的是，为了完成麻省理工学院（MIT）与欧洲粒子物理研究所（CERN）之间的协同工作，并得到了美国国防部高级研究计划局（DARPA）和欧洲委员会（European Commission）的支持。

标准化 web

- W3C 致力于实现所有的用户都能够对 web 加以利用（不论其文化教育背景、能力、财力以及其身体残疾）。
- W3C 同时与其他标准化组织协同工作，比如 Internet 工程工作小组（Internet Engineering Task Force）、无线应用协议（WAP）以及 Unicode 联盟（Unicode Consortium）。
- W3C 由美国麻省理工学院计算机科学和人工智能实验室 (MIT CSAIL)，总部位于法国的欧洲信息数学研究联盟(ERCIM) 和日本的庆应大学（Keio University）联合运作，并且在世界范围内拥有分支办事处。

W3C 成员

- 正因为 Web 是如此的重要（不论在其影响范围还是在投资方面），以至于不应由任何一家单独的组织来对它的未来进行控制，因此 W3C 扮演着一个会员组织的角色：
- W3C 的会员包括了：软件开发商、内容提供商、企业用户、通信公司、研究机构、研究实验室、标准化团体以及政府。

W3C 规范的批准步骤

- 在 W3C 发布某个新标准的过程中，规范是通过下面的严格程序由一个简单的理念逐步确立为推荐标准的：
- W3C 收到一份提交
- 由 W3C 发布一份记录
- 由 W3C 创建一个工作组
- 由 W3C 发布一份工作草案
- 由 W3C 发布一份候选的推荐
- 由 W3C 发布一份被提议的推荐
- 由 W3C 发布推荐

HTML

- HTML 指的是超文本标记语言: **H**yper**T**ext **M**arkup **L**anguage
- HTML 不是一种编程语言，而是一种**标记**语言
- 标记语言是一套**标记标签** (markup tag)
- HTML 使用标记标签来**描述**网页
- HTML 文档包含了HTML **标签**及**文本**内容
- HTML文档也叫做 **web 页面**

HTML 标签

- HTML 标记标签通常被称为 HTML 标签 (HTML tag)。
- HTML 标签是由*尖括号*包围的关键词，比如 <html>
- HTML 标签通常是*成对出现*的，比如 和
  - 标签对中的第一个标签是*开始标签*，第二个标签是*结束标签*
  - 开始和结束标签也被称为*开放标签*和*闭合标签*

- 例子: <标签>内容</标签>

HTML例子

```
<html>
    <head>
        <title>□页面标题</title>
    </head>
    <body>
        <h1>□这是一个标题</h1>
        <p>这是一个段落。</p>
        <p>□这是另外一个段落。</p>
    </body>
</html>
```

XML

- XML 指可扩展标记语言（EXtensible Markup Language）。
- XML 是一种很像HTML的标记语言。
- XML 的设计宗旨是传输数据，而不是显示数据。
- 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 是独立于软件和硬件的信息传输工具。**
- XML 于 1998 年 2 月 10 日成为 W3C 的推荐标准
- XML 是各种应用程序之间进行数据传输的最常用的工具。（目前：JSON）

XML 和 HTML 之间的差异

- XML 不是 HTML 的替代。

- XML 和 HTML 为不同的目的而设计：
  - XML 被设计用来传输和存储数据，其焦点是数据的内容。
  - HTML 被设计用来显示数据，其焦点是数据的外观。

- HTML 旨在显示信息，而 XML 旨在传输信息。

XML 用途

- **XML 简化数据传输**
  - 对开发人员来说，其中一项最费时的挑战一直是在互联网上的不兼容系统之间交换数据。
  - 由于可以通过各种不兼容的应用程序来读取数据，以 XML 交换数据降低了这种复杂性。
- **XML 简化数据共享**
  - 在真实的世界中，计算机系统和数据使用不兼容的格式来存储数据。
  - XML 数据以纯文本格式进行存储，因此提供了一种独立于软件和硬件的数据存储方法。
  - 这让创建不同应用程序可以共享的数据变得更加容易。

其他应用场景

- 配置文件
  - 开源软件很多配置文件
- UI描述
  - HTML和Android布局文件

XML 树结构

- XML 文档必须包含**根元素**。该元素是所有其他元素的父元素。

- XML 文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端。

XML 树结构

- 所有的元素都可以有子元素:

```
<root>
    <child>
        <subchild>.....</subchild>
    </child>
</root>
```

- 父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。
- 所有的元素都可以有文本内容和属性（类似 HTML 中）。

元素

- 元素是 XML 以及 HTML 文档的**主要构建模块**。
- 元素可包含文本、其他元素或者是空的。

```
<body>some text</body>
<message>some text</message>
```

属性

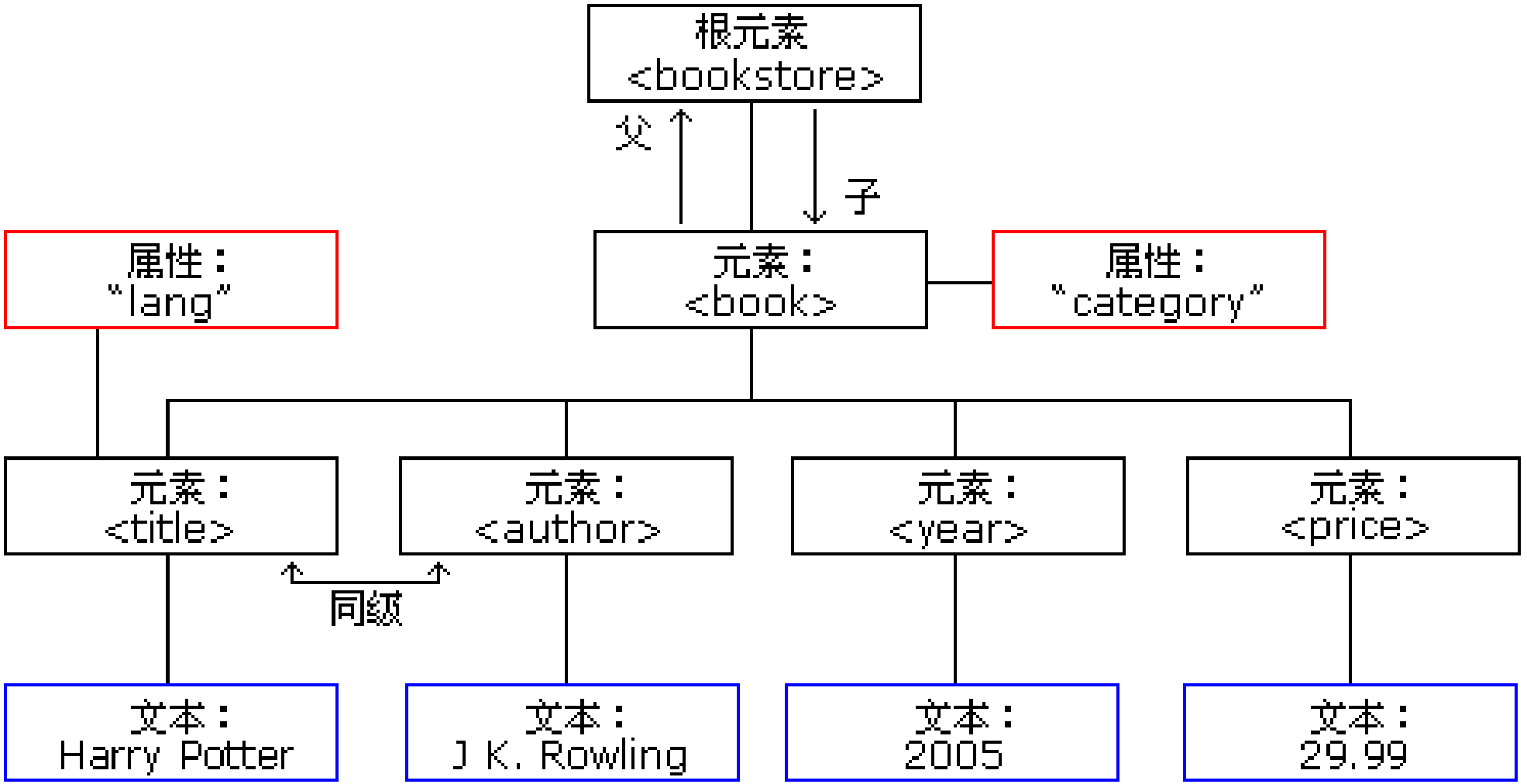
- 属性可提供**有关元素的额外信息**。
- 属性总是被置于某元素的开始标签中。属性总是以**名称/值**的形式成对出现的。下面的 "img" 元素拥有关于源文件的额外信息：

```
<img src="computer.gif" />
```

- 元素的名称是 "img"。属性的名称是 "src"。属性的值是 "computer.gif"。由于元素本身为空，它被一个 " /" 关闭。

```xml
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

- 实例中的根元素是 <bookstore>。文档中的所有 <book> 元素都被包含在 <bookstore> 中。
- <book> 元素有 4 个子元素：<title>、<author>、<year>、<price>。


根元素
<bookstore>
父
子
属性：
"lang"
元素：
<book>
属性：
"category"
元素：
<title>
元素：
<author>
元素：
<year>
元素：
<price>
同级
文本：
Harry Potter
文本：
J K. Rowling
文本：
2005
文本：
29.99

XML语法——XML 声明

- XML 声明文件的可选部分，如果存在需要放在文档的第一行，如下所示：

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

- 以上实例包含 XML 版本（UTF-8 也是 HTML5, CSS, JavaScript, PHP, 和 SQL 的默认编码。）

XML语法——所有的 XML 元素都必须有一个关闭标签

- 在 XML 中，省略关闭标签是非法的。所有元素都**必须**有关闭标签：

```
<p>This is a paragraph.</p>
```

XML语法—— XML 标签（标记）

- XML 标签对大小写敏感。标签 <Letter> 与标签 <letter> 是不同的。
- 必须使用相同的大小写来编写打开标签和关闭标签：

```
<Message>这是错误的</message>

<message>这是正确的</message>
```

- **注释：**打开标签和关闭标签通常被称为开始标签和结束标签。不论您喜欢哪种术语，它们的概念都是相同的。

XML 必须正确嵌套

```
<b><i>This text is bold and italic</i></b>
```

- 在上面的实例中，正确嵌套的意思是：由于 <i> 元素是在 <b> 元素内打开的，那么它必须在 <b> 元素内关闭。

```
<b><i>This text is bold and italic</b></i>
```
错误

XML语法—— XML 属性值必须加引号

- 与 HTML 类似，XML 元素也可拥有属性（名称/值的对）。
- 在 XML 中，XML 的属性值必须加引号。
- 请研究下面的两个 XML 文档。 左边是错误的，右边是正确的:

```
<note date=12/11/2007>
        <to>Tove</to>
        <from>Jani</from>
</note>
```

```
<note date="12/11/2007">
    <to>Tove</to>
    <from>Jani</from>
</note>
```

XML语法——实体引用

- 在 XML 中，一些字符拥有特殊的意义。
- 如果您把字符 "<" 放在 XML 元素中，会发生错误，这是因为解析器会把它当作新元素的开始。
- 这样会产生 XML 错误：`<message>if salary < 1000 then</message>`
- 为了避免这个错误，请用**实体引用**来代替 "<" 字符：

```
<message>if salary &lt; 1000 then</message>
```

- 在 XML 中，有 5 个预定义的实体引用：

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

XML语法—— XML 中的注释

- *<!-- 开头*

```
<!--This is a comment-->
```

XML 元素

- 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。

- 一个元素可以包含:
  - 其他元素
  - 文本
  - 属性
  - 或混合以上所有...

```
<bookstore>
    <book category="CHILDREN">
        <title>Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title>Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

在上面的实例中，<bookstore> 和 <book> 都有**元素内容**，因为他们包含其他元素。<book> 元素也有**属性**（category="CHILDREN"）。<title>、<author>、<year> 和 <price> 有**文本内容**

XML 命名规则

- XML 元素必须遵循以下命名规则:
  - 名称可以包含字母、数字以及其他的字符
  - 名称不能以数字或者标点符号开始
  - 名称不能以字母 xml（或者 XML、Xml 等等）开始
  - 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

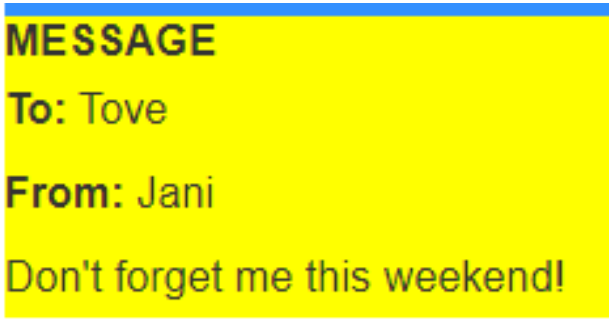
- 使名称具有描述性。使用下划线的名称也很不错：<first_name>、<last_name>。
- 名称应简短和简单，比如：<book_title>，而不是：<the_title_of_the_book>。
- 避免 "-" 字符。如果您按照这样的方式进行命名："first-name"，一些软件会认为您想要从 first 里边减去 name。
- 避免 "." 字符。如果您按照这样的方式进行命名："first.name"，一些软件会认为 "name" 是对象 "first" 的属性。
- 避免 ":" 字符。冒号会被转换为命名空间来使用（稍后介绍）。

XML 元素是可扩展的

- XML 元素是可扩展，以携带更多的信息。
- 请看下面的 XML 实例：

```
<note>
    <to>Tove</to>
    <from>Jani</from>
    <body>Don't forget me this weekend!</body>
</note>
```

- 让我们设想一下，我们创建了一个应用程序，可将 <to>、<from> 以及 <body> 元素从 XML 文档中提取出来，并产生以下的输出：



- 想象一下，XML 文档的作者添加的一些额外信息：

```
<note>
    <date>2008-01-10</date>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

- 应用程序并没有进行相应的修改，那么这个应用程序会中断或崩溃吗？
- 不会。这个应用程序仍然可以找到 XML 文档中的 <to>、<from> 以及 <body> 元素，并产生同样的输出。
- XML 的优势之一，就是可以在不中断应用程序的情况下进行扩展

XML 属性

- 属性（Attribute）提供有关元素的额外信息。

- 属性值必须被引号包围，不过单引号和双引号均可使用。比如一个人的性别，person 元素可以这样写：

```
<person gender="female">
```

- 或者这样也可以：

```
<person gender='female'>
```

- 如果属性值本身包含双引号，您可以使用单引号，就像这个实例：

```
<gangster name='George "Shotgun" Ziegler'>
```

- 或者您可以使用字符实体：

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML 元素 vs. 属性

```
<person gender="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

```
<person>
      <gender>female</gender>
      <firstname>Anna</firstname>
      <lastname>Smith</lastname>
</person>
```

- 在左边实例中，gender 是一个属性。在右边实例中，gender 是一个元素。这两个实例都提供相同的信息。
- 没有什么规矩可以告诉我们什么时候该使用属性，而什么时候该使用元素。实际经验是在 XML 中，您应该尽量避免使用属性。如果信息感觉起来很像数据，那么请使用元素吧。

例子

- 下面的三个 XML 文档包含完全相同的信息:

- 第一个实例中使用了 date 属性:

```
<note date="10/01/2008">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

- 第二个实例中使用了 date 元素：

```
<note>
        <date>10/01/2008</date>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

- 第三个实例中使用了扩展的 date 元素：

```
<note>
        <date>
                <day>10</day>
                <month>01</month>
                <year>2008</year>
        </date>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

避免 XML 属性?

- 因使用属性而引起的一些问题:
  - 属性不能包含多个值（元素可以）
  - 属性不能包含树结构（元素可以）
  - 属性不容易扩展（为未来的变化）

- 属性难以阅读和维护。请尽量使用元素来描述数据。而仅仅使用属性来提供与数据无关的信息。

- 不要做这样的蠢事（这不是 XML 应该被使用的方式）：

```
<note day="10" month="01" year="2008"
      to="Tove" from="Jani" heading="Reminder"
      body="Don't forget me this weekend!">
</note>
```

针对元数据的 XML 属性

- 有时候会向元素分配 ID 引用。这些 ID 索引可用于标识 XML 元素
- id 属性仅仅是一个标识符，用于标识不同的便签。它并不是便签数据的组成部分。
- 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，而数据本身应当存储为元素。

```
<messages>
    <note id="501">
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
    </note>
    <note id="502">
        <to>Jani</to>
        <from>Tove</from>
        <heading>Re: Reminder</heading>
        <body>I will not</body>
    </note>
</messages>
```

XML 命名空间

- XML 命名空间提供避免元素命名冲突的方法。

- 命名冲突
  - 在 XML 中，元素名称是由开发者定义的，当两个不同的文档使用相同的元素名时，就会发生命名冲突。

- 这个 XML 携带 HTML 表格的信息：

```
<table>
    <tr>
        <td>Apples</td>
        <td>Bananas</td>
    </tr>
</table>
```

- 这个 XML 文档携带有关桌子的信息（一件家具）：

```
<table>
        <name>African Coffee Table</name>
        <width>80</width>
        <length>120</length>
</table>
```

- 假如这两个 XML 文档被一起使用，由于两个文档都包含带有不同内容和定义的 <table> 元素，就会发生命名冲突。

- XML 解析器无法确定如何处理这类冲突。

使用前缀来避免命名冲突

- 在 XML 中的命名冲突可以通过使用名称前缀从而容易地避免。
- 该 XML 携带某个 HTML 表格和某件家具的信息：

```
<h:table>
    <h:tr>
        <h:td>Apples</h:td>
        <h:td>Bananas</h:td>
    </h:tr>
</h:table>

<f:table>
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
```

XML 命名空间 - xmlns 属性

- 当在 XML 中使用前缀时，一个所谓的用于前缀的**命名空间**必须被定义。

- 命名空间是在元素的开始标签的 **xmlns 属性**中定义的。

- 命名空间声明的语法如下。xmlns:*前缀*="*URI*"。

```
<root>

    <h:table xmlns:h="http://www.w3.org/TR/html4/">
        <h:tr>
            <h:td>Apples</h:td>
            <h:td>Bananas</h:td>
        </h:tr>
    </h:table>

    <f:table xmlns:f="http://www.w3cschool.cc/furniture">
        <f:name>African Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
    </f:table>

</root>
```

- 在上面的实例中，<table> 标签的 xmlns 属性定义了 h: 和 f: 前缀的合格命名空间。

- 当命名空间被定义在元素的开始标签中时，所有带有相同前缀的子元素都会与同一个命名空间相关联。

- 命名空间，可以在他们被使用的元素中或者在 XML 根元素中声明:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="http://www.w3cschool.cc/furniture">

    <h:table>
        <h:tr>
            <h:td>Apples</h:td>
            <h:td>Bananas</h:td>
        </h:tr>
    </h:table>

    <f:table>
        <f:name>African Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
    </f:table>

</root>
```

- 命名空间 URI 不会被解析器用于查找信息。其目的是赋予命名空间一个惟一的名称

默认的命名空间

- 为元素定义默认的命名空间可以让我们省去在所有的子元素中使用前缀的工作

```
xmlns="namespaceURI"
```

```
<table xmlns="http://www.w3.org/TR/html4/">
    <tr>
        <td>Apples</td>
        <td>Bananas</td>
    </tr>
</table>
```

默认的命名空间

- 那么这种没有指定前缀的命名空间就会作为页面中元素的默认命名空间，除非在标签中使用其他命名空间的前缀，否则解析器都会认为元素是在默认命名空间下存在。
- 一个文档中只能有一个默认的命名空间，如下的语法是错误的:
-
```
<book xmlns="http://www.atguigu.com/xml/b"
xmlns="http://www.atguigu.com/xml/a">
```

- 这里我们指定了两个命名空间而都没有使用前缀，解析器在解析文档时会不知道使用哪个命名空间，所以在一个文档中只能有一个默认的命名空间，其他命名空间必须使用前缀

- 根据以下数据回答下列问题：“小明今年22岁，来自安徽大学。兴趣是看电影和旅游。他有两个姐姐，一个叫小芬，今年25岁，职业是护士。还有一个叫小芳，今年23岁，是一名小学老师”
- 用XML语法描述以上数据

```xml
<?xml version="1.0" encoding="utf-8"?>
<person
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="person.xsd">
    <name>小明</name>
    <age>22</age>
    <university>安徽大学</university>
    <interest>电影</interest>
    <interest>旅游</interest>
    <sister>
        <name>小芬</name>
        <age>25</age>
        <job>护士</job>
    </sister>
    <sister>
        <name>小芳</name>
        <age>23</age>
        <job>小学老师</job>
    </sister>
</person>
```

```
<person>
    <name>小明</name>
    <age>22</age>
    <university>安徽大学</university>
    <hobbies>
        <hobby>电影</hobby>
        <hobby>旅游</hobby>
    </hobbies>
    <sisters>
        <sister>
            <name>小芬</name>
            <age>25</age>
            <job>护士</job>
        </sister>
        <sister>
            <name>小芳</name>
            <age>23</age>
            <job>小学老师</job>
        </sister>
    </sisters>
</person>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person>
    <小明>
        <age>22</age>
        <school="安徽大学"/>
        <个人爱好="电影旅游" 姐姐="小芬小芳"/>
        <小芬>
        <age>25</age>
        <job="护士"/>
        <小芳>
        <age>23</age>
        <job="小学老师"/>
    </小明>
</person>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person>
    <小明 school="安徽大学"个人爱好="电影旅游" 姐姐="小芬小芳"> >
        <age>22</age>
        <小芬 job="护士">
        <age>25</age>
        </小芬>
        <小芳 job="小学老师">
        <age>23</age>
       </小芳>
    </小明>
</person>
```

关于XML，下面哪些描述是错误的:

- A)　每个合格的XML都有唯一的根元素
- B)　XML和Java,C/C++一样是门编程语言
- C）XML的格式上是要求严格的，每个元素的开闭必须完整，不允许交叉开闭
- D)　XML常用于WebService中用来做数据交换的标准
- E）XML中元素是大小写敏感的

文档类型定义（DTD）

- 文档类型定义（DTD，Document Type Definition）可定义合法的XML文档构建模块

- DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。

为什么使用 DTD?

- 通过 DTD，您的每一个 XML 文件均可携带一个有关其自身格式的描述。
- 通过 DTD，独立的团体可一致地使用某个标准的 DTD 来交换数据。
- 而您的应用程序也可使用某个标准的 DTD 来验证从外部接收到的数据。
- 您还可以使用 DTD 来验证您自身的数据。
- DTD是一种保证XML文档格式正确的有效方法，可以通过比较XML文档和DTD文件来看文档是否符合规范，元素和标签使用是否正确。

内部的 DOCTYPE 声明

- 假如 DTD 被包含在您的 XML 源文件中，它应当通过下面的语法包装在一个 DOCTYPE 声明中：

```
<!DOCTYPE root-element [element-declarations]>
```

- 带有 DTD 的 XML 文档实例：

```
<?xml version="1.0"?>
<!DOCTYPE note [
        <!ELEMENT note (to,from,heading,body)>
        <!ELEMENT to (#PCDATA)>
        <!ELEMENT from (#PCDATA)>
        <!ELEMENT heading (#PCDATA)>
        <!ELEMENT body (#PCDATA)>
        ]>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend</body>
</note>
```

DTD 解释如下：

- **!DOCTYPE note** (第二行)定义此文档是 **note** 类型的文档。
- **!ELEMENT note** (第三行)定义 **note** 元素有四个元素：“to、from、heading,、body"
- **!ELEMENT to** (第四行)定义 **to** 元素为 "#PCDATA" 类型
- **!ELEMENT from** (第五行)定义 **from** 元素为 "#PCDATA" 类型
- **!ELEMENT heading** (第六行)定义 **heading** 元素为 "#PCDATA" 类型
- **!ELEMENT body** (第七行)定义 **body** 元素为 "#PCDATA" 类型

外部文档声明

- 假如 DTD 位于 XML 源文件的外部，那么它应通过下面的语法被封装在一个 DOCTYPE 定义中：

```
<!DOCTYPE root-element SYSTEM "filename">
```

- 这个 XML 文档和上面的 XML 文档相同，但是拥有一个外部的 DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

- 这是包含 DTD 的 "note.dtd" 文件:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

DTD - XML 构建模块

- 所有的 XML 文档均由以下简单的构建模块构成:
  - 元素
  - 属性
  - 实体
  - PCDATA
  - CDATA

命名实体

- 命名实体（在 XML 规范中也称为内部实体）就是我们在谈论 “实体” 时所指的实体。命名实体在 DTD 或内部子集（即文档中 <!DOCTYPE> 语句的一部分）中声明，在文档中用作引用。在 XML 文档解析过程中，实体引用将由它的表示替代。

- 简单来说，实体就是宏，它们在我们处理文档时得到扩展。

```
<!ENTITY c "Chris">
<!ENTITY ch "&c; Herborth">
```

- 在一个文档中使用 &c; 将扩展为 Chris， &ch; 将扩展为完整的 Chris Herborth。

外部实体

- 外部实体表示外部文件的内容。
- 外部实体在有些情况下很有用，比如说，您在创建一本图书并且想将每一章存储为一个单独的文件。您可能会创建一组如下所示的实体。

- **外部实体引用其他文件**

```
<!ENTITY chap1 SYSTEM "chapter-1.xml">
<!ENTITY chap2 SYSTEM "chapter-2.xml">
<!ENTITY chap3 SYSTEM "chapter-3.xml">
```

外部实体

- 现在，当您在主图书 XML 文件中将这些实体放到一起时，这些文件的内容将插入在引用点。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Pull in the chapter content: -->
&chap1;
&chap2;
&chap3;
```

- 当该 XML 文档被解析时，它将被读取为一个大文档，包含 chapter-1.xml、chapter-2.xml 和 chapter-3.xml 文件的内容；

PCDATA

- PCDATA 的意思是被解析的字符数据（parsed character data）。
- 可把字符数据想象为 XML 元素的开始标签与结束标签之间的文本。

- **PCDATA 是会被解析器解析的文本。这些文本将被解析器检查实体以及标记。**

- 文本中的标签会被处理，而实体会被展开。

CDATA

- CDATA 的意思是字符数据（character data）
- **CDATA 是不会被解析器解析的文本。**在这些文本中的标签不会被当作标记来对待，其中的实体也不会被展开。

DTD - 元素

- 在 DTD 中，XML 元素通过元素声明来进行声明。元素声明使用下面的语法:

- <!ELEMENT element-name category>
- 或
- <!ELEMENT element-name (element-content)>

空元素

- 空元素通过类别关键词EMPTY进行声明:
- <!ELEMENT element-name EMPTY>
- 实例:
- <!ELEMENT br EMPTY>

- XML example:
-

只有 PCDATA 的元素

- 只有 PCDATA 的元素通过圆括号中的 #PCDATA 进行声明:
- <!ELEMENT element-name (#PCDATA)>
- 实例:
- <!ELEMENT from (#PCDATA)>

带有任何内容的元素

- 通过类别关键词 ANY 声明的元素，可包含任何可解析数据的组合:

- <!ELEMENT element-name ANY>

- 实例:

- <!ELEMENT note ANY>

带有子元素（序列）的元素

- 带有一个或多个子元素的元素通过圆括号中的子元素名进行声明:

- <!ELEMENT element-name (child1)>
- 或
- <!ELEMENT element-name (child1,child2,...)>

- 实例:
- <!ELEMENT note (to,from,heading,body)>

- 当子元素按照由逗号分隔开的序列进行声明时，这些子元素必须按照相同的顺序出现在文档中。在一个完整的声明中，子元素也必须被声明，同时子元素也可拥有子元素。"note" 元素的完整声明是：

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

声明只出现一次的元素

- <!ELEMENT element-name (child-name)>

- 实例:

- <!ELEMENT note (message)>
- 上面的例子声明了：message 子元素必须出现一次，并且必须只在 "note" 元素中出现一次。

声明最少出现一次的元素

- <!ELEMENT element-name (child-name+)>

- 实例:

- <!ELEMENT note (message+)>
- 上面的例子中的加号（+）声明了：message 子元素必须在 "note" 元素内出现至少一次。

声明出现零次或多次的元素

- <!ELEMENT element-name (child-name*)>

  实例:

  <!ELEMENT note (message*)>
- 上面的例子中的星号（*）声明了：子元素 message 可在 "note" 元素内出现零次或多次。

声明出现零次或一次的元素

- <!ELEMENT element-name (child-name?)>

  实例:

  <!ELEMENT note (message?)>
- 上面的例子中的问号(?)声明了： 子元素 message 可在 "note" 元素内出现零次或一次。

声明"非.../即..."类型的内容

- 实例:

  <!ELEMENT note (to,from,header,(message|body))>
- 上面的例子声明了： "note" 元素必须包含 "to" 元素、"from" 元素、"header" 元素， 以及非 "message" 元素既 "body" 元素。

声明混合型的内容

- 实例:

  <!ELEMENT note (#PCDATA|to|from|header|message)*>
- 上面的例子声明了： "note" 元素可包含出现零次或多次的 PCDATA、"to"、"from"、"header" 或者 "message"。

DTD - 属性

- 属性声明使用下列语法:
- <!ATTLIST element-name attribute-name attribute-type attribute-value>

  DTD 实例:

  <!ATTLIST payment type CDATA "check">

  XML 实例:

  <payment type="check" />

局限性：

- 1) DTD 不遵守 XML 语法
- 2) DTD 数据类型有限
- 3) DTD 不可扩展
- 4) DTD 不支持命名空间

XML Schema

- XML Schema 是基于 XML 的 DTD 替代者。
- XML Schema 可描述 XML 文档的结构。
- XML Schema 语言也可作为 XSD（XML Schema Definition）来引用。
- XML Schema 的作用是定义 XML 文档的合法构建模块，类似 DTD。

XML Schema:

- 定义可出现在文档中的元素
- 定义可出现在文档中的属性
- 定义哪个元素是子元素
- 定义子元素的次序
- 定义子元素的数目
- 定义元素是否为空，或者是否可包含文本
- 定义元素和属性的数据类型
- 定义元素和属性的默认值以及固定值

为什么XML Schema 是 DTD 的继任者?

- XML Schema 可针对未来的需求进行扩展
- XML Schema 更完善，功能更强大
- XML Schema 基于 XML 编写
- XML Schema 支持数据类型
- XML Schema 支持命名空间
- XML Schema 是 W3C 标准
- XML Schema 在 2001 年 5 月 2 日成为 W3C 标准

XML Schema 支持数据类型

- XML Schema 最重要的能力之一就是对数据类型的支持。
- 通过对数据类型的支持:
  - 可更容易地描述允许的文档内容
  - 可更容易地验证数据的正确性
  - 可更容易地与来自数据库的数据一并工作
  - 可更容易地定义数据约束（data facets）
  - 可更容易地定义数据模型（或称数据格式）
  - 可更容易地在不同的数据类型间转换数据
- 数据约束，或称 facets，是 XML Schema 原型中的一个术语，中文可译为"面"，用来约束数据类型的容许值。

XML Schema 可保护数据通信

- 当数据从发送方被发送到接受方时，其要点是双方应有关于内容的相同的"期望值"。
- 通过 XML Schema，发送方可以用一种接受方能够明白的方式来描述数据。
- 一种数据，比如 "03-11-2004"，在某些国家被解释为11月3日，而在另一些国家为当作3月11日。
- 但是一个带有数据类型的 XML 元素，比如：<date type="date">2004-03-11</date>，可确保对内容一致的理解，这是因为 XML 的数据类型 "date" 要求的格式是 "YYYY-MM-DD"。

XML Schema 使用 XML 语法

- 另一个关于 XML Schema 的重要特性是，它们由 XML 编写。

- 由 XML 编写 XML Schema 有很多好处：
  - 不必学习新的语言
  - 可使用 XML 编辑器来编辑 XML Schema 文件
  - 可使用 XML 解析器来解析 XML Schema 文件

XML Schema 可扩展

- XML Schema 是可扩展的，因为它们由 XML 编写。

- **通过可扩展的 Schema 定义，您可以：**
  - 在其他 Schema 中重复使用您的 Schema
  - 创建由标准类型衍生而来的您自己的数据类型

XSD（XML Schema Definition）如何使用

- 一个简单的 XML 文档:
- 请看这个名为 "note.xml" 的 XML 文档:

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

XML Schema

- 下面这个例子是一个名为 "note.xsd" 的 XML Schema 文件，它定义了上面那个 XML 文档（"note.xml"）的元素：

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.w3schools.com"
           xmlns="http://www.w3schools.com">

    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to" type="xs:string"/>
                <xs:element name="from" type="xs:string"/>
                <xs:element name="heading" type="xs:string"/>
                <xs:element name="body" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

在 XML 文档中引用 Schema

```
<?xml version="1.0"?>

<note xmlns="http://www.runoob.com"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.runoob.com note.xsd">

    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

```
xmlns="http://www.runoob.com"
```

- 规定了默认命名空间的声明。此声明会告知 schema 验证器，在此 XML 文档中使用的所有元素都被声明于 "http://www.runoob.com" 这个命名空间。

- 定义可用的 XML Schema 实例命名空间：

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

- 您就可以使用 schemaLocation 属性了。此属性的值有两部分。第一部分是需要使用的命名空间。第二部分是供命名空间使用的 XML schema 的位置：

```
xsi:schemaLocation="http://www.runoob.com note.xsd"
```

<schema> 元素

- <schema> 元素是每一个 XML Schema 的根元素：

```
<?xml version="1.0"?>

<xs:schema>
    ...
    ...
</xs:schema>
```

- <schema> 元素可包含属性。一个 schema 声明往往看上去类似这样：

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.runoob.com"
           xmlns="http://www.runoob.com">
    ...
    ...
</xs:schema>
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

- 显示 schema 中用到的元素和数据类型来自命名空间 "http://www.w3.org/2001/XMLSchema"。同时它还规定了来自命名空间 "http://www.w3.org/2001/XMLSchema" 的元素和数据类型应该使用前缀 xs:

```
targetNamespace="http://www.runoob.com"
```

- 显示被此 schema 定义的元素 (note, to, from, heading, body) 来自命名空间： "http://www.runoob.com"。

```
xmlns="http://www.runoob.com"
```

- 指出默认的命名空间是 "http://www.runoob.com"。

XSD 简易元素

- 简易元素指那些仅包含文本的元素。
- 简易元素不会包含任何其他的元素或属性。
- 不过，"仅包含文本"这个限定却很容易造成误解。文本有很多类型。它可以是 XML Schema 定义中包括的类型中的一种（布尔、字符串、数据等等），或者它也可以是您自行定义的定制类型。

XSD 复合元素

- 复合元素指包含其他元素及/或属性的 XML 元素。

定义简易元素

- 定义简易元素的语法：

```
<xs:element name="xxx" type="yyy"/>
```

- 此处 xxx 指元素的名称，yyy 指元素的数据类型。XML Schema 拥有很多内建的数据类型。

最常用的类型是:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

实例

- 这是一些 XML 元素:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- 这是相应的简易元素定义:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

简易元素的默认值和固定值

- 简易元素可拥有指定的默认值或固定值。
- 当没有其他的值被规定时，默认值就会自动分配给元素。
- 在下面的例子中，默认值是 "red"：

```
<xs:element name="color" type="xs:string" default="red"/>
```

- 固定值同样会自动分配给元素，并且您无法规定另外一个值。
- 在下面的例子中，固定值是 "red"：

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XSD 属性

- 简易元素无法拥有属性。假如某个元素拥有属性，它就会被当作某种复合类型。

- 定义属性的语法是

```
<xs:attribute name="xxx" type="yyy"/>
```

实例

- 这是带有属性的 XML 元素:

```
<lastname lang="EN">Smith</lastname>
```

- 这是对应的属性定义:

```
<xs:attribute name="lang" type="xs:string"/>
```

属性的默认值和固定值

- 属性可拥有指定的默认值或固定值。
- 当没有其他的值被规定时，默认值就会自动分配给元素。
- 在下面的例子中，默认值是 "EN":

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

- 固定值同样会自动分配给元素，并且您无法规定另外的值。
- 在下面的例子中，固定值是 "EN":

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

可选的和必需的属性

- 在默认的情况下，属性是可选的。如需规定属性为必选，请使用 "use" 属性：

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

对内容的限定

- 当 XML 元素或属性拥有被定义的数据类型时，就会向元素或属性的内容添加限定。
- 假如 XML 元素的类型是 "xs:date"，而其包含的内容是类似 "Hello World" 的字符串，元素将不会（通过）验证。
- 通过 XML schema，您也可向您的 XML 元素及属性添加自己的限定。这些限定被称为 facet（编者注：意为(多面体的)面，可译为限定面）

对值的限定

- 下面的例子定义了带有一个限定且名为 "age" 的元素。age 的值不能低于 0 或者高于 120:

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

对一组值的限定

- 如需把 XML 元素的内容限制为一组可接受的值，我们要使用枚举约束（enumeration constraint）。
- 下面的例子定义了带有一个限定的名为 "car" 的元素。可接受的值只有：Audi, Golf, BMW：

```
<xs:element name="car">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Audi"/>
            <xs:enumeration value="Golf"/>
            <xs:enumeration value="BMW"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 上面的例子也可以被写为：

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
<xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
</xs:restriction>
</xs:simpleType>
```

- **注意：** 在这种情况下，类型 "carType" 可被其他元素使用，因为它不是 "car" 元素的组成部分。

对一系列值的限定

- 如需把 XML 元素的内容限制定义为一系列可使用的数字或字母，我们要使用模式约束（pattern constraint）。
- 下面的例子定义了带有一个限定的名为 "letter" 的元素。可接受的值只有小写字母 a - z 其中的一个:

```
<xs:element name="letter">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 使用方法和正则表达式类似

对长度的限定

- 如需限制元素中值的长度，我们需要使用 length、maxLength 以及 minLength 限定。
- 本例定义了带有一个限定且名为 "password" 的元素。其值必须精确到 8 个字符：

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 这个例子也定义了带有一个限定的名为 "password" 的元素。其值最小为 5 个字符，最大为 8 个字符：

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="5"/>
            <xs:maxLength value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XSD 复合元素

- 复合元素指包含其他元素及/或属性的 XML 元素。

- **有四种类型的复合元素：**
  - 空元素
  - 包含其他元素的元素
  - 仅包含文本的元素
  - 包含元素和文本的元素

- **注意：** 上述元素均可包含属性!

复合元素的例子

- 复合元素，"product"，是空的：

```
<product pid="1345"/>
```

- 复合元素，"employee"，仅包含其他元素：

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

- 复合 XML 元素，"food"，仅包含文本：

```
<food type="dessert">Ice cream</food>
```

- 复合XML元素，"description"包含元素和文本：

```
<description>
    It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

在 XML Schema 中，如何定义复合元素?

- 请看这个复合 XML 元素，"employee"，仅包含其他元素:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

- 在 XML Schema 中，我们有两种方式来定义复合元素:

- 1. 通过命名此元素，可直接对"employee"元素进行声明

```
<xs:element name="employee">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 假如您使用上面所描述的方法，那么仅有 "employee" 可使用所规定的复合类型。请注意其子元素，"firstname" 以及 "lastname"，被包围在指示器 <sequence>中。这意味着子元素必须以它们被声明的次序出现。

- "employee" 元素可以使用 type 属性，这个属性的作用是引用要使用的复合类型的名称:

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
<xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

- 如果您使用了上面所描述的方法，那么若干元素均可以使用相同的复合类型，比如这样

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
<xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

XSD 指示器

- 通过指示器，我们可以控制在文档中使用元素的方式。
- Order 指示器：
  - All
  - Choice
  - Sequence
- Occurrence 指示器：
  - maxOccurs
  - minOccurs

Order 指示器

- Order 指示器用于定义元素的顺序。
- **All 指示器**
- <all> 指示器规定子元素可以按照任意顺序出现，且每个子元素必须只出现一次：

```
<xs:element name="person">
    <xs:complexType>
        <xs:all>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

- **Choice 指示器**
- <choice> 指示器规定可出现某个子元素或者可出现另外一个子元素（非此即彼）：

- **Sequence 指示器**
- <sequence> 规定子元素必须按照特定的顺序出现：

Occurrence 指示器

- Occurrence 指示器用于定义某个元素出现的频率。

- **注意：** 对于所有的 "Order" 和 "Group" 指示器（any、all、choice 、sequence、group name 以及 group reference），其中的 max Occurs 以及 minOccurs 的默认值均为 1。

maxOccurs 指示器

- <maxOccurs> 指示器可规定某个元素可出现的最大次数:

```
<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 上面的例子表明，子元素 "child_name" 可在 "person" 元素中最少出现一次（其中 minOccurs 的默认值是 1），最多出现 10 次。

- **minOccurs 指示器**
- <minOccurs> 指示器可规定某个元素能够出现的最小次数:

- **提示：**如需使某个元素的出现次数不受限制，请使用 maxOccurs="unbounded" 这个声明:

XML 中的空白字符

```
<Author><FirstName>John</FirstName>
    <LastName>Smith</LastName></Author>

<Author>
<FirstName>John</FirstName>
<LastName>Smith</LastName>
</Author>

<Author>
    <FirstName>
        John
    </FirstName>
    <LastName>
        Smith
    </LastName>
</Author>
```

XML 中的空白字符

- XML 将以下四种字符归为空白字符：回车符（\r 或 ch(13)）、换行符（\n 或 ch(10)）、制表符 (\t) 以及空格 (' ')。在 XML 文档中，空白字符分为两类：
- *有意义空白字符* 是文档内容的一部分，应予以保留。
  - 位于属性值中。
  - 是元素文本内容的一部分且该文本还包含其他字符

- *无意义空白字符* 在编辑 XML 文档时使用，以增加可读性。这些空白字符一般在文档交付时不予保留。
- 通常，若没有 DTD 或 XML Schema模式定义，所有空白字符都是有意义空白字符，应当保留。（与具体解析器相关）

- 复合XML元素，"description"包含元素和文本：

```
<description>
    It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

Xml schema对空白字符的限定

- 如需规定对空白字符（whitespace characters）的处理方式，我们需要使用 whiteSpace 限定。
- 下面的例子定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "preserve"，这意味着 XML 处理器不会移除任何空白字符：

```
<xs:element name="address">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 这个例子也定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "replace"，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符）：

```
<xs:element name="address">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:whiteSpace value="replace"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 这个例子也定义了带有一个限定的名为 "address" 的元素。这个 whiteSpace 限定被设置为 "collapse"，这意味着 XML 处理器将移除所有空白字符（换行、回车、空格以及制表符会被替换为空格，开头和结尾的空格会被移除，而多个连续的空格会被缩减为一个单一的空格）：

```
<xs:element name="address">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:whiteSpace value="collapse"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML 中的空白字符处理

- **涉及到：**
- **可扩展样式表转换语言**（英语：Extensible Stylesheet Language Transformations，缩写**XSLT**）是一种样式转换标记语言，可以将XML数据档转换为另外的XML或其它格式，如HTML网页，纯文字
- 具体XML Parser

- 让我们看看这个名为 "shiporder.xml" 的 XML 文档：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="shiporder.xsd">
    <orderperson>John Smith</orderperson>
    <shipto>
        <name>Ola Nordmann</name>
        <address>Langgt 23</address>
        <city>4000 Stavanger</city>
        <country>Norway</country>
    </shipto>
    <item>
        <title>Empire Burlesque</title>
        <note>Special Edition</note>
        <quantity>1</quantity>
        <price>10.90</price>
    </item>
    <item>
        <title>Hide your heart</title>
        <quantity>1</quantity>
        <price>9.90</price>
    </item>
</shiporder>
```

- 上面的XML文档包括根元素 "shiporder"，其中包含必须名为 "orderid" 的属性。"shiporder" 元素包含三个不同的子元素："orderperson"、"shipto" 以及 "item"。"item" 元素出现了两次，它含有一个 "title"、一个可选 "note" 元素、一个 "quantity" 以及一个 "price" 元素。
- xsi全名：xml schema instance
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"，告知XML解析器根据某个 schema 来验证此文档。
- xsi:noNamespaceSchemaLocation="shiporder.xsd" 规定了 schema 的位置（在这里，它与 "shiporder.xml" 处于相同的文件夹）。

创建一个 XML Schema

- 现在，我们需要为上面这个 XML 文档创建一个 schema。

- 我们可以通过打开一个新的文件来开始，并把这个文件命名为 "shiporder.xsd"。要创建schema，我们仅仅需要简单地遵循 XML 文档中的结构，定义我们所发现的每个元素。首先我们开始定义一个标准的 XML 声明：

创建一个 XML Schema

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
</xs:schema>
```

- 在上面的 schema 中，我们使用了标准的命名空间 (xs)，与此命名空间相关联的 URI 是 Schema 的语言定义（Schema language definition），其标准值是 http://www.w3.org/2001/XMLSchema。

- 接下来，我们需要定义 "shiporder" 元素。此元素拥有一个属性，其中包含其他的元素，因此我们将它认定为复合类型。
- "shiporder" 元素的子元素被 xs:sequence 元素包围，定义了子元素的次序：

```
<xs:element name="shiporder">
    <xs:complexType>
        <xs:sequence>
            ...
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 然后我们需要把 "orderperson" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。
- 类型 (xs:string) 的前缀是由命名空间的前缀规定的，此命名空间与指示预定义的 schema 数据类型的 XML schema 相关联：

```
<xs:element name="orderperson" type="xs:string"/>
```

- 接下来，我需要把两个元素定义为复合类型："shipto" 和 "item"。我们从定义 "shipto" 元素开始：

```
<xs:element name="shipto">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 通过 schema，我们可使用 maxOccurs 和 minOccurs 属性来定义某个元素可能出现的次数。maxOccurs 定义某元素出现次数的最大值，而 minOccurs 则定义某元素出现次数的最小值。maxOccurs 和 minOccurs 的默认值都是 1!
- 现在，我们可以定义 "item" 元素了。这个元素可在 "shiporder" 元素内部出现多次。这是通过把 "item" 元素的 maxOccurs 属性的值设定为 "unbounded" 来实现的，这样 "item" 元素就可出现创作者所希望的任意多次。请注意，"note" 元素是可选元素。我们已经把此元素的 minOccurs 属性设定为 0 了：

```
<xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 现在，我们可以声明 "shiporder" 元素的属性了。由于这是一个必选属性，我们规定 use="required"。
- **注意：**此属性的声明必须被置于最后：

```
<xs:attribute name="orderid" type="xs:string" use="required"/>
```

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="shiporder">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="orderperson" type="xs:string"/>
                <xs:element name="shipto">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element name="address" type="xs:string"/>
                            <xs:element name="city" type="xs:string"/>
                            <xs:element name="country" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="item" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="title" type="xs:string"/>
                            <xs:element name="note" type="xs:string" minOccurs="0"/>
                            <xs:element name="quantity" type="xs:positiveInteger"/>
                            <xs:element name="price" type="xs:decimal"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="orderid" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- 根据以下数据回答下列问题：“小明今年22岁，来自安徽大学。兴趣是看电影和旅游。他有两个姐姐，一个叫小芬，今年25岁，职业是护士。还有一个叫小芳，今年23岁，是一名小学老师”（20分）
- 用XML语法描述以上数据
- 给出前问中XML文档对应的XML Schema定义（注：XML Schema的标准命名空间为xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"，常用标签为：xs:schema、xs:simpleType、xs:complexType、xs:sequence、xs:restriction、xs:minInclusive、xs:maxInclusive等，常用的数据类型为xs:string、xs:decimal、xs:integer、xs:boolean等）

创建一个 XML Schema

- 现在，我们需要为上面这个 XML 文档创建一个 schema。

- 我们可以通过打开一个新的文件来开始，并把这个文件命名为 “person.xsd"。要创建schema，我们仅仅需要简单地遵循 XML 文档中的结构，定义我们所发现的每个元素。首先我们开始定义一个标准的 XML 声明：

创建一个 XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
</xs:schema>
```

- 在上面的 schema 中，我们使用了标准的命名空间 (xs)，与此命名空间相关联的 URL 是 Schema 的语言定义（Schema language definition），其标准值是 http://www.w3.org/2001/XMLSchema。

- 接下来，我们需要定义 “person” 元素。此元素包含其他的子元素，因此我们将它认定为复合类型。
- " person " 元素的子元素被 xs:sequence 元素包围，定义了子元素的次序:

```
<xs:element name=“person">
    <xs:complexType>
        <xs:sequence>
            ...
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- 然后我们需要把 “name" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。
- 类型 (xs:string) 的前缀是由命名空间的前缀规定的，此命名空间与指示预定义的 schema 数据类型的 XML schema 相关联：

```
<xs:element name=“name" type="xs:string"/>
```

- 然后我们需要把 “age" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。
- age 的值不能低于 0 或者高于 120:

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- 然后我们需要把 “university" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。
- 类型 (xs:string) 的前缀是由命名空间的前缀规定的，此命名空间与指示预定义的 schema 数据类型的 XML schema 相关联：

```
<xs:element name=“university" type="xs:string"/>
```

- 然后我们需要把 “interest" 元素定义为简易类型（这是因为它不包含任何属性或者其他的元素）。
- 通过 schema，我们可使用 maxOccurs 和 minOccurs 属性来定义某个元素可能出现的次数。maxOccurs 定义某元素出现次数的最大值，而 minOccurs 则定义某元素出现次数的最小值。maxOccurs 和 minOccurs 的默认值都是 1!
- 这个元素可在 “person" 元素内部出现多次。这是通过把 “interest " 元素的 maxOccurs 属性的值设定为 "unbounded" 来实现的，这样 " interest " 元素就可出现创作者所希望的任意多次。

- `<xs:element name=“interest" type="xs:string" maxOccurs=“unbounded"/>`

- 定义sister复合类型

```
<xs:element name="sister" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="age">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">
                        <xs:minInclusive value="0"/>
                        <xs:maxInclusive value="120"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="job" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="age">
                    <xs:simpleType>
                        <xs:restriction base="xs:integer">
                            <xs:minInclusive value="0"/>
                            <xs:maxInclusive value="120"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="university" type="xs:string"/>
                <xs:element name="interest" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="sister" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element name="age">
                                <xs:simpleType>
                                    <xs:restriction base="xs:integer">
                                        <xs:minInclusive value="0"/>
                                        <xs:maxInclusive value="120"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="job" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- Age元素出现了多次，因此可以专门给它定义一个数据类型。

```
<xs:simpleType name="ageinfo">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="120"/>
    </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="age" type="ageinfo"/>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="ageinfo">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="age" type="ageinfo"/>
                <xs:element name="university" type="xs:string"/>
                <xs:element name="interest" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="sisters" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element name="age" type = "ageinfo"/>
                            <xs:element name="job" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

答疑

JSON

- JSON: **J**ava**S**cript **O**bject **N**otation(JavaScript 对象表示法)
- JSON 是存储和交换文本信息的语法。类似 XML。
- JSON 独立于语言：JSON 使用 Javascript语法来描述数据对象，但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言。
- JSON 具有自我描述性，更易理解
- JSON 比 XML 更小、更快，更易解析。

JSON 历史

- JSON是Douglas Crockford在2001年开始推广使用的数据格式，在2005年-2006年正式成为主流的数据格式，雅虎和谷歌就在那时候开始广泛地使用JSON格式。

- JSON是基于JavaScript脚本语言的一部分
- 所有JSON格式的文本也是合法的JavaScript代码
- JSON是JavaScript的子集

JSON 语法

- JSON 语法是 JavaScript 对象表示语法的子集。
- 数据在名称/值对中
- 数据由逗号分隔
- 大括号保存对象
- 中括号保存数组

JSON 名称/值对

- JSON 数据的书写格式是：名称/值对（键/值对）。
- 名称/值对包括字段名称（在双引号中），后面写一个冒号，然后是值：

  {"name" : "菜鸟教程“}

注意：有的地方也把字段名称叫做属性，值称为属性值；

注意

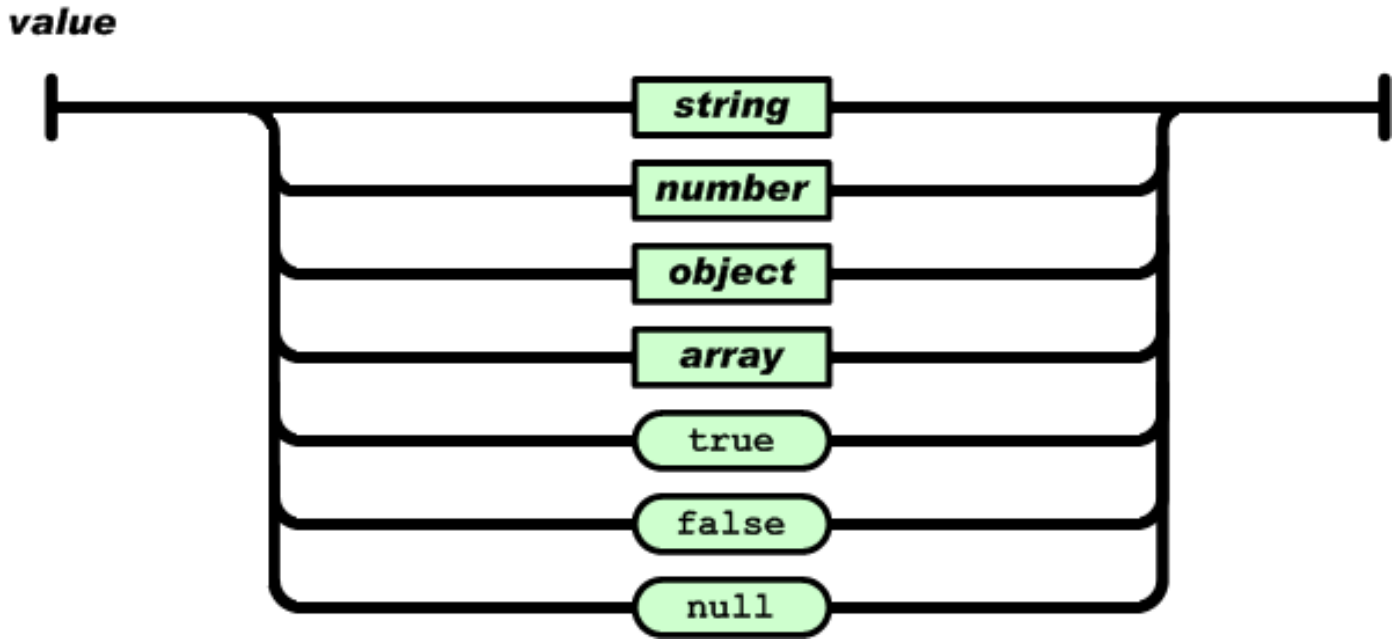
- JSONs数据必须是对象或者数组：
  - ['foo', {'bar': ('baz', None, 1.0, 2)}]
  - {'4': 5, '6': 7}

- “a”: 0 （错误）

JSON 值

- JSON 值可以是:
- 数字（整数或浮点数）
- 字符串（在双引号中）
- 逻辑值（true 或 false）
- 数组（在中括号中）
- 对象（在大括号中）
- null

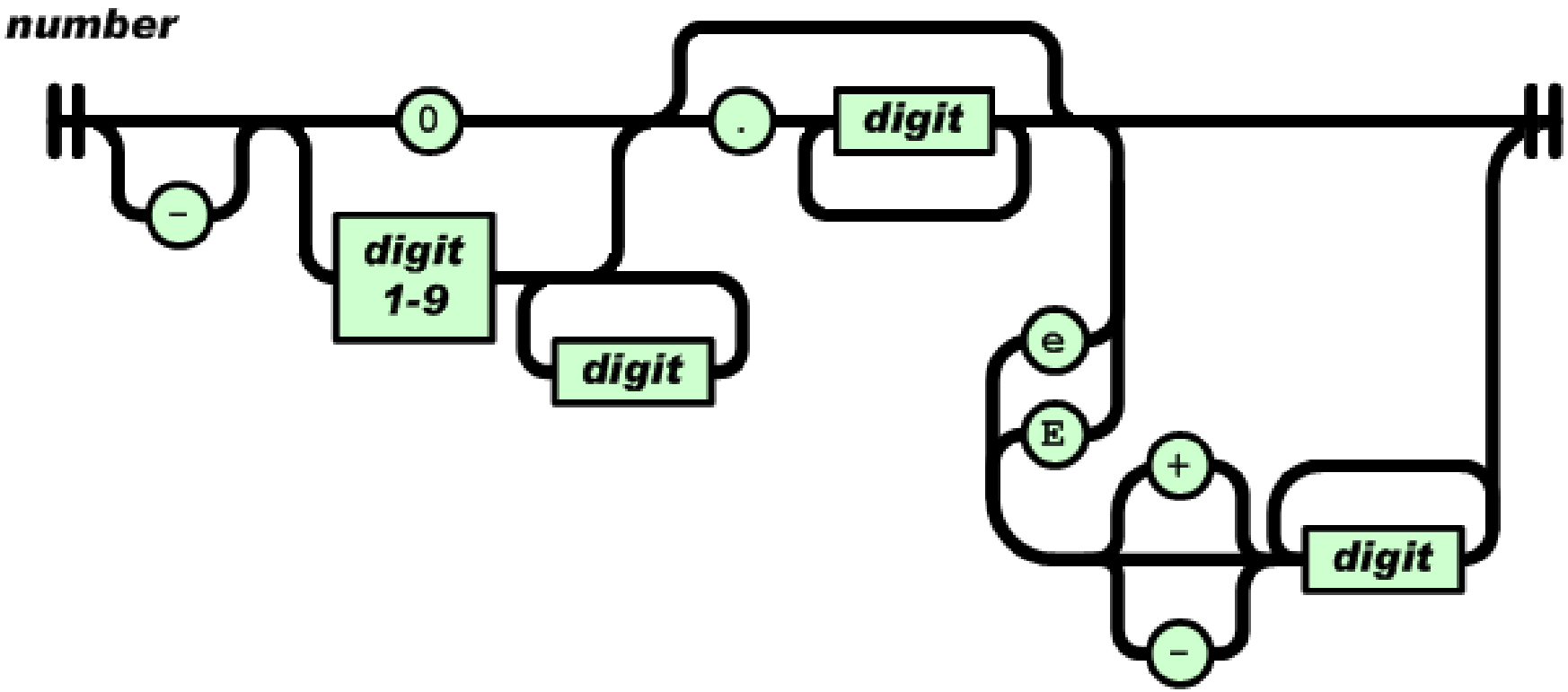
JSON 值



JSON 数字

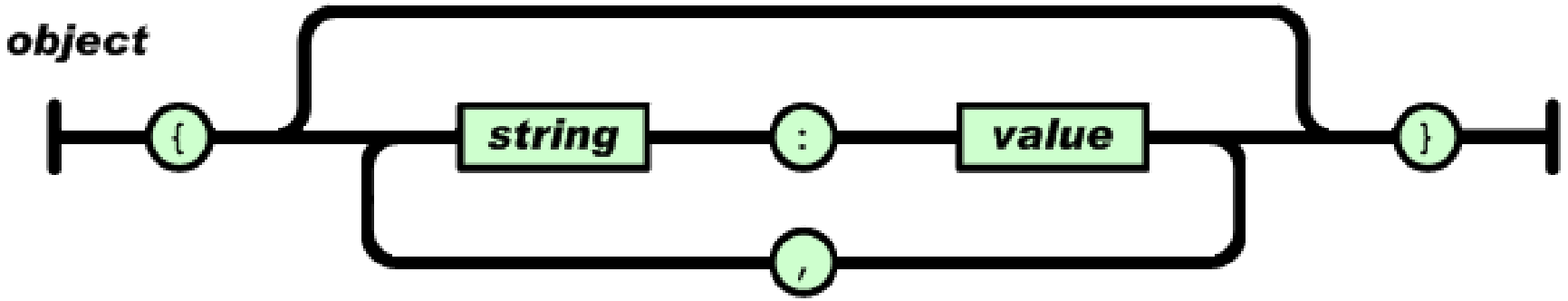
- JSON 数字可以是整型或者浮点型:
  - {"age":30 }

JSON数字



JSON 对象

- JSON 对象在大括号（{}）中书写，对象可以包含多个名称/值对：
  - { "name":"菜鸟教程" , "url":"www.runoob.com" }



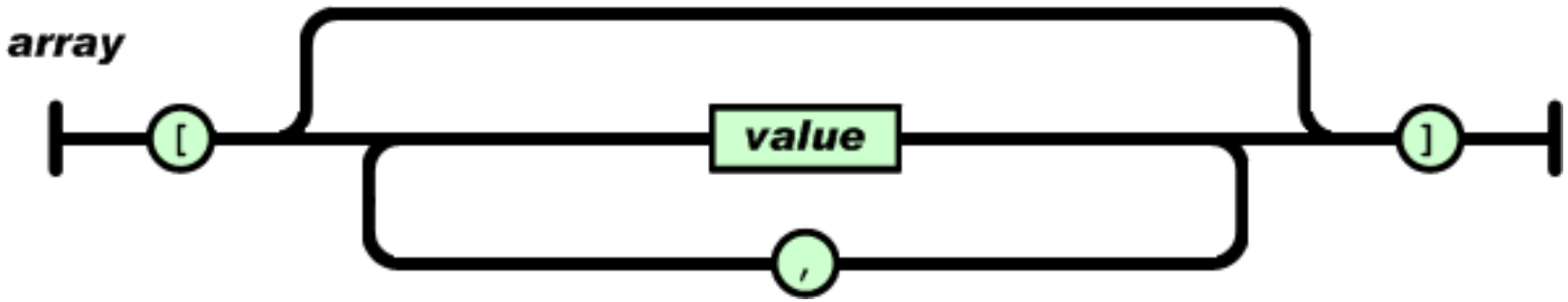
JSON 数组

- JSON 数组在中括号中书写，数组可包含多个对象：

```
{ "sites":[
    { "name":"菜鸟教程" , "url":"www.runoob.com" },
    { "name":"google" , "url":"www.google.com" },
    { "name":"微博" , "url":"www.weibo.com" }
    ] }
```

- 在上面的例子中，对象 "sites" 是包含三个对象的数组。每个对象代表一条关于某个网站（name、url）的记录

JSON 数组

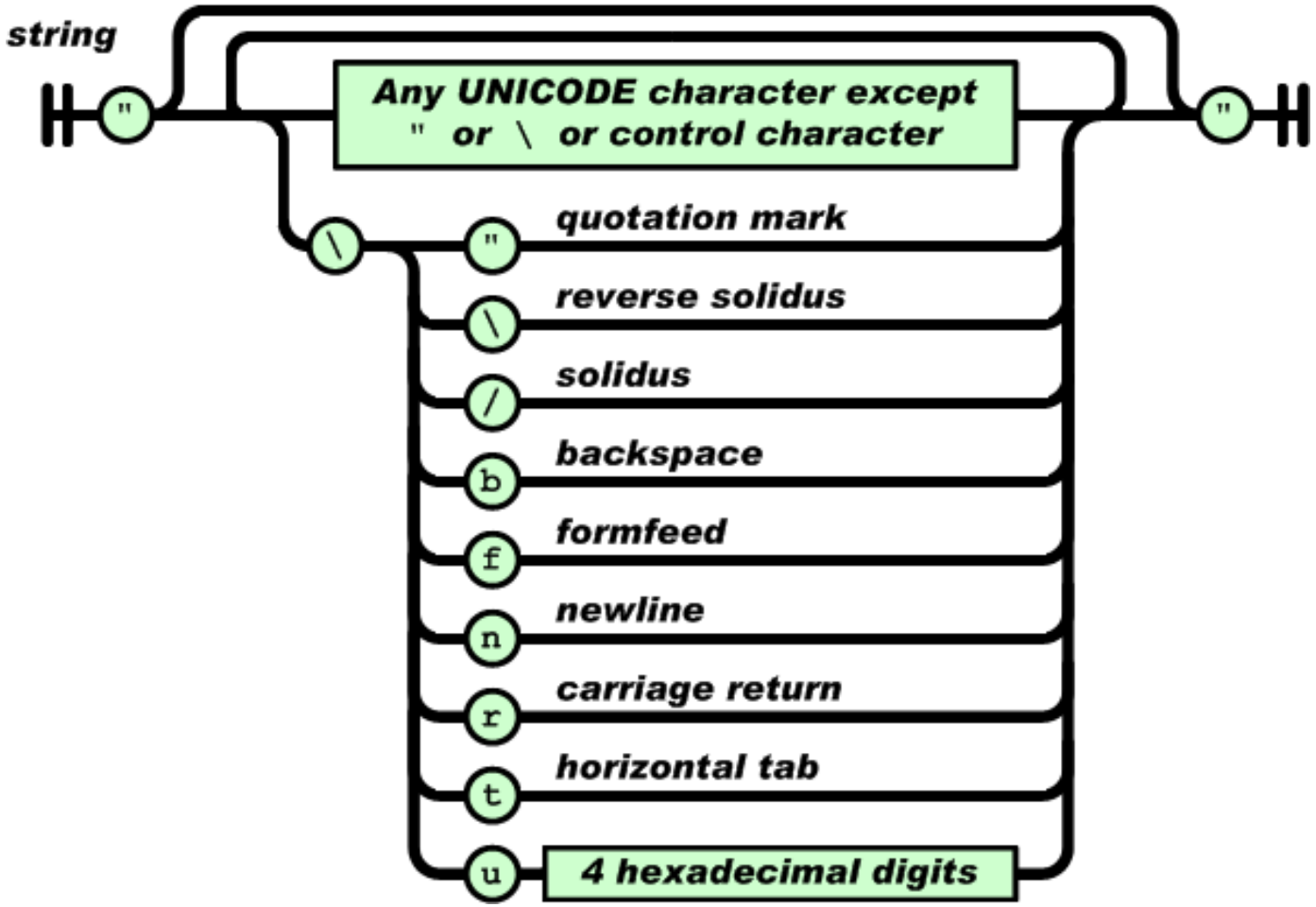


JSON 布尔值， null 值

- JSON 布尔值可以是 true 或者 false：
  - {"flag":true }

- JSON 可以设置 null 值：
  - {"runoob":null }

JSON字符串


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

JSON 实例

```
{
  "sites": [
    { "name":"菜鸟教程" , "url":"www.runoob.com" },
    { "name":"google" , "url":"www.google.com" },
    { "name":"微博" , "url":"www.weibo.com" }
  ]
}
```

- **注意： JSON「最后不能加逗号」**

XML 实例

```
<sites>
    <site>
        <name>菜鸟教程</name> <url>www.runoob.com</url>
    </site>
    <site>
        <name>google</name> <url>www.google.com</url>
    </site>
    <site>
        <name>微博</name> <url>www.weibo.com</url>
    </site>
</sites>
```

JSON 与 XML 的相同之处

- JSON 和 XML 数据都是 "自我描述"， 都易于理解。
- JSON 和 XML 数据都是有层次的结构
- JSON 和 XML 数据可以被大多数编程语言使用

JSON 与 XML 的不同之处

- JSON 不需要结束标签
- JSON 更加简短
- JSON 读写速度更快
- JSON 可以使用数组

下列XML元素，哪一个是合法的()

- A）<A>hello</A>
- B) <A>1 + 1 < 3</A>
- C) <A>hello
- D) <A x="1"><A>

下列XML元素，哪一个是合法的()

- A） <A>hello</A>
- B） <A>1 + 1 < 3</A>
- C) <A>hello
- D) <A x="1"><A>

- <A x=“1” /A> <A x=“1” ></A>

有关JSON描述错误的是（）

- A) JSON 指的是 JavaScript 对象表示法
- B）JSON 是存储和交换文本信息的语法。类似 XML。
- C）JSON 比 XML体积稍大，但是更快，更易解析。
- D）JSON 是轻量级的文本数据交换格式

下面哪一个是JSON数据？

- A）　{name:"xiaoming",age,"student"}
- B）　{"name":"xiaoming","age":"student"}
- C）　{"xiaoming","student"}
- D）　["xiaoming","student"]

答疑

- 根据以下数据回答下列问题：“小明今年22岁，来自安徽大学。兴趣是看电影和旅游。他有两个姐姐，一个叫小芬，今年25岁，职业是护士。还有一个叫小芳，今年23岁，是一名小学老师”（20分）
- （1）用JSON语法描述以上数据

```
{
  "name": "小明",
  "age": "22",
  "university": "安徽大学",
  "interests": [
    "电影",
    "旅游"
  ],
  "sisters": [
    {
      "name": "小芬",
      "age": "25",
      "job": "护士"
    },
    {
      "name": "小芳",
      "age": "23",
      "job": "小学老师"
    }
  ]
}
```