开始

翻译正在进行中。

本文将指导您完成AJAX基础知识，并为您提供两个简单的实践示例，帮助您开始使用。

什么是 AJAX?

AJAX代表异步JavaScript和XML。简而言之，它是使用 XMLHttpRequest 对象与服务器端脚本进行通信。它可以发送以及接收各种格式的信息，包括JSON，XML，HTML，甚至文本文件。AJAX最吸引人的特性是，它的“异步”性质，这意味着它可以做所有这一切，而不必刷新页面。这允许您根据用户事件更新页面的某些部分。

AJAX的两个主要功能允许您执行以下操作：

- 向服务器发出请求，而不重新加载页面
- 接收和处理服务器中的数据

步骤 1 – 如何发出HTTP请求

为了使用JavaScript向服务器发出 HTTP 请求，您需要一个提供此功能的类的实例。这就是XMLHttpRequest的地方。这样的类最初是在Internet Explorer中作为一个名为XMLHTTP的ActiveX对象引入的。然后，Mozilla，Safari和其他浏览器，实现一个XMLHttpRequest类，支持Microsoft的原始ActiveX对象的方法和属性。同时微软也实现了XMLHttpRequest。

```
// Old compatibility code, no longer needed.
if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+ ...
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 and older
```

```
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}

// new
const httpRequest = new XMLHttpRequest();
```

注意：为了说明的目的，上面是用于创建XMLHTTP实例的代码的一个简化版本。有关更实际的示例，请参阅本文的步骤3。

接下来，您需要决定在收到服务器对您的请求的响应后您想要做什么。在这个阶段，你只需要告诉HTTP请求对象哪个JavaScript函数将处理响应。这是通过将对象的onreadystatechange属性设置为当请求的状态更改时应调用的JavaScript函数的名称，如下所示：

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

注意，函数名后面没有括号，没有参数传递，因为你只是给函数赋了一个引用，而不是实际调用它。此外，您可以使用JavaScript技术来动态定义函数（称为"匿名函数"），并定义将立即处理响应的操作，而不是给出函数名称，如下所示：

```
httpRequest.onreadystatechange = function(){
    // 处理服务器响应
};
```

接下来，在声明接收到响应后会发生什么，您需要实际提出请求。您需要调用HTTP请求类的**open( )** 和 **send( )** 方法，如下所示：

```
httpRequest.open('GET', 'http://www.example.org/db.json', true);
httpRequest.send(null);
```

- 调用open（）的第一个参数是HTTP请求方法 - GET，POST，HEAD或任何其他您要使用的服务器支持的方法。保持方法大写按照HTTP标准;否则某些浏览器（如Firefox）可能不会处理请求。有关可能的HTTP请求方法的更多信息，您可以检查 W3C规范。

- 第二个参数是您请求的网页的网址。作为安全功能，您不能调用第三方网域上的网页。确保在所有网页上使用确切的域名，否则当您调用open（）时，您将收到“权限被拒绝”错误。一个常见的陷阱是通过domain.tld访问您的网站，但尝试调用www.domain.tld的网页。如果真的需要向另一个域发送请求，请参阅 HTTP访问控制。
- 可选的第三个参数设置请求是否是异步的。如果为TRUE (默认值)，则JavaScript函数的执行将继续，而服务器的响应尚未到达。这是AJAX中的A.

send( ) 方法的参数可以是任何要发送到服务器的数据，如果POST请求。表单数据应以服务器可以轻松解析的格式发送。这可以作为查询字符串，如：

```
"name=value&anothername="+encodeURIComponent(myVar)+"&so=on"
```

或以其他几种格式，包括JSON，SOAP等。

注意，如果你想POST数据，你可能必须设置请求的MIME类型。例如，在对作为查询字符串发送的表单数据调用 send() 之前，请使用以下行：

```
// setRequestHeader & urlencoded

httpRequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

步骤 2 – 处理服务器响应

请记住，当您发送请求时，您提供了用于处理响应的JavaScript函数的名称。

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

让我们看看这个函数应该做什么。首先，该函数需要检查请求的状态。如果状态的值为XMLHttpRequest.DONE（计算为4），这意味着已收到完整的服务器响应，您可以继续处理它。

```
if (httpRequest.readyState === XMLHttpRequest.DONE) {
    // 一切都很好，收到回复
} else {
    // 还没准备好
}
```

readyState值的完整列表记录在 XMLHTTPRequest.readyState，如下所示：

- 0 (未初始化)
- 1 (正在加载)
- 2 (加载完毕)
- 3 (交互中)
- 4 (完成)

(Source)

接下来要检查的是HTTP服务器响应的响应代码。所有可能的代码都列在W3C网站上。在下面的示例中，我们通过检查200 OK响应代码来区分成功或失败的AJAX调用。

```
if (httpRequest.status === 200) {
    // 完美!
} else {
    // 有一个问题的请求
    //例如响应可能包含404（找不到）
    //或500（内部服务器错误）响应代码
}
```

现在，在检查请求的状态和响应的HTTP状态代码后，您可以根据需要对服务器发送给您的数据进行任何操作。您有两个选项可以访问该数据：

- `http_request.responseText` – 以文本字符串形式返回服务器响应
- `http_request.responseXML` – 将响应作为XMLDocument对象返回，您可以使用JavaScript DOM函数遍历

请注意，上述步骤仅在使用异步请求（open（）的第三个参数设置为true）时有效。如果使用同步请求，则不需要指定函数，可以在调用send（）后立即访问服务器返回的数据，因为脚本将停止并等待服务器应答。

步骤 3 – 一个简单的例子

Let's put it all together and do a simple HTTP request. Our JavaScript will request an HTML document, `test.html`, which contains the text "I'm a test." and then we'll `alert()` the contents of the `test.html` file. Note that this example uses only JavaScript (no jQuery is involved here). Also, the HTML, XML and PHP files should be placed in the same directory.

```
<span id="ajaxButton" style="cursor: pointer; text-decoration: underline">
  Make a request
</span>
<script type="text/javascript">
(function() {
  var httpRequest;
  document.getElementById("ajaxButton").onclick = function() { makeRequest('test.html'); };

  function makeRequest(url) {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
      alert('Giving up :( Cannot create an XMLHTTP instance');
      return false;
    }
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', url);
    httpRequest.send();
  }

  function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText);
      } else {
        alert('There was a problem with the request.');
      }
    }
  }
})();
</script>
```

In this example:

- The user clicks the link "Make a request" in the browser;
- The event handler calls the `makeRequest()` function with a parameter – the name `test.html` of an HTML file in the same directory;
- The request is made and then (`onreadystatechange`) the execution is passed to `alertContents()`;
- `alertContents()` checks if the response was received and it's an OK and then `alert()`s the contents of the `test.html` file.

Note: If you're sending a request to a piece of code that will return XML, rather than to a static XML file, you must set some response headers if your page is to work in Internet Explorer in addition to Mozilla. If you do not set header `Content-Type: application/xml`, IE will throw a JavaScript error, "Object Expected", after the line where you try to access an XML element.

Note 2: If you do not set header `Cache-Control: no-cache` the browser will cache the response and never re-submit the request, making debugging "challenging." You can also append an always-different additional GET parameter, like the timestamp or a random number (see bypassing the cache)

Note 3: If the `httpRequest` variable is used globally, competing functions calling `makeRequest()` may overwrite each other, causing a race condition. Declaring the `httpRequest` variable local to a closurecontaining the AJAX functions prevents the race condition.

In the event of a communication error (such as the webserver going down), an exception will be thrown in the onreadystatechange method when attempting to access the status field. To mitigate this problem, you could wrap your if...then statement in a try...catch:

```
function alertContents() {
  try {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText);
      } else {
        alert('There was a problem with the request.');
      }
    }
  }
  catch( e ) {
    alert('Caught Exception: ' + e.description);
  }
}
```

步骤 4 – 使用XML响应

In the previous example, after the response to the HTTP request was received we used the `responseText` property of the request object, which contained the contents of the `test.html` file. Now let's try the `responseXML` property.

First off, let's create a valid XML document that we'll request later on. The document (`test.xml`) contains the following:

```
<?xml version="1.0" ?>
<root>
    I'm a test.
</root>
```

In the script we only need to change the request line to:

```
...
onclick="makeRequest('test.xml')">
...
```

Then in `alertContents()`, we need to replace the line `alert(httpRequest.responseText);` with:

```
var xmldoc = httpRequest.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```

This code takes the `XMLDocument` object given by `responseXML` and uses DOM methods to access some of the data contained in the XML document. You can see the `test.xml` here and the updated test script here.

步骤 5 – 使用数据

Finally, let's send some data to the server and receive a response. Our JavaScript will request a dynamic page this time, `test.php`, which will take the data we send and return a "computed" string - "Hello, [user data]!" - which we'll `alert()`.

First we'll add a text box to our HTML so the user can enter their name:

```
<label>Your name:
  <input type="text" id="ajaxTextbox" />
</label>
<span id="ajaxButton" style="cursor: pointer; text-decoration: underline">
  Make a request
</span>
```

We'll also add a line to our event handler to get the user's data from the text box and send it to the `makeRequest()` function along with the URL of our server-side script:

```
  document.getElementById("ajaxButton").onclick = function() {
      var userName = document.getElementById("ajaxTextbox").value;
      makeRequest('test.php',userName);
  };
```

We need to modify `makeRequest()` to accept the user data and pass it along to the server. We'll change the request method from `GET` to `POST`, and include our data as a parameter in the call to `httpRequest.send()`:

```
  function makeRequest(url, userName) {

    ...
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('POST', url);
    httpRequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    httpRequest.send('userName=' + encodeURIComponent(userName));
  }
```

The function `alertContents()` can be written the same way it was in Step 3 to alert our computed string, if that's all the server returns. However, let's say the server is going to return both the computed string and the original user data. So if our user typed "Jane" in the text box, the server's response would look like this:

```
{"userData":"Jane","computedString":"Hi, Jane!"}
```

To use this data within `alertContents()`, we can't just alert the `responseText`, we have to parse it and alert `computedString`, the property we want:

```
function alertContents() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    if (httpRequest.status === 200) {
      var response = JSON.parse(httpRequest.responseText);
      alert(response.computedString);
    } else {
      alert('There was a problem with the request.');
    }
  }
}
```

The test.php file should contain the following:

```
$name = (isset($_POST['userName'])) ? $_POST['userName'] : 'no name';
$computedString = "Hi, " . $name;
$array = ['userName' => $name, 'computedString' => $computedString];
echo json_encode($array);
```

For more on DOM methods, be sure to check Mozilla's DOM implementation documents.