AJAX的数据格式

管理科学与工程学科
耿方方

主要内容

- XML
- JOSN

XML

什么是XML

- XML(Extensible Markup Language):可扩展的标记语言。
- 可扩展的意思：相对与我们接触的HTML（超文本标记语言），我们在编辑网页文件时，所有能够在网页文件中使用的HTML标签都是预先规定好的，不能根据自己的意愿添加规定外的标签。而如果编辑一个XML文件，我们可根据自己的意愿定义标签来完成，例如：
- <note>
- <to>George</to>
- <from>John</from>
- <heading>Reminder</heading>
- <body>Don't forget the meeting!</body>
- </note>
- 标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。XML是一种元标记语言，即定义了用于定义其它特定领域有关语义的、结构化的标记语言，这些标记语言将文档分成许多部件并对这些部件加以标识。

XML

什么是XML

- XML 是一种标记语言，很类似 HTML
- XML 的设计宗旨是传输数据，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，可扩展标记语言（XML）于 1998 年 2 月 10 日成为 W3C 的推荐标准。

XML

XML与HTML的区别

- XML 不是 HTML 的替代。
- XML 和 HTML 为不同的目的而设计：
- XML 被设计为传输和存储数据，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的用途

- XML把数据从HTML分离

如果你需要在 HTML 文档中显示动态数据，那么每当数据改变时将花费大量的时间来编辑 HTML。通过 XML，数据能够存储在独立的 XML 文件中。这样你就可以专注于使用 HTML 进行布局和显示，并确保修改底层数据不再需要对 HTML 进行任何的改变。通过使用几行 JavaScript，你就可以读取一个外部 XML 文件，然后更新 HTML 中的数据内容。

- XML简化数据共享

XML数据以纯文本格式进行存储，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。

- XML简化数据传输

通过 XML，可以在不兼容的系统之间轻松地交换数据。对开发人员来说，其中一项最费时的挑战一直是在因特网上的不兼容系统之间交换数据。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的用途

- XML简化平台的变更

升级到新的系统（硬件或软件平台），总是非常费时的。必须转换大量的数据，不兼容的数据经常会丢失。XML数据以文本格式存储。这使得 XML 在不损失数据的情况下，更容易扩展或升级到新的操作系统、新应用程序或新的浏览器。

- XML使您的数据更有用

由于 XML 独立于硬件、软件以及应用程序，XML 使您的数据更可用，也更有用。不同的应用程序都能够访问您的数据，不仅仅在 HTML 页中，也可以从 XML 数据源中进行访问。通过 XML，您的数据可供各种阅读设备使用（手持的计算机、语音设备、新闻阅读器等），还可以供盲人或其他残障人士使用。

- XML用于创建新的 Internet 语言

很多新的 Internet 语言是通过 XML 创建的。

XML树结构

- XML文档形成一种树结构，它从“根部”开始，然后扩展到“枝叶”。

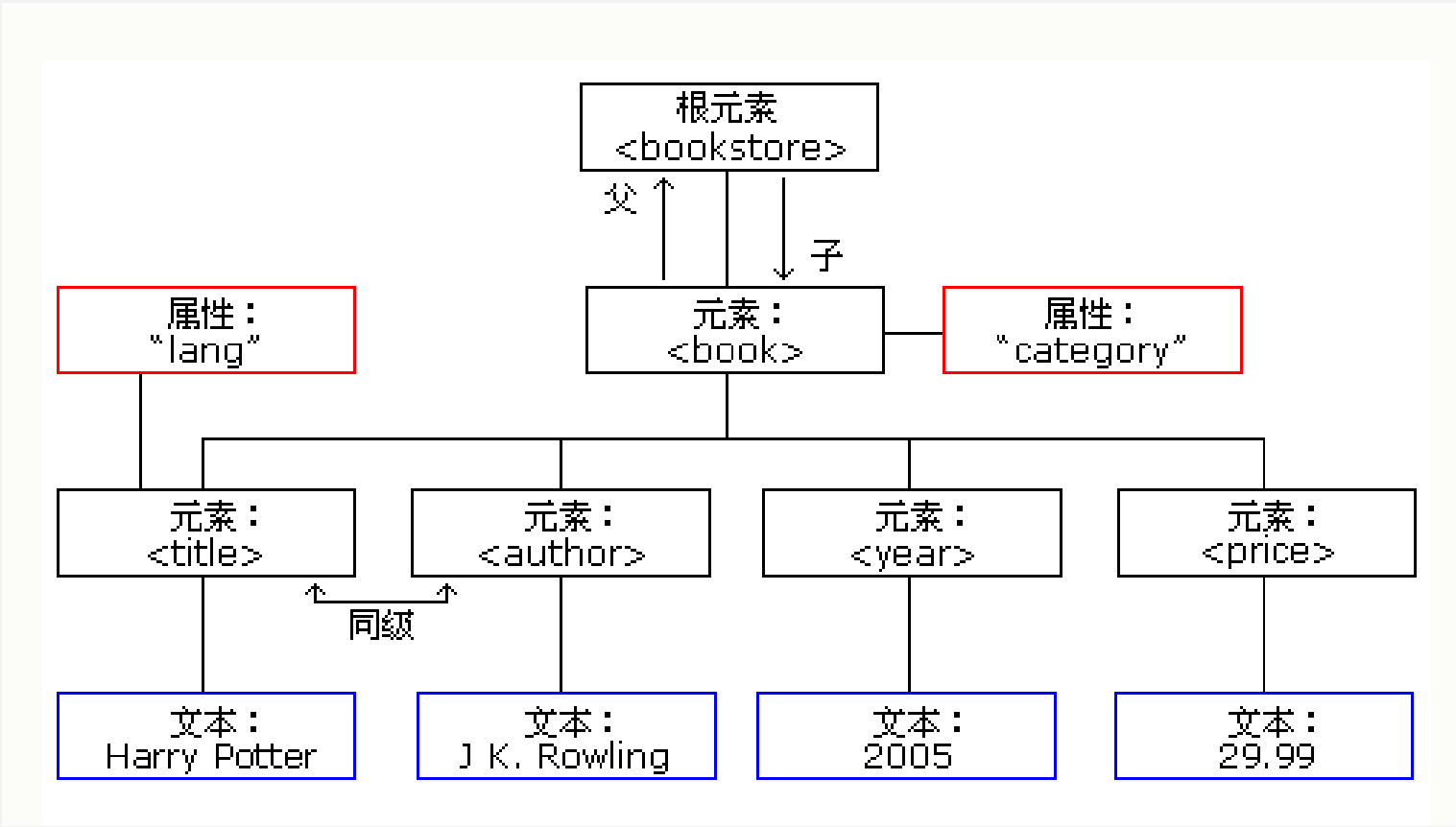
XML文档必须包含根元素。该元素是所有其他元素的父元素。XML文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端。父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。所有元素均可拥有文本内容和属性（类似 HTML 中）。

XML树结构

- 例如：<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>
</bookstore>
```

XML树结构


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语法规则

- 任何的起始标签都必须有一个结束标签。
- 可以采用另一种简化语法，可以在一个标签中同时表示起始和结束标签。这种语法是在大于符号之前紧跟一个斜线（/），例如<Web前端开发与实践 />，XML解析器会将其翻译成< Web前端开发与实践></ Web前端开发与实践>。
- 标签必须按合适的顺序进行嵌套，所有结束标签必须按顺序匹配起始标签。
- 所有的属性都必须有值。
- 所有的属性都必须在值的周围加上双引号。
- XML标签对大小写敏感。

- XML中的注释

  在XML中编写注释的语法与HTML的语法很相似：

  <!-- This is a comment -->

- 在XML中，空格会被保留

  HTML会把多个连续的空格字符裁减（合并）为一个：

  HTML: Hello my name is David.

  输出: Hello my name is David.

- 在XML中，文档中的空格不会被删节。

XML

- 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。
- XML 元素必须遵循以下命名规则：
- 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
- 名称不能包含空格
- 可使用任何名称，没有保留的字词。

XML

XML属性

- 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 标签可以这样写：
- <person sex="female">
- 避免 XML 属性？
- 因使用属性而引起的一些问题：
- 属性无法包含多重的值（元素可以）
- 属性无法描述树结构（元素可以）
- 属性不易扩展（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）：

XML

XML属性

- <note day="08" month="08" year="2008"
- to="George" from="John" heading="Reminder"
- body="Don't forget the meeting!">
- </note>

XML声明

- 大多数XML文档以XML声明作为开始，它向解析器提供了关于文档的基本信息。建议使用XML声明，但它不是必须的。如下所示：
- <?xml version="1.0" encoding="ISO-8859-1" standalone=" no" ?>
- version是使用得XML版本；目前使用的是1.0；
- encoding是该文档使用的字符集。如果没有指定encoding，XML解析器会假定字符在UTF-8字符集中。
- standalone(yes or no)定义了是否可以在不读取任何其他文件的情况下处理该文档。因为standdalone=“no”是缺省值，所以很少会在XML声明中看到standalone。

XML

CDATA

- 术语 CDATA 指的是不应由 XML 解析器进行解析的文本数据（Unparsed Character Data）。
- 在 XML 元素中，"<" 和 "&" 是非法的。
- "<" 会产生错误，因为解析器会把该字符解释为新元素的开始。
- "&" 也会产生错误，因为解析器会把该字符解释为字符实体的开始。
- 某些文本，比如 JavaScript 代码，包含大量 "<" 或 "&" 字符。为了避免错误，可以将脚本代码定义为 CDATA。
- CDATA 部分中的所有内容都会被解析器忽略。
- CDATA 部分由 "<![CDATA[" 开始，由 "]]>" 结束：

XML

CDATA

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1;
  }
else
  {
  return 0;
  }
}
]]>
</script>
```

XML

处理指令

- XML文件还可以包括处理指令，这些指令可以将命令或信息传给正在处理XML数据的应用。处理指令格式如下：
- <?target instructions?>
- 其中target是进行XML数据处理的应用名称，instrucions是一个字符串，包含了传给应用得信息和命令。
- 由于这些指令都是应用专署的，因此XML文件可以有多条处理指令，用来告诉不同的应用去做类似的事情，但是可能做事的方式各不相同。
- <?cocoon-process type=”sql”?>
- 当cocoon处理XML文档时，它会寻找以cocoon-process开头的处理指令，然后处理XML文档。在该示例中，type=“sql”属性告诉cocoon，XML文档包含一个sql语句。

XML

处理指令

- XML文件还可以包括处理指令，这些指令可以将命令或信息传给正在处理XML数据的应用。处理指令格式如下：
- <?target instructions?>
- 其中target是进行XML数据处理的应用名称，instrucions是一个字符串，包含了传给应用得信息和命令。
- 由于这些指令都是应用专署的，因此XML文件可以有多条处理指令，用来告诉不同的应用去做类似的事情，但是可能做事的方式各不相同。

XML

命名空间

- XML 命名空间提供避免元素命名冲突的方法。
- 假设有个表示个人尊称的<title>元素，这是对元素名称非常合理的选择。如果您经营一家网上书店，您或许会创建一个表示书名的<title>元素。如果您经营一家网上抵押放款公司，您或许会创建表示一份财产名称的<title>元素。
- 要使用命名空间，需要定义一个名称空间前缀，然后将它映射至一个特殊字符串。
- 例如：

XML

命名空间

- <?xml version=” 1.0” ?>
- <customer_summary
- xmlns.addr=” http://www.xyz.com/address/” >
- xmlns.books=” http://www.xyz.com/books”
- xmlns.mortgage=” http://www.xyz.com/title” >
- <addr:name><title>Mrs</title></addr:name>
- <books:title>Lord of the Rings</books.title>
- <mortgate:title>NC2948-388-1983</mortgage:title>
- 有了命名空间标准，就可以使用两套或多套XML标准集，以模块化的方式编写XML文档。

XML

- XML文档包含元素和属性，有效地XML文档必须符合W3C制定的规则。
- 通过名称空间可以实现词汇表共享。名称空间可以是默认的，在定义名称空间的元素的作用域范围内应用。属性不能属于任何默认的名称空间。
- 一个完整的XML文档包括如下组成部分
- -XML声明
- -DOCTYPE声明
- -处理指令
- 元素
- 注释
- CDATA区

- 拥有正确语法的 XML 被称为“形式良好”的 XML。
- 通过 DTD 验证的 XML 是“合法”的 XML。
- “形式良好”（Well Formed）的 XML 文档会遵守前几章介绍过的 XML 语法规则：
- XML 文档必须有根元素
- XML 文档必须有关闭标签
- XML 标签对大小写敏感
- XML 元素必须被正确的嵌套
- XML 属性必须加引号

- 验证 XML 文档

合法的 XML 文档是“形式良好”的 XML 文档，同样遵守文档类型定义(DTD)的语法规则：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE note SYSTEM "Note.dtd">

<note>

<to>George</to>

<from>John</from>

<heading>Reminder</heading>

<body>Don't forget the meeting!</body>

</note>
```

- 验证 XML 文档

XML 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:

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

- 验证 XML 文档

XML Schema

```
<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>
```

XML验证

- 一个简单的XML:
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <breakfast_menu>
- <food>
- <name>Belgian Waffles</name>
- <price>$5.95</price>
- <description>two of our famous Belgian Waffles with plenty of real maple syrup</description>
- <calories>650</calories>
- </food>
- <food>
- <name>Strawberry Belgian Waffles</name>
- <price>$7.95</price>
- <description>light Belgian waffles covered with strawberries and whipped cream</description>
- <calories>900</calories>
- </food>
- </breakfast_menu>

- 案例1：使用AJAX读取XML文件

JOSN

什么是josn

- JSON 指的是 JavaScript 对象表示法（JavaScript Object Notation）
- JSON 是轻量级的文本数据交换格式
- JSON 独立于语言
- JSON 具有自我描述性，更易理解
- 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。

JOSN

- 相比 XML 的不同之处

  没有结束标签

  更短

  读写的速度更快

  能够使用内建的 JavaScript eval() 方法进行解析

  使用数组

  不使用保留字

- 类似 XML

  JSON 是纯文本

  JSON 具有“自我描述性”（人类可读）

  JSON 具有层级结构（值中存在值）

  JSON 可通过 JavaScript 进行解析

  JSON 数据可使用 AJAX 进行传输

JOSN

为什么使用JOSN?

- 对于 AJAX 应用程序来说，JSON 比 XML 更快更易使用：
- 使用 XML

  读取 XML 文档

  使用 XML DOM 来循环遍历文档

  读取值并存储在变量中
- 使用 JSON

  读取 JSON 字符串

  用 eval() 处理 JSON 字符串

JOSN

- JSON 语法是 JavaScript 对象表示法语法的子集。

  数据在名称/值对中

  数据由逗号分隔

  花括号保存对象

  方括号保存数组

  例如JOSN数组:

```
 {
"employees": [
{ "firstName":"John" , "lastName":"Doe" },
{ "firstName":"Anna" , "lastName":"Smith" },
{ "firstName":"Peter" , "lastName":"Jones" }
]
}
```

JOSN

- 实例2:
- <body>
- <h2>通过 JSON 字符串来创建对象</h3>
- First Name:
- <script type="text/javascript">
- var employees = [
- { "firstName":"Bill" , "lastName":"Gates" },
- { "firstName":"George" , "lastName":"Bush" },
- { "firstName":"Thomas" , "lastName": "Carter" }
-];
- employees[1].firstName="Jobs";
- document.getElementById("fname").innerHTML=employees[1].firstName;
- </script>
- </body>

JOSN

- 把 JSON 文本转换为 JavaScript 对象
- JSON 最常见的用法之一，是从 web 服务器上读取 JSON 数据（作为文件或作为 HttpRequest），将 JSON 数据转换为 JavaScript 对象，然后在网页中使用该数据。
- 由于 JSON 语法是 JavaScript 语法的子集，JavaScript 函数 eval() 可用于将 JSON 文本转换为 JavaScript 对象。
- eval() 函数使用的是 JavaScript 编译器，可解析 JSON 文本，然后生成 JavaScript 对象。必须把文本包围在括号中，这样才能避免语法错误：
- var obj = eval ("(" + txt + ")");

JOSN

- 把 JSON 文本转换为 JavaScript 对象
- JSON 最常见的用法之一，是从 web 服务器上读取 JSON 数据（作为文件或作为 HttpRequest），将 JSON 数据转换为 JavaScript 对象，然后在网页中使用该数据。
- 由于 JSON 语法是 JavaScript 语法的子集，JavaScript 函数 eval() 可用于将 JSON 文本转换为 JavaScript 对象。
- eval() 函数使用的是 JavaScript 编译器，可解析 JSON 文本，然后生成 JavaScript 对象。必须把文本包围在括号中，这样才能避免语法错误：
- var obj = eval ("(" + txt + ")");

JOSN

- 案例3:
- <h2>通过 JSON 字符串来创建对象</h3>
-
- First Name:
- Last Name:
-
- <script type="text/javascript">
- var txt = '{"employees":[' +
- '{"firstName":"Bill","lastName":"Gates" },' +
- '{"firstName":"George","lastName":"Bush" },' +
- '{"firstName":"Thomas","lastName":"Carter" }]}';

- var obj = eval ("(" + txt + ")");

- document.getElementById("fname").innerHTML=obj.employees[1].firstName
- document.getElementById("lname").innerHTML=obj.employees[1].lastName
- </script>

JOSN

- JOSN解析器
- 提示：eval() 函数可编译并执行任何 JavaScript 代码。这隐藏了一个潜在的安全问题。
- 使用 JSON 解析器将 JSON 转换为 JavaScript 对象是更安全的做法。JSON 解析器只能识别 JSON 文本，而不会编译脚本。
- 在浏览器中，这提供了原生的 JSON 支持，而且 JSON 解析器的速度更快。
- 较新的浏览器和最新的 ECMAScript（JavaScript）标准中均包含了原生的对 JSON 的支持。

JOSN

- 案例4:
- <h2>JSON解析器</h2>
-
- First Name:
- Last Name:
-
- <script type="text/javascript">
- var txt = '{"employees":[' +
- '{"firstName":"Bill","lastName":"Gates" },' +
- '{"firstName":"George","lastName":"Bush" },' +
- '{"firstName":"Thomas","lastName":"Carter" }]}';
- obj = JSON.parse(txt);
- document.getElementById("fname").innerHTML=obj.employees[1].firstName
- document.getElementById("lname").innerHTML=obj.employees[1].lastName
- </script>