Ajax技术的数据响应优化

周学术讨论班

谭力 | 华东师范大学软件工程实验室 | 2009.3.4

内容提要

回顾

文章概述

文章要点

总结与展望

提问

回顾

Ø 2008.10.27 周学术讨论班

《Bulletproof Ajax》- Ajax的安全性、缺陷及其解决方案

- Ajax的基本概念 - **Basic concepts**
  - Ajax的全称是**Asynchronous JavaScript and XML**，即异步JavaScript与XML， 是几项按一定方式组合在一起，协同发挥各自作用的技术组合；
  - **2005 年2月**被**Jesse James Garrett**首先提出。
- 安全的Ajax - **Secure Ajax**
  - **网络本身的安全性**、**JavaScript的安全性**以及**数据加密**
- 健壮的Ajax - **Robust Ajax**
  - **性能**、**兼容性**、**可移植性**、**收藏与后退功能失效**以及**内存泄漏**

回顾

Ø 2008.12.16

《Ajax技术的数据响应优化》成文  《计算机应用》

Ø 2009.2.28

修回

Ø 2009.3.3

刊用

内容提要

回顾

文章概述

文章要点

总结与展望

提问

文章概述

Ø 目的、方法与结论

针对Ajax工作原理和数据传输性能进行了分析，随后从响应数据的返回格式角度入手，进一步通过实验对比了两种数据格式XML和JSON的差异和优劣，并基于实际情况给出了权衡取舍的一些建议，总结出了较为优化的Ajax模型，使目前Ajax带来的数据响应冗余的缺陷得到了改善。

Ø 关注点

- **Ajax技术的性能优化**
  - **数据响应性能**
    - **响应数据的返回格式（XML和JSON）**

内容提要

回顾

文章概述

文章要点

总结与展望

提问

文章要点

Ø 1 目前 Ajax性能优化的研究进展

n 意义及背景

应用程序的性能取决于两个因素：**算法复杂度**和**对系统资源的消耗**（最重要的是**内存**和**CPU**）。人们对程序的逻辑性的重视往往高于其性能，但实际上一个程序的性能非常重要，它是用户选择使用某个软件产品的首要考虑因素。一个性能极差的软件，用户会马上放弃它。

我们往往重点关注于如何用Ajax实现强大的功能，却淡化了对Ajax性能的进一步要求。

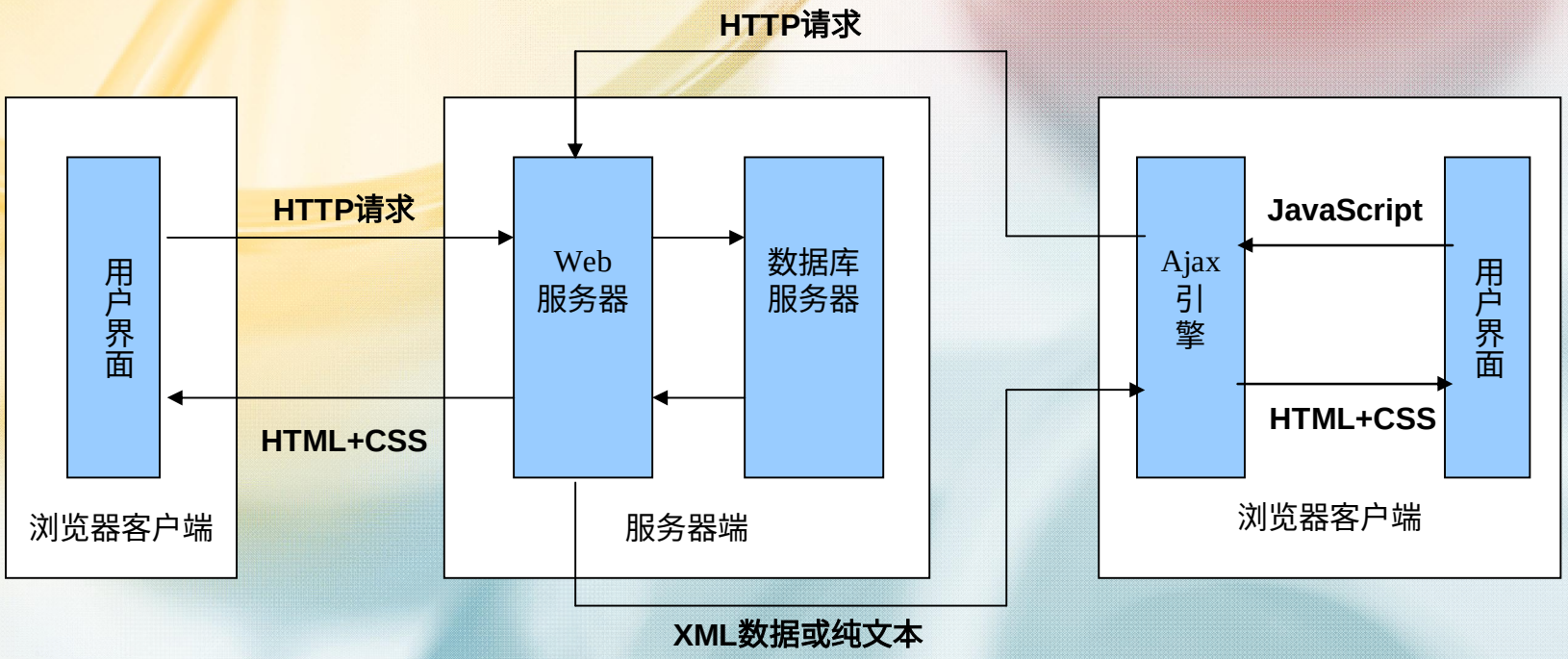
n 有代表性的方法

目前，国内关于Ajax性能优化已有的研究主要集中在：文献[3]**对基于Ajax的MVC模式进行改造**。文献[4]**提出一种基于JSON的对象序列化算法来解决解析XML所造成的缺陷**。文献[5]**对Ajax的首页加载模式进行改进**等。

文章要点

Ø 1 目前 Ajax性能优化的研究进展

- n Ajax的工作原理



传统的Web应用模式—同步交互方式

基于Ajax的Web应用模式—异步交互方式

文章要点

Ø 2 响应数据传输格式分析

n 2.1 背景知识

Ajax引擎可以用以下两种数据响应格式来获取从服务器返回的信息：一是**纯文本**，用XMLHttpRequest对象的responseText属性来获取；二是**XML文档**，用XMLHttpRequest对象的responseXML属性来获取。若使用纯文本格式，当前最常用的格式是JSON。用JSON和XML来返回响应数据各有利弊。

XML (**eXtensible Markup Language**)是一种类似于HTML的，用户自定义标签的标记语言。XML是目前Web数据表示较为通用的标准，兼容性较好。但当XML数据量大，结构复杂时，会在一定程度上影响服务器的响应速度和处理效率。因为若使用XML，我们不得不加上一些并不需要的冗余数据如自定义的**开始标签和结束标签**。

文章要点

Ø 2 响应数据传输格式分析

n 2.1 背景知识（续）

而JSON正是作为一种XML的替代品提出的。JSON (**JavaScript Object Notation**)是一种基于JavaScript的，比XML更简洁的轻量级数据交换格式，它采用**“名/值”序偶**和**值的有序列表**的形式，本质上是一个关联数组（associative array）或者一个哈希表（ hash table）。它作为一种**轻量级的数据交换格式**，具有简洁、数据量小以及便于解析等优点，受到了开发人员的欢迎。

如何在XML和JSON之间权衡，选用一个当前场景下最合适的响应数据载体，是优化一个Ajax应用程序需要考虑的问题。

文章要点

Ø 2 响应数据传输格式分析

n 2.2 用户可读性

XML采用用户自定义标签来包含数据，牺牲了存储空间，可文档显得整洁容易理解，换取了用户可读性。

JSON虽具有简洁的语法，但当用户面对一系列的冒号、逗号和大括号时，却显得有些茫然。（后面会有示例说明）

对于习惯了HTML和XML中一脉相承的开始结束标签的用户来讲，JSON的代码不是很容易阅读，像XML这种基于标签的语法，要更整洁一些。明显地，具有明确语义的开始和结束标签比起符号来更能展现文档的结构和含义。

但是在Ajax这个场景中，用XML和JSON表示的只是服务器返回的中间数据，对于用户是不可见的。当数据真正呈现在页面上时，使用XML和JSON最后的页面效果是一样的。因此，从优化Ajax的数据响应过程角度来说，两种数据响应格式对于用户的可读性是不需要考虑的。

文章要点

Ø 2 响应数据传输格式分析

n 2.3 数据量

下面通过一个示例：**在Google Suggest中输入查询字符串”compiler”后返回搜索建议**，来对两种数据响应格式作出对比：

```
<?xml version=”1.0” encoding=”utf-8”?>
<suggestions terms=”compiler”>
<suggestion term=”compiler compliance level” results=”451, 000 results” />
<suggestion term=”compiler design” results=”952, 000 results” />
<suggestion term=”compiler options” results=”253, 000 results” />
<suggestion term=”compiler optimization” results=”417, 000 results” />
<suggestion term=”compiler services” results=”51, 400 results” />
<suggestion term=”compiler course” results=”5, 400, 000 results” />
<suggestion term=”compiler conference” results=”5, 080, 000 results” />
<suggestion term=”compiler error” results=”1, 700, 000 results” />
<suggestion term=”compilers and compiler generators” results=”314, 000 results” />
</suggestions>
```

示例的XML表示

```
{"suggestions":{
"terms":"compiler",
"suggestion":[{
"term":"compiler compliance level","results":"451, 000 results"},{
"term":"compiler design","results":"952, 000 results"},{
"term":"compiler options","results":"253, 000 results"},{
"term":"compiler optimization","results":"417, 000 results"},{
"term":"compiler services","results":"51, 400 results"},{
"term":"compiler course","results":"5, 400, 000 results"},{
"term":"compiler conference","results":"5, 080, 000 results"},{
"term":"compiler error","results":"1, 700, 000 results"},{
"term":"compilers and compiler generators","results":"314, 000 results"}
]
}
}
```

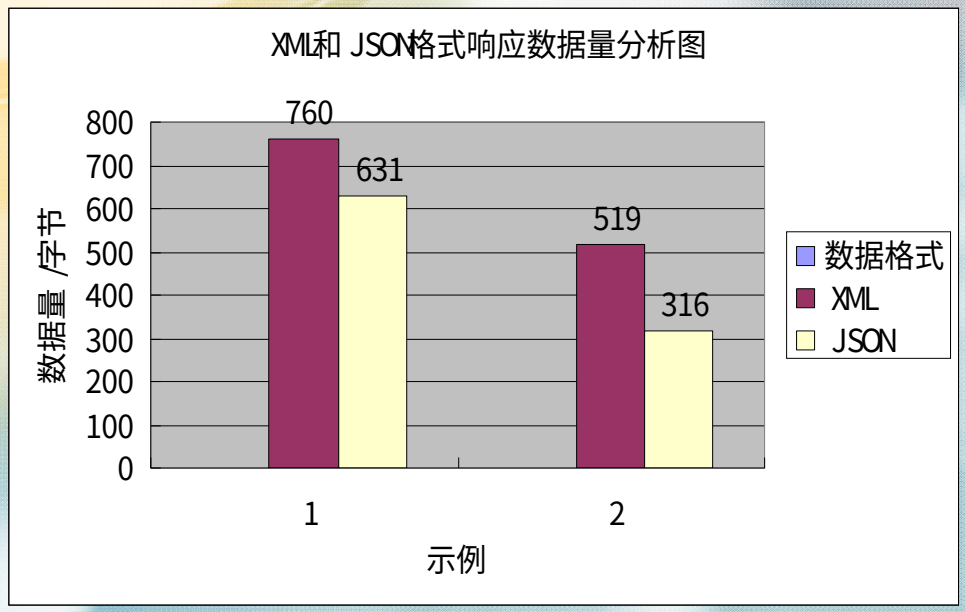
示例的JSON表示

文章要点

Ø 2 响应数据传输格式分析

n 2.3 数据量（续）

可得出下图：



文章要点

Ø 2 响应数据传输格式分析

n 2.3 数据量（续）

随着应用程序中的数据交换量的增长，数据结构的复杂化，JSON数据量小的优势将更为明显（如示例2）。这样客户端与服务器端进行数据交换时，占用的带宽就会尽可能小，应用程序就运行得更快、更有效率。

虽然表面上在一次请求中数据缩减量不大，但如果这样的并发请求数量众多，节省的数据量就有很大差别了。设想这样一个一天处理上万条并发请求的服务器，一个月所节省的系统资源开销是相当可观的。

文章要点

Ø 2 响应数据传输格式分析

n 2.4 客户端解析效率

JSON跟XML都是**结构化的数据交换格式**，两者的不同在于XML本身是**DOM树结构**的，需要**JavaScript操作DOM元素**来进行解析才能获取其中的数据，而且DOM在各个浏览器中的实现也不尽相同，所以针对XML DOM的编程会变的更为复杂。而JSON本身就是JavaScript，因此不需要Ajax引擎来解析，一旦**调用JavaScript的eval()方法将JSON字符串序列化成为JavaScript对象**之后，就可以**直接读取其属性**来获取数据。相比之下，访问和处理数据时JSON比XML要快捷许多。

下面通过一个简单的测试程序来**计算代码的执行时间**，从而比较两种格式读取数据时的解析效率。

文章要点

Ø 2 响应数据传输格式分析

n 2.4 客户端解析效率（续）

```
var t1 = new Date().getTime();
for(var i=0; i<10000; i++) {
  var data = request.responseXML;
  var name = data.getElementsByTagName
          ("name")[0].firstChild.nodeValue;
  var website = data.getElementsByTagName
          ("website")[0].firstChild.nodeValue;
  var email = data.getElementsByTagName
          ("email")[0].firstChild.nodeValue;
}
var t2 = new Date().getTime();
alert(t2-t1);
```

解析XML中响应数据的代码

```
var t1 = new Date().getTime();
for(var i=0; i<10000; i++) {
  var data = eval('('+request.responseText+')');
  var name = data.person.name;
  var email = data.person.email;
  var website = data.person.website;
}
var t2 = new Date().getTime();
alert(t2-t1);
```

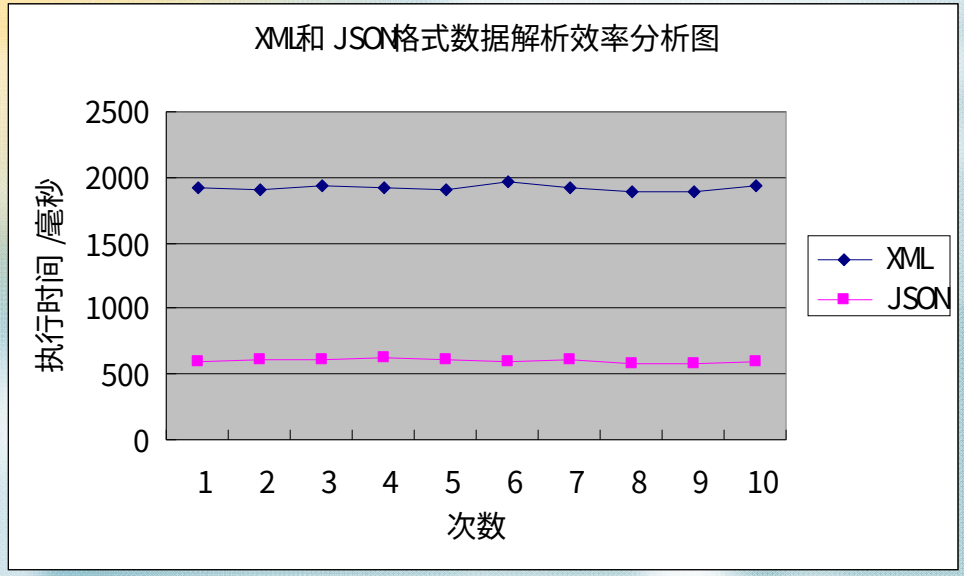
解析JSON中响应数据的代码

文章要点

Ø 2 响应数据传输格式分析

n 2.4 客户端解析效率（续）

在上面两段分别解析XML和JSON中响应数据的代码中，for循环中的代码是两者各自的解析操作，在解析操作代码段的开始和结束分别使用了JavaScript中的getTime()函数来记录时间，最后计算差值，即代码的运算时间。**为消除偶然误差，重复试验（10次）**可统计出下图：



文章要点

Ø 2 响应数据传输格式分析

n 2.5 服务器端开发效率

对于**如何将一段数据序列化为一个XML文档**，各种服务器端编程语言都有提供多种方式来实现，如.NET框架下，C#中的XmlSerializer类，通过结合TextWriter类，它可以序列化一个对象成为XML格式。在Java中则提供了XmlStreamWriter和XMLEncoder类以及API函数SAX来处理普通对象到XML的转换。另外，在PHP以及.NET等多种开发环境下，开发人员都可以使用XmlWriter类等工具来直接生成XML文档。

而对于JSON的自动生成支持工具，目前来讲还比较少，开发人员可能需要自己手动编写一段代码或者使用一些零散的开源类库来完成这项工作。**而且对于较为复杂的数据，作为一种新兴的数据交换格式，JSON的生成效率和稳定性会比XML差一些。**

因此，从服务器端开发效率上来讲，较早出现并且更加标准化、规范化的XML**目前**会比JSON有更好的表现。

文章要点

Ø 2 响应数据传输格式分析

n 2.6 安全分析及优化

JSON承载的只是数据，不会含有赋值和调用，所以它是安全中立的，即JSON本身并不会引入安全隐患。而当开发人员用eval()函数把JSON数据作为JavaScript代码执行，从而转化为JavaScript对象时，可能会带来意想不到的安全性问题。攻击者可以在JSON数据中携带恶意的JavaScript代码发送给客户端，这样eval()函数就会执行这些恶意代码，这就存在相当大的风险，系统可能会因此而崩溃。

JSON本质上就是JavaScript - 代码混淆器

因此，如果使用JSON作为数据载体，必须确保JSON是安全的。一方面，可以通过口令验证来保障JSON数据来源的可信度，即读取的响应数据不是来自于某个不可靠的第三方；另一方面，可以在客户端使用正则表达式来检查JSON数据是否包含有恶意代码关键字。从而降低系统遭受攻击的可能性。XML由于解析时不含有任何本地执行过程，因此相对JSON来讲更安全一些。

文章要点

Ø 2 响应数据传输格式分析

n 2.7 权衡与取舍

在以上的讨论中，本文从各个角度分析了XML和JSON各自的优劣。根据上面的分析，我们可以列出下表作一个总结：

数据格式	用户可读性	数据量	客户端解析效率	服务器端开发效率	安全性
XML	好 (N/A)	大	低	高	较好
JSON	一般	小	高	低	差

文章要点

Ø 2 响应数据传输格式分析

n 2.7 权衡与取舍（续）

于是，针对**具体的开发场景**，可以给出如下选取方案：

在开发一个基于Ajax技术的Web应用程序时，在**安全性要求不高及服务器处理能力较强**的场景下，选用JSON更好；在**用户体验要求不高的安全敏感**场景下，选用XML较为合适；在**用户体验和安全性要求都较高**的场景下，应从大局着眼，选用较为安全的XML而牺牲一部分系统性能。

总之，开发人员需要认真评估**在不同场景下两种响应数据表示方式的成本和效率**，了解两者的差异后，再来根据实际需要进行合理选择，或者**直接采用页面重载刷新的方式而不是Ajax**。

内容提要

回顾

文章概述

文章要点

总结与展望

提问

总结与展望

Ø 为什么要使用 Ajax

Ajax是一种很强大的创建类桌面应用程序的Web应用开发工具，可以在不需要刷新页面的情况下进行异步的后台数据库交互，从而在不会打断用户在应用程序中其他操作的同时更新页面内容，大大提高了Web应用程序的响应速度。

Ø 为什么要优化 Ajax的数据响应过程

作为开发一个成熟软件的角度来讲，如何高效地运行Ajax程序越来越被重视。首先面临的是响应数据传输的问题，我们需要根据实际情况选择数据响应的返回格式，从而使服务器和客户端负载都尽量达到最小化。

能否高效地利用系统资源，这是在富互联网应用（Rich Internet Application, RIA）日渐发达的今天，开发人员面临的一大考验。我们希望看到的是，在不改变浏览器机制和用户习惯的前提下，Web访问者更好地体验Ajax带来的丰富动态功能，就像使用桌面应用程序那样。

总结与展望

Ø 本文的不足之处

尚存的不足之处在于提出的部分观点尚无法给出量化的模型来衡量，有的只能通过主观经验来判断，如怎样根据安全隐患的严重程度来决定选用XML还是JSON。因此，从理论化的角度来讲，还需要进一步的研究。

Ø 为什么上不了正刊 – 创新性不够+推广价值不高

《计算机学报》、《软件学报》和《计算机研究与发展》 - **null**

《计算机科学》 - **null**

《计算机应用》 - 4篇刊用，1正刊(2007-09)，3增刊(2007-S1)

《计算机工程》 - 3篇刊用，3正刊(2007-24), (2008-07), (2008-19)

因此，要想在高水平的期刊上发表文章，理论深度是一定要有的，经得起推敲的东西才是有价值的，如算法、数学证明以及形式化推导。

总结与展望

Ø 参考文献

1. GARRETT J J. Ajax: A New Approach to Web Applications[EB/OL], http://www.adaptivepath.com/ideas/essays/archives/000385.php, 2005-02-18.
2. CROCKFORD D. JSON and Browser Security[EB/OL], http://yuiblog.com/blog/2007/04/10/json-and-browser-security, 2007-4-10.
3. 王东，孙彬. 基于Ajax的MVC框架的改造分析[J]. 计算机应用，2007，27(S1)：293-295.
4. 张涛，黄强，等. 一个基于JSON 的对象序列化算法[J]. 计算机工程与应用，2007，43(15)：98-100.
5. 阳锋，徐建波. AJAX技术的性能改进研究[J]. 计算机工程与科学，2008，30(6)：146-148.
6. HADLOCK K. Ajax——Web开发、可重用组件及模式[M]. 叶俊 译. 北京：清华大学出版社，2007.
7. KEITH J. Bulletproof Ajax中文版[M]. 刘申，宋薇 译. 北京：人民邮电出版社，2007.
8. 汤代禄. 互联网的变革--Web 2.0理念与设计[M]. 北京：电子工业出版社，2007.

谢谢！