Asynchronous JavaScript and XML: AJAX

Introduction

With today's internet being measured in milliseconds, any delay is looked at suspiciously. If a webpage does not load immediately, a user is going to begin assuming that the website is poorly constructed and possibly not worth their time. Asynchronous JavaScript and XML, AJAX, is a set of web development techniques that enable websites and web applications to send and receive data asynchronously, thus allowing it to be done in the background and not interfering with the rest of the webpage. The user can interact with one portion of a webpage and not deal with the entire page responding or reloading. Instead, just that one section responds, meaning a quicker, smoother experience and less movement on the webpage.

AJAX is not a programming language, nor is it a new technology. AJAX combines several standards, including JavaScript, HTML, CSS, and XML or JSON. Modern web developers have favored JSON more due to its lighter, compact nature. HTML and CSS can be used to mark up and style the page while it is dynamically displayed using JavaScript. An XMLHttpRequest object executes AJAX and allows the user to interact with the content without refreshing the page by manipulating data asynchronously with the web server.

Brief History and Evolution of AJAX in Web Development

The origins of AJAX date back to the late 1990s. In the early 1990's, every user action on a website required a completely new page to be loaded from the server. The entire page would disappear and must reload. In those days, with dial-up connections, this meant lengthy waits for the average user any time a partial change was made to the page.

With the introduction of Microsoft's XMLHttpRequest object in Internet Explorer 5. This was the fundamental technology that allowed content on a web page to update dynamically without the need to reload the entire page. However, AJAX only gained widespread attention in 2005, when Jesse James Garrett published an article in which he coined the term "AJAX".

Initially, AJAX was primarily used for small-scale, dynamic updates within web pages, such as refreshing a weather forecast or validating form data. The early adoption of AJAX can be seen in Google products like Google Maps and Gmail, which showcased the potential of AJAX for creating rich, interactive web applications.

Over the years, AJAX has evolved significantly. The reliance on XML has decreased, with many developers now favoring JSON (JavaScript Object Notation) due to its lightweight nature and compatibility with JavaScript. AJAX has become a fundamental part of Web 2.0, a term that denotes a new era in web development characterized by greater user interactivity, collaboration, and information sharing.

Modern web development frameworks and libraries, such as jQuery, AngularJS, and React, have simplified AJAX implementation, allowing for more dynamic and responsive web applications. These technologies have expanded the capabilities of AJAX, making it an essential component in the toolbox of web developers.

As web applications grow in complexity and functionality, AJAX plays a crucial role in enhancing the user experience by enabling seamless interactions between the user and the server. Its asynchronous nature ensures that web applications are faster, more efficient, and provide a more fluid user experience.

Traditional Form Submission Using HTML Post

When you create a form in HTML, you typically define two important attributes: "action" and "method." The "action" attribute specifies the URL to which the form data should be sent when the user submits the form, and the "method" attribute specifies the HTTP method to be used for the submission, which can be either "GET" or "POST." When the “method” is set to “POST”, the form data is sent in the request body, which is not visible in the address bar and is typically used for sensitive or large data.

The “action” attribute specifies the URL of the server-side script or resource that will handle the form data. When a user submits a form, the browser sends an HTTP request to the URL. For example, if you have a contact form, the “action” should take the input data, send it to a server-side script for validation, and give feedback on whether the data was accepted or rejected.

When a user submits a form with the standard HTML form submission using “action” and “method”, the entire page is usually reloaded when the server sends back a new HTML page to the browser. The user is unable to interact with the page during HTTP transport. This can result in a noticeable delay. When the page reloads, any unsaved changes or form data entered on the page is lost, creating a potentially frustrating experience for the user.

AJAX Form Submission

AJAX allows for form submissions without requiring a full page reload, making the user experience much smoother and more interactive. The form data can be captured with JavaScript, using an event listener that waits for the submit button to be clicked and sent to the server in the background using XMLHttpRequest or the Fetch API, updating the current page based on the server’s response. The rest of the webpage does not have to change, as it is not fully reloaded, and the user stays on that page while waiting for a response from the server. The form data can be kept in the form, waiting for the user to correct it and resubmit the data, clear it, or navigate away from the page after a successful submission.

Action and Post Attributes in AJAX

In AJAX, you are not required to use the “action” and “method” attributes because you can grab submissions with JavaScript, and you can specify the URL and HTTP method in your AJAX request. However, to be clear, you are still specifying the HTTP method as “POST” inside your JavaScript Code. AJAX operations require you to provide detailed configuration for the request, including the HTTP method, target URL, headers, and data to be sent.

For example:

```
    XHR.open('POST', 'process.php');
    XHR.send(formData);
```

The first line of that code specifies the ‘POST’ method and names a file named ‘process.php’ as the destination URL. The second line then sends the captured form data onto the PHP file to begin processing.

Another example using jQuery code:

```
    $.ajax({
        URL: 'email.php',
        type: 'POST',
        data: {
            name: nameInput,
            from: fromInput,
            confirm: confirmEmailInput,
            subject: subjectInput,
            message: messageInput
          },
```

Here the URL is ‘email.php’ shown in the second line and the third line qualifies the ‘POST’ method as the “type” used and the data is given inside the post rather than using data encapsulated in a variable. So, while you do not need the “action” and “method” attributes to be in your HTML Form, you will still be explicitly specifying the information that those two attributes normally declare, you are just doing it in your JavaScript AJAX operation instead of in your HTML.

AJAX Operating Without HTML Forms

We have discussed how AJAX can use form elements as data sources, easily gathering user input through the webpage form. The forms provide a structured way for the input to be collected and sent asynchronously to the server. Forms also allow for client-side validation before AJAX ever sends the data, enhancing the user experience and potentially reducing the load on the server. However, AJAX can send and receive data that is not tied to a form element.

AJAX can send and receive JSON objects, text, or XML. This makes AJAX useful for a multitude of situations where data is generated or retrieved from other sources. AJAX can also send and receive data through JavaScript without user interaction, meaning developers have more dynamic and complex options when making their webpages. AJAX can also be integrated with custom user interface components that do not rely on traditional form elements. An example could be a situation where a user needs to choose a state or country from a map, perhaps for a destination, and an AJAX component could take that choice and present options available in that region. Text messages can be inserted into webpages as messages to the user based on their interactions.

AJAX Within an MVC Framework

MVC, which stands for Model-View-Controller, is a design pattern in web development that helps separate the logic of different layers in a web application, making it more modular and easier to manage.

1. **Model**: Represents the data and the application's business logic. It's responsible for accessing the database, processing data, and defining the business rules.
2. **View**: The user interface of the application. Views display data from the model to the user and send user commands to the controller.
3. **Controller**: Acts as an intermediary between Model and View. It listens to user inputs, processes them (possibly with the help of Model), and returns the output display (View).

AJAX within an MVC framework is a combination that can leverage the strengths of both technologies to build responsive, dynamic web applications.

In an MVC framework:

- **AJAX with View**: AJAX can be used to update the View dynamically. For instance, fetching data and refreshing part of the view without a full page reload.
- **AJAX with Controller**: AJAX requests are sent to the controller. The controller processes these requests and can return data in various formats (like JSON or XML) instead of a full-fledged view.
- **AJAX with Model**: AJAX in MVC usually interacts indirectly with the Model through the Controller. This means AJAX requests often trigger controller actions that then manipulate the Model.

Considerations When Using AJAX Within an MVC Framework

Integrating AJAX within an MVC framework offers a structured and efficient way to manage dynamic content and user interactions. However, it requires a thoughtful approach to align with

the MVC architecture's principles. The following are some considerations when using AJAX with an MVC architecture:

- **Separation of Concerns:** maintaining a clear separation of concerns is crucial. Ajax calls should be managed in a way that respects this principle, ensuring that business logic, data manipulation, and UI updates remain distinct.
- **Routing and URL Management:** MVC frameworks often have specific routing mechanisms. AJAX calls within such a framework must align with this routing system, directing requests to the correct controller actions.
- **Security Considerations:** MVC frameworks often come with built-in security features. When implementing AJAX, it's important to leverage these features to protect against common web vulnerabilities like CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting).
- **Handling Responses and Errors:** MVC frameworks may provide specific ways to handle AJAX responses and errors. For instance, returning JSON data or HTTP status codes in a standardized format that AJAX callbacks can easily interpret.
- **Asynchronous Data Binding:** Some MVC frameworks have options for binding data asynchronously. AJAX can be used to fetch data and update the bound elements in the View.
- **Unit Testing and Debugging:** Debugging and testing AJAX within an MVC framework can be more complex, as it might involve testing across different layers (Controller, View, and sometimes Model) and handling asynchronous behaviors.
- **Performance Optimization:** In an MVC context, AJAX should be used judiciously to ensure that it enhances rather than hinders performance. This includes optimizing server-side actions called by AJAX and minimizing unnecessary requests.

Conclusion

AJAX has been a pivotal technology in shaping the landscape of modern web development. Its introduction shifted the manner web applications are built and how users interact with them, setting new standards for user experience. The demand for fast, dynamic, agile, and interactive web applications will only grow, and AJAX is a leading technique in meeting these needs. Its integration with emerging technologies and its role in advancing web development will continue to push growth and innovation. AJAX has become a fundamental element in modern web development, integral to creating responsive and seamless user experiences. It is a crucial component in various web development frameworks and libraries, underscoring its importance in the industry.