مؤسسه فرهنگی هنری
دیباگران تهران

آموزش کاربردی

AJAX

(Asynchronous JavaScript and XML)

روزبه امیرعصامی

به نام خدا

آموزش کاربردی

AJAX

مؤلف:

روزبه امیر عصامی

هرگونه چاپ و تکثیر از محتویات این کتاب بدون اجازه کتبی ناشر ممنوع است. متخلفان به موجب قانون حمایت حقوق مؤلفان، مصنفان و هنرمندان تحت پیگرد قانونی قرار می‌گیرند.

عنوان کتاب: آموزش کاربردی AJAX

مولف : روزبه امیرعصامی

ناشر: موسسه فرهنگی هنری دیباگران تهران

ویراستار: مهدیه مخبری

صفحه آرایی:نازنین نصیری

طراح جلد:داریوش فرسایی

نوبت چاپ: اول

تاریخ نشر: ۱۴۰۲

چاپ و صحافی:صدف

تیراژ:۱۰۰ جلد

شابک: ۸-۶۶۷-۲۱۸-۶۲۲-۹۷۸

نشانی واحد فروش:تهران، خیابان انقلاب،خیابان دانشگاه -تقاطع شهدای ژاندارمری-پلاک ۱۵۸ساختمان دانشگاه- طبقه دوم-واحد۴ تلفن ها: ۶۶۹۶۵۷۴۹-۲۲۰۸۵۱۱۱

فروشگاههای اینترنتی دیباگران تهران :

WWW.MFTBOOK.IR

www.dibagarantehran.com

سرشناسه:امیرعصامی،روزبه،۱۳۶۱-

عنوان و نام پدیدآور:آموزش کاربردی AJAX/مولف:روزبه امیرعصامی؛

ویراستار:مهدیه مخبری.

مشخصات نشر: تهران : دیباگران تهران :۱۴۰۱

مشخصات ظاهری:۱۱۴ ص:مصور،

شابک: ۸-۶۶۷-۲۱۸-۶۲۲-۹۷۸

وضعیت فهرست نویسی: فیپا

موضوع:ای جکس(تکنولوژی توسعه وبگاه)

موضوع:(Ajax(website development technology

موضوع:وبگاه ها-توسعه web site development

شناسه افزوده: مخبری،مهدیه،۱۳۶۹-ویراستار

رده بندی کنگره:۵۱۰۵/۸۸۸۵ TK

رده بندی دیویی:۰۰۶/۷۶

شماره کتابشناسی ملی:۹۱۳۵۶۶۳

نشانی تلگرام:mftbook@ **نشانی اینستاگرام دیبا dibagaran_publishing**

هر کتاب دیباگران، یک فرصت جدید علمی و شغلی.

هر گوشی همراه، یک فروشگاه کتاب دیباگران تهران.

از طریق سایتهای دیباگران، در هر جای ایران به کتابهای ما دسترسی دارید.

فهرست مطالب

مقدمه ناشر

خط مشی انتشارات مؤسسه فرهنگی هنری دیباگران تهران در عرصه کتاب هایی با کیفیت عالی است که بتواند خواسته های به روز جامعه فرهنگی و علمی کشور را تا حد امکان پوشش دهد.

هر کتاب دیباگران تهران، یک فرصت جدید شغلی و علمی

حمد و سپاس ایزد منان را که با الطاف بیکران خود این توفیق را به ما ارزانی داشت تا بتوانیم در راه ارتقای دانش عمومی و فرهنگی این مرز و بوم در زمینه چاپ و نشر کتب علمی و آموزشی گام‌هایی هرچند کوچک برداشته و در انجام رسالتی که بر عهده داریم، مؤثر واقع شویم.

گستردگی علوم وسرعت توسعه روزافزون آن، شرایطی را به وجود آورده که هر روز شاهد تحولات اساسی چشمگیری در سطح جهان هستیم. این گسترش و توسعه، نیاز به منابع مختلف از جمله کتاب را به عنوان قدیمی‌ترین و راحت‌ترین راه دستیابی به اطلاعات و اطلاع‌رسانی، بیش از پیش برجسته نموده است.

در این راستا، واحد انتشارات مؤسسه فرهنگی هنری دیباگران تهران با همکاری اساتید، مؤلفان، مترجمان، متخصصان، پژوهشگران و محققان در زمینه های گوناگون و مورد نیاز جامعه تلاش نموده برای رفع کمبودها و نیازهای موجود، منابعی پُربار، معتبر و با کیفیت مناسب در اختیار علاقمندان قرار دهد.

کتابی که در دست دارید تألیف **"جناب آقای روزبه امیرعصامی"** است که با تلاش همکاران ما در نشر دیباگران تهران منتشرگشته و شایسته است از یکایک این گرامیان تشکر و قدردانی کنیم.

با نظرات خود مشوق و راهنمای ما باشید

با ارائه نظرات و پیشنهادات وخواسته های خود،به ما کمک کنید تا بهتر و دقیق تر در جهت رفع نیازهای علمی و آموزشی کشورمان قدم برداریم.برای رساندن پیام هایتان به ما از رسانه های دیباگران تهران شامل سایتهای فروشگاهی و صفحه اینستاگرام و شماره های تماس که در صفحه شناسنامه کتاب آمده استفاده نمایید.

مدیر انتشارات

مؤسسه فرهنگی هنری دیباگران تهران
dibagaran@mftplus.com

فصل اول

آشنایی با AJAX

Ajaxچیست؟

Ajax مخفف Asynchronous Javascript and Xml است. Ajax فقط وسیله‌ای برای بارگذاری داده‌ها از سرور و به‌روزرسانی بخش‌هایی از یک صفحه وب بدون بارگیری مجدد کل صفحه است.

اساساً، کاری که Ajax انجام می‌دهد. استفاده از شیء XMLHttpRequest (XHR) داخلی مرورگر برای ارسال و دریافت اطلاعات به یک وب سرور به‌صورت ناهمزمانی می‌باشد که در پس‌زمینه، بدون مسدودکردن صفحه یا تداخل با تجربه کاربر صورت می‌گیرد.

Ajax به قدری محبوب شده است که به سختی برنامه‌ای را پیدا می‌کنید که از Ajax استفاده نکند هرچند کم. نمونه‌ای از برخی از برنامه‌های آنلاین در مقیاس بزرگ که توسط Ajax هدایت می‌شوند عبارت‌اند از: Gmail، Google Maps، Google Docs، YouTube، Facebook، Flickr و بسیاری از برنامه‌های کاربردی دیگر.

توجه: Ajax یک فناوری جدید نیست، درواقع، Ajax حتی واقعاً یک فناوری نیست. Ajax فقط یک اصطلاح برای توصیف فرایند تبادل داده‌ها از یک وب سرور به‌صورت ناهمزمان از طریق جاوا اسکریپت، بدون رفرش کردن دوباره صفحه است.

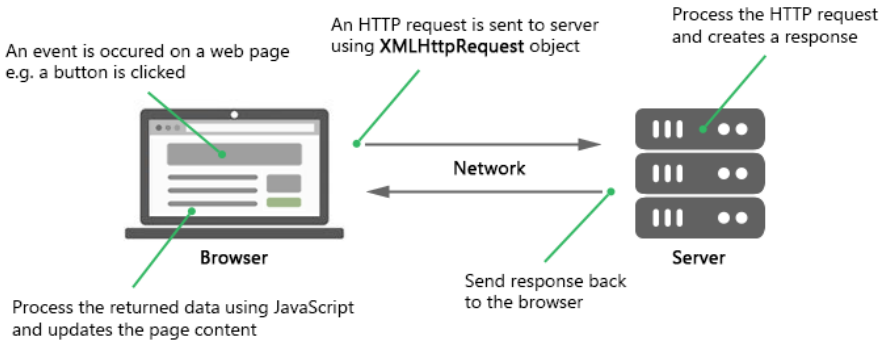
☑ **نکته:** عبارت X (یعنی XML) در AJAX اشتباه نگیرید. فقط دلایل تاریخی وجود دارد. سایر فرمت‌های تبادل داده مانند JSON، HTML یا متن ساده را می‌توان به‌جای XML استفاده کرد.

آشنایی با نحوهٔ عملکرد Ajax (ایجکس)

برای انجام ارتباطAjax ، جاوا اسکریپت از یک شیء ویژه تعبیه شده در مرورگر استفاده می‌کند. یک شی XMLHttpRequest (XHR)برای ایجاد درخواست‌های HTTP به سرور و دریافت داده‌ها در پاسخ.

همه مرورگرهای مدرن (Chrome، Firefox، IE7+، Safari، Opera) از شیء XMLHttpRequest پشتیبانی می‌کنند.

تصاویر زیر نحوهٔ عملکرد ارتباط Ajax را نشان می‌دهد:



نحوهٔ عملکرد Ajax

از آنجایی که درخواست‌های Ajax معمولاً ناهمزمان هستند، اجرای اسکریپت به محض ارسال درخواست Ajax ادامه می‌یابد، یعنی مرورگر اجرای اسکریپت را متوقف نمی‌کند تا زمانی که پاسخ سرور بازگردد.

نحوهٔ کارAJAX

کاربر یک رویداد را روی صفحه وب فعال می‌کند، مانند کلیک روی یک باتن.

HTTPRequest (درخواستHTML) با استفاده از شیء XMLHTTPRequest به سرور ارسال می‌شود. این شی با پارامتر درخواست روی شبکه پیکربندی شده است.

XMLHTTPRequestیک درخواست ناهمزمان به سرور ایجاد می‌کند.

در سمت سرو یک شی سرولت (servlet) یا یک شنونده (listener) رویداد، درخواست دریافت شده از سمت کاربر مانند داده‌های بازیابی شده از پایگاه داده را کنترل می‌کند. پاسخ با داده‌های درخواستی در قالب سند XML ساخته می‌شود.

با استفاده از تابع برگشتی، شیء XMLHTTPRequest داده‌ها را دریافت می‌کند، آنها را پردازش کرده و HTML وDOMرا به‌روزرسانی می‌کند تا صفحه با داده‌های جدید درخواست شده توسط کاربر نمایش داده شود.

AJAXچندین فناوری را با هم ترکیب می‌کند، زیرا خودش نمی‌تواند به‌طور مستقل صفحات پویا و تعاملی ایجاد کند. موارد زیر لیستی از فناوری‌هایی است که AJAX برای ساخت صفحات وب از آنها استفاده می‌کند.

جاوا اسکریپت (JavaScript): با فعال‌شدن یک رویداد، تابع جاوا اسکریپت فراخوانی می‌شود. تعاملات Ajax با کدهای جاوا اسکریپت آغاز می‌شود و پس از کامل شدن تعاملات آن، جاوا اسکریپت کدهای منبع HTML صفحه را به‌روزرسانی می‌کند.

دام (DOM): مخفف Document Object Model است و برای نمایش ساختار اسناد XML و HTML استفاده می‌شود.

CSS (سی اس اس): برای سبک‌دهی نمایش محتوای صفحه استفاده می‌شود. آموزش کامل و رایگان سی اس اس

XMLHTTPRequest: برای انجام تعامل ناهمگام از کاربر به سرور با استفاده از شیء جاوا اسکریپت به‌کار می‌رود.

مزایایAJAX

با استفاده از AJAX نیازی به ارسال فرم برای اعتبارسنجی نیست. AJAX امکان اعتبارسنجی فرم را به‌طور همزمان هنگام ورود داده‌ها توسط کاربر فراهم می‌کند.

AJAX از بارگیری مجدد کل صفحه جلوگیری می‌کند، زیرا فقط بخشی از صفحه وب را به‌روزرسانی می‌کند.

AJAX براساس استانداردهای باز مانند HTML و CSS برای ارائه صفحه وب است. داده‌ها در XML واکشی شده از سرور ارسال، بازیابی و ذخیره می‌شوند.

واکشی داده‌ها با استفاده از شیء XMLHttpRequest انجام می‌گیرد.

معایب

به این خاطر که به‌روزرسانی توسط کد جاوا اسکریپت (JavaScript) و در سمت سرویس‌گیرنده (client) انجام می‌گیرد، state (وضعیت) در تاریخچه (history) مرورگر ثبت نشده و از این طریق امکان استفاده از دکمه‌های Back و Forward برای پیمایش بین وضعیت‌های مختلف صفحه از کاربر سلب می‌شود.

بنا به دلایل ذکرشده در بالا، کاربر قادر نخواهد بود وضعیت دلخواه را bookmark یا نشانه‌گذاری کند.

داده‌هایی که توسط تکنولوژی AJAX بارگذاری می‌شوند، توسط موتورهای جستجوی مطرح شماره یا اندیس گذاری نمی‌شوند.

کاربرانی که از مرورگرهایی بدون قابلیت پشتیبانی از جاوا اسکریپت استفاده می‌کنند (یا جاوا اسکریپت در آنها غیرفعال شده است)، نمی‌توانند از امکاناتی که AJAX در صفحهٔ وب مورد نظر فراهم می‌کند، بهره‌مند شوند.

دو آیتم اول ذکرشده در لیست معایب بالا را می‌توان با استفاده از یک iframe و خواندن/نوشتن داده‌ها از بخش بعد از کاراکتر # در آدرسURL ، برطرف کرد.

سیاست امنیتی خاستگاه مشترک (Same Origin Policy) در مرورگرها

سیاست خاستگاه مشترک یک ویژگی امنیتی است که در پیاده‌سازی جاوا اسکریپت در بیشتر مرورگرهای مطرح و همچنین دیگر تکنولوژی‌های مورد استفاده در بستر مرورگر، همچون Flash کاربرد دارد. این سیاست در اصل به شما اجازه می‌دهد به صفحات موجود در یک سایت/دامنه درخواست دهید، اما مانع از این می‌شود که به صفحات مستقر بر روی دیگر دامنه‌ها، زیردامنه (subdomain) یا از طریق یک پروتکل دیگر، درخواست ارسال کنید (و داده در صفحهٔ جاری بارگذاری نمایید). از آنجایی که این ویژگی متعلق به جاوا اسکریپت می‌باشد، متعاقباً جزئی از جی کوئری نیز محسوب می‌شود. برای مثال نمی‌توانید از صفحهٔ متعلق به دامنهٔ دیگر، به‌وسیلهٔ توابعAJAX ، اطلاعات در صفحهٔ جاری بارگذاری نمایید. روش‌هایی برای هک‌کردن (یا دور زدن) سیاست مزبور وجود دارد که در تمامی مرورگرها به‌درستی کار نمی‌کند یا مشکلات دیگری را به‌وجود می‌آورد. با این حال، گاهی لازم می‌شود درخواست‌هایی را به صفحهٔ متعلق به دامنهٔ دیگر ارسال کنید، به‌خصوص زمانی که هر دو دامنه به شما تعلق دارد یا مالک دامنهٔ دوم اجازهٔ دسترسی به صفحات آن را به شما می‌دهد. خوشبختانه، استاندارد JSONP به شما این امکان را می‌دهد. اگرچه این کار با جاوا اسکریپت یک نوع هک محسوب شده و انجام آن در این زبان با زحمت فراوانی همراه است، اما jQuery از این قابلیت به‌راحتی پشتیبانی می‌کند و به شما اجازه می‌دهد تا به‌آسانی فراخوانی را براساس JSONP (درست به‌راحتی و مشابه توابعajax) انجام دهید. کافی است نوع داده‌ای بازگشتی (return type)را به "json" تغییر دهید تا فراخوانی ()get و ()post ویرایش شده و براساس استاندارد JSONP صورت گیرد.

JSONP یک تکنیک می‌باشد که برنامه‌نویسان تحت وب برای غلبه‌کردن بر محدودیت‌های cross-domain اعمال شده توسط مرورگرها از آن استفاده می‌کنند. به کمک آن می‌توان اطلاعات مورد نظر را از سیستم‌هایی غیر از سیستمی که صفحه را میزبانی می‌کند (سرویس می‌دهد)، بازیابی نمود.

JavaScript Object Notation(نشانه‌گذاری شیء جاوا اسکریپت)، به اختصارJSON ، یک فرمت برای انتقال آسان و سریع داده‌های ساخت یافته، از طریق اینترنت می‌باشد. اگرچه این تکنیک در اصل برای کار با جاوا اسکریپت طراحی شده، اما هم اکنون بسیاری دیگر از زبان‌های برنامه‌نویسی نیز از آن پشتیبانی می‌کنند. این تکنولوژی به شما اجازه می‌دهد به‌راحتی آرایه‌ها و اشیا خود را به فرمت JSON تبدیل کرده و آنها را با سرعت بالا انتقال دهید.

در مبحث بعدی، یک مثال ایجاد می‌کنیم که در آن یک فایل را، با استفاده از تکنولوژیJSON ، از subdomain دیگر درخواست می‌کنیم.

ارسال درخواست و بازیابی پاسخ

- ساخت نمونه XMLHTTPRequest با:

```
var req = new XMLHTTPRequest();
```

- ارسال درخواست به سرور با استفاده از تابع:open()

فایل می‌تواند هم txt.و هم xlm.باشد

```
req.open("GET","test.txt");
req.open("POST", add-emp.php);
```

- متد GET معمولاً برای ارسال مقدار داده کم به سرور استفاده می‌شود. با متد POST داده‌ها به‌عنوان بخشی از متن درخواست HTTP ارسال می‌شوند. در GET داده‌ها به‌عنوان پارامتر URL ارسال می‌شوند و در آدرس مرورگر قابل مشاهده هستند؛ درحالی‌که در POST داده‌ها را نمی‌توان مشاهده کرد.
- استفاده از تابع send() شیء XMLHTTPRequest() ;برای ارسال درخواست به سرور:

```
req.send();
```

پارامترهای تابع send() اختیاری است که با آنها می‌توان بدنه درخواست را تعیین کرد.

Ajax GET و ارسال درخواست

از GET معمولاً برای بازیابی اطلاعات از سرور استفاده می‌شود. ()send بلافاصله درخواست ناهمزمان را برمی‌گرداند. به همین دلیل قبل از پردازش بیشتر روی پاسخ باید بررسی کنیم که پاسخ در کجای چرخه حیات خودش وجود دارد. از ویژگی readyState شیء XMLHTTPRequest استفاده می‌شود.

ویژگی readyState به‌راحتی با یک عدد صحیح هنگامی فراخوانی تابع onreadystatechange با تغییرreadyState، وضعیت درخواست HTTP را توصیف می‌کند. مقادیر:readyState

- مقدار ۰ برای حالت UNSENT یا فرستاده نشده، درخواست هنوز شروع نشده است.
- مقدار ۱ برای حالت OPENED یا باز شده، تابع open() اتصال سرور را با موفقیت برقرار کرده تا درخواست عملی شود.
- مقدار ۲ برای حالت HEADER_RECEIVED یا هدر دریافت شده، سرور با موفقیت درخواست را دریافت کرده است.
- مقدار ۳ برای حالت LOADING یا در حال پردازش، پردازش درخواست در حال انجام است.
- مقدار ۴ برای حالت DONE یا انجام شده، پردازش درخواست کامل شده و پاسخ در سرور آماده است.

رویداد readstatechange هنگام تغییر ویژگی readyState فعال می‌شود.

کد وضعیت HTTP وضعیت پاسخ XMLHTTPRequest را برمی‌گرداند، معمولاً از کد وضعیت استفاده می‌شود.

–**200**سرور درخواست را با موفقیت پردازش کرد.

–**404**سرور نمی‌تواند صفحه درخواستی را پیدا کند.

–**503**سرور به‌طور موقت در دسترس نیست.

از متد POST برای ارسال داده‌های فرم به سرور استفاده می‌شود. داده‌های فرم را می‌توان با استفاده از شیء FormDataیا با استفاده از یک رشته پرس‌وجو مثل

req.send(key=value1&key=value2&..&keyN=valueN)

ارسال کرد.

در ارسال داده‌ها با یک رشته پرس‌وجو (query) باید به‌طور صریح هدر (header) درخواست را با استفاده از setRequestHeader()تنظیم کنیم.

Req.setRequestHeader(“Content-type”, “application/x-www-form-urlencoded”);

setrequestHeader() بعد از فراخوانی open() و قبل از send() فراخوانی می‌شود.

معمولاً از هدرهای درخواست در setRequestHeader() استفاده می‌شود.

txt/html, text/plain, application/xml, application/json.

با کمک داده‌های فرم می‌توانیم به‌راحتی با استفاده از XMLHTTPRequest.send() مجموعه‌ای از جفت‌های کلید – مقدار برای نمایش فیلدهای فرم و مقادیر آنها بسازیم.

اکشن‌هایAJAX

موارد زیر لیستی از اکشن‌های رخ داده در AJAX می‌باشند:

- کاربر یک رویداد را فعال می‌کند، یک تابع جاوا اسکریپت فراخوانی شده و شیء XMLHTTPRequest ایجاد و پیکربندی می‌شود.
- تماس ناهمزمان توسط XMLHHTPRequest با سرور برقرار می‌شود، سرور پاسخ را در قالب XML برمی‌گرداند.
- پاسخ با استفاده از فراخوانی تابع callback() شیء XMLHTTPRequest پردازش می‌شود و DOM به‌روزرسانی خواهد شد.

امنیت سمت کاربر درAJAX

- از ساخت XML یا JSON به‌صورت پویا خودداری کنید، برای امنیت XML و JSON از یک کتابخانه امن برای حفظ امنیت ویژگی‌ها و عناصر داده استفاده کنید.
- همیشه داده‌هایی که نیاز به رمزگذاری دارند را با استفاده از TLS / SSL در سمت سرور نگه دارید.

- هرگز از eval() سمت کاربر استفاده نکنید، همیشه از txt.به جای html.استفاده کنید، زیرا txt.از بسیاری از مشکلات XSS جلوگیری می‌کند.
- برای جلوگیری از مشکلات نفوذ(injection) ، همیشه قبل از ارسال از رمزگذاری صحیح داده‌ها اطمینان حاصل کنید.
- کدهای ضعیف جاوا اسکریپت به هکرها کمک کرده و مشکلات امنیتی ایجاد می‌کند.
- کاربران می‌توانند کدهای جاوا اسکریپت را بخوانند؛ بنابراین مطمئن شوید که همه داده‌ها و قوانین مهم و حیاتی به جای مرورگر در سرور قرار دارند.
- کدهای جاوا اسکریپت که در زمان بارگیری لازم نیستند را به پایین صفحه منتقل کنید تا صفحه سریعتر بارگیری شود. انتقال کدهای جاوا اسکریپت به انتهای صفحه باعث می‌شود تا مرورگر ابتدا قسمت‌های ضروری و لازم را نمایش دهد و بعد از آن جاوا اسکریپت را بارگیری کند.
- همیشه از جاوا اسکریپت کوچک شده استفاده کنید، زیرا حذف کاراکترهای غیرضروری اندازه جاوا اسکریپت را کاهش می‌دهد.

امنیت سمت سرور درAJAX

- از نوشتن کد سریال‌سازی در سمت سرور خودداری کنید.
- همیشه از توکن‌های CSRF در سمت سرور استفاده کنید.
- همیشه هنگام استفاده از JSON یا XML از فریمورک استفاده کنید.
- احراز هویت، مجوزهای قانونی و سایر محافظت‌های داده را یا در web.xml مشخص کنید یا آن را به‌صورت برنامه‌ای انجام دهید.

نتیجه‌گیری

یاد گرفتیم که با استفاده از AJAX می‌توانیم داده‌ها را به‌آسانی به‌صورت غیرهمزمان از طریق شبکه مبادله کنیم و بدون بارگیری مجدد کل محتوا آن را به‌صورت پویا به‌روزرسانی کنیم. محبوبیت AJAX در حال افزایش است، زیرا یک زبان اسکریپت نویسی با استفاده از JavaScript است که مزایای بسیاری دارد.

- AJAXمجموعه‌ای از فناوری‌های قوی است که برای توسعه صفحات وب پویا استفاده می‌شود.
- با انتقال تنها داده‌های مورد نیاز فرم به سرور به ایجاد صفحات با واکنش‌گرایی بیشتر کمک می‌کند.

یکی از اشیای مهمی که AJAX از آن استفاده می‌کند، شیء XMLHTTPRequest از جاوا اسکریپت است که به ارتباط غیرهمزمان کاربر و سرور کمک می‌کند.

- با استفاده از AJAX می‌توانیم بار شبکه و استفاده از پهنای باند را فقط با درخواست داده‌های مورد نیاز کاهش دهیم.
- بدون AJAX در صفحات وب سنتی دریافت اطلاعات از سرور به زمان بیشتری نیاز دارد. حتی اگر تغییرات کوچکی در صفحه رخ دهد، کل صفحه دوباره بارگیری می‌شود.
- بزرگترین مزیت AJAX استفاده از فرم است که عنصر مشترک در صفحه وب است. با استفاده از AJAX اعتبارسنجی فیلدها در لحظه با یک تماس رفت و برگشتی به سرور برای بازیابی و یا ذخیره داده‌ها بدون ارسال کل صفحه به سرور انجام می‌شود. با عدم ارسال کامل داده‌ها استفاده از شبکه به حداقل رسیده و عملیات سریعتر انجام می‌شود.

Ajax با بهبود فوق‌العاده ویژگی‌های سرعت، عملکرد و قابلیت استفاده، برنامه‌های وب را واکنشگراتر، سریع‌تر و کاربرپسندتر می‌کند.

فصل دوم

آموزش کامل fetch و ajax در جاوااسکریپت

همانطور که می‌دانید در ابتدا آجاکس و یا همان **ajax** در زبان جاوااسکریپت، وظیفه ارتباط با سرور در نوع داده غیر همزمان را برعهده داشت و امروزه در مرورگرهای آپدیت و به‌روز، دستورfetch به ترتیب به معرفی و انواع آنها می‌پردازیم و تا حدی در جزئیات آنها ریز می‌شویم.

استفاده از شیءXMLHttpRequest در java script

با استفاده از شئ XMLHttpRequest در آجاکس می‌توانیم یک درخواست به سمت سرور ایجاد و ارسال کنیم.

مانند مثال زیر :

```
<div id="Result"></div>
<p style="text-align:center;"> <button onclick="Load()">get</button>
</p>
<script>
    function Load() {
        const MyRequest = new XMLHttpRequest();
        MyRequest.onload = function () {
            document.getElementById("Result").innerHTML = this.responseText;
        }
        MyRequest.open("GET", "parsa.html");
        MyRequest.send();
    }
</script>
```

پس کلیت کار بسیار آسان است. درخواستی به سمت سرور ارسال می‌شود، سرور آن را دریافت و تحلیل می‌کند. سپس پاسخی به آن ارسال می‌کند و تمام این موارد در پشت پرده و بدون رفرش صفحه انجام می‌شود.

ساده‌ترین و متداول‌ترین مثال این قسمت، سرچ **گوگل** است. همانطور که می‌بینید، بدون آنکه صفحه رفرش شود، با هر کاراکتری که وارد می‌کنید، نتایج تغییر و آپدیت می‌شود.

به‌عنوان مثال اول می‌توانید یک فرم را با متدهای AJAX ارسال نمایید.

```
<!DOCTYPE html>
<html>
<body>
    <div class="row">
        <input type="text" id="UserName" required placeholder="نام شما">
    </div>
    <div class="row">
        <br>
        <button type="button" onclick="MyLoad();">ارسال</button>
    </div>
    <br>
    <p id="Result"></p>
    <script>
        function MyLoad() {
            const xhttp = new XMLHttpRequest();
            var MyInput = document.getElementById("UserName").value;
            var result = document.getElementById("Result");
            if (MyInput == "") {
                result.innerHTML = "pooor kon!";
            } else {
                xhttp.onload = function () {
                    result.innerHTML = this.responseText;
                }
                xhttp.open("GET", "reg.php?UserName=" + MyInput + "");
                xhttp.send();
            }
        }
    </script>
</body>

</html>
```

Synchronous یا Asynchronous

همزمان در جاوااسکریپت یعنی منتظر پاسخ بمان و تا زمانی که پاسخی دریافت نکردی، هیچ کار دیگری انجام نده، اما نا همزمان به آن معناست که برعکس صحبت قبل عمل می‌کند و به کارهای دیگه هم رسیدگی خواهد کرد.

MyRequest.open(“Method”, “URL”, true or false);

اگر true بزاریم یعنی ناهمزمان (**Asynchronous**) و اگر false بزاریم یعنی همزمان (**Synchronous**) که به‌صورت دیفالت، ناهمزمان است.

onreadystatechange در آجاکس جاوااسکریپت

با استفاده از ویژگی onreadystatechange می‌توانیم یک تابع تعریف کنیم تا اگر وضعیت ویژگی readyState تغییر کرد بتوانیم وضعیت درخواست ارسال شده به سمت سرور را کنترل کنیم.

نام ویژگی	توضیح
readyState	کنترل وضعیت درخواست عدد ۰ یعنی درخواست هنوز آماده‌سازی نشده عدد **۱** یعنی اتصال با سرور برقرارشده عدد **۲** یعنی درخواست دریافت شده عدد **۳** یعنی پردازش روی درخواست انجام شده عدد **۴** یعنی درخواست به‌طور کامل دریافت و پاسخ آماده‌شده
status	۲۰۰ = OK ۲۰۳ = Non-Authoritative Information ۲۰۴ = No Content ۴۰۰ = Bad Request ۴۰۳ = Forbidden ۴۰۴ = Not Found ۵۰۰ = Internal Server Error ۵۰۲ = Bad Gateway ۵۰۳ = Service Unavailable
statusText	برگشت وضعیت *status* به‌صورت رشته) مثلاً OK یا Not Found و(...

برای مثال کد زیر را ببینید:

```
<div id="Result"></div>
<p style="text-align:center;">
    <button onclick="Load()">get</button>
</p>
<script>
    function Load() {
        const MyRequest = new XMLHttpRequest();
        var Result = document.getElementById("Result");
        MyRequest.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                Result.innerHTML = this.responseText;
            }
        };
        MyRequest.open("GET", "parsa.html");
        MyRequest.send();
    }
</script>
```

برای ارسال داده‌های فرم با استفاده از متد POST باید اول با استفاده از **setRequestHeader** مشخص کنیم که نوع داده‌های ارسالی چه چیزی هستند (مثلاً داده‌های فرم)، بعد در متد send باید داده‌ها را وارد کنیم.

```
function MyLoad(){
  const xhttp = new XMLHttpRequest();
  var MyInput = document.getElementById("UserName").value;
  var result = document.getElementById("Result");
  if(MyInput == ""){
  result.innerHTML = "pooor kon!";
  }else{
  xhttp.onload = function(){
  result.innerHTML = this.responseText;
  }
  xhttp.open("POST", "parsa.php");
 xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhttp.send("UserName="+MyInput+"");
  }
}
```

برای قراردادن لودینگ در حین بارگذاری صفحه می‌توانید از کد زیر استفاده کنید:

```
<div id="Result"></div>
<button onclick="Load()">get</button>
<script>
    function Load() {
        const MyRequest = new XMLHttpRequest();
        var Result = document.getElementById("Result");
        MyRequest.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                Result.innerHTML = this.responseText;
            } else {
                Result.innerHTML = "<div><img src='img/loading.gif'
width='70'></div>";
            }
        };
        MyRequest.open("GET", "parsa.html");
        MyRequest.send();
    }
</script>
```

✓ **مثال:** دو سلکت آپشن قرار دهید. یکی برای استان و یکی برای شهر. با کلیک بر روی استان، شهرهای مربوطه در سلکت دوم فعال شود.

معرفی دستور Fetch

جاوااسکریپت می‌تواند درخواست‌های شبکه را به سرور ارسال کند و اطلاعات جدید را هر زمان که نیاز باشد بارگیری کند. Fetch یک رابط مدرن است که به شما امکان می‌دهد؛ درخواست‌های HTTP را از مرورگرهای وب به سرورها ارسال کنید. اگر با شیء XMLHttpRequest (XHR) کار کرده‌اید، Fetch API می‌تواند تمام وظایف را مانند شیء **XHR** انجام دهد.

باید بدانید، Fetchبا promise کار می‌کند، درصورتی‌که XHR با callback.

تابع Promise بعد از اجراشدن، یک پرامیس به ما برمی‌گرداند. این پرامیس شامل یک سری اطلاعات از نتیجه درخواست است. نمونه کد زیر:

```
<script>
    async function users() {
        let response = await
fetch('https://randomuser.me/api/?results=10');
        console.log(response);
    };
    users();
</script>
```

و همچنین کد زیر:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
.then((response) => {
console.log(response.status);
console.log(response.statusText);
console.log(response.ok);
console.log(response.url);
});
```

چیزی که در body قرار می‌گیرد، شامل پاسخ اصلی است که از سمت سرور آمده است در این مرحله باید مشخص کنیم پاسخی که از سرور آمده را به چه فرمتی می‌خواهیم؟

```
response.text();
response.json();
response.formData();
response.blob();
response.arrayBuffer();
```

در مثال زیر خروجی JSON می‌خواهیم:

```
async function users() {
   let response = await fetch('https://randomuser.me/api/?results=10');
   let data = await response.json();
 return data;
 };
users().then(data => console.log(data));
```

مفهوم promise

ما مفهوم **callback** را درک می‌کنیم، اما اگر کد شما در داخل کال بک‌ها، callbackداشته باشد و ادامه یابد، چه اتفاقی می‌افتد؟

خب، این ساختار بازگشتی بازگشت به callback به‌عنوان" **جهنم** "**callback** نامیده می‌شود و به حل این نوع مشکل کمک می‌کند. **Promises** در عملیات **جاوا اسکریپت ناهمزمان** زمانی مفید هستند که ما نیاز به اجرای دو یا چند عملیات پشت سر هم (یا بازگشت بهcallback) داریم، جایی که هر تابع بعدی با تکمیل عملکرد قبلی شروع می‌شود.

یک **Promises** شیئی است که ممکن است در آینده یک مقدار واحد تولید کند، یا یک مقدار حل‌شده یا دلیلی برای حل نشدن (رد) آن.

قبل از اینکه Promise معرفی شود برای انجام چند وظیفهٔ غیر همزمان از callback function استفادهٔ زیادی می‌شد.

به گفته " **developer.mozilla** یک **Promises** یک شی است که نشان‌دهنده تکمیل یا شکست نهایی یک عملیات ناهمزمان است. اساساً، یک **Promises** یک شیء برگشتی است که به جای ارسال callback به یک تابع، callbackرا به آن متصل می‌کند.« Promises های مسئله «جهنمcallback » را حل می‌کند که چیزی جز ساختار بازگشتی از callback نیست.

یک promise ممکن است در سه حالت ممکن باشد...

Fulfilled: زمانی که عملیات با موفقیت به پایان رسید.

Rejected: زمانی که عملیات شکست‌خورده باشد.

در حال تعلیق: حالت اولیه، نه انجام شده و نه رد شده است.

"همگام‌سازی و انتظار، نوشتن Promise ها را آسان‌تر می‌کند..."

```
function myDisplayer(some) {
document.getElementById("demo").innerHTML = some;
}
let myPromise = new Promise(function(myResolve, myReject) {
let x = 0;
// some code (try to change x to 5)
if (x == 0) {
myResolve("OK");
} else {
myReject("Error");
}
});
myPromise.then(
function(value) {myDisplayer(value);},
function(error) {myDisplayer(error);}
);
```

Async & Await **چیست؟**

async یک تابع را **Promise** می‌کند

await باعث می‌شود یک تابع منتظر یک **Promise**باشد.

کلمه کلیدی **async** قبل از یک تابع باعث می‌شود تابع یک **Promise** را برگرداند:

```
async function myFunction() {
  return “Hello”;
}
```

یا

```
function myFunction() {
return Promise.resolve(“Hello”);
}
```

بنابراین در جاوا اسکریپت **عملیات ناهمزمان** را می‌توان در نسخه‌های مختلف...

ES5 -> Callback

ES6 -> Promise

ES7 -> Async/wait

می‌توانید از Async/wait برای انجام درخواست **Rest API** در جایی که می‌خواهید داده‌ها قبل از فشاردادن به نمای به‌طور کامل بارگیری شوند، استفاده کنید.

```
function myDisplayer(some) {
document.getElementById(“demo”).innerHTML = some;
}
async function myFunction() {return “Hello”;}
myFunction().then(
function(value) {myDisplayer(value);},
function(error) {myDisplayer(error);}
);
```

یا در مثال دیگر:

```
const showPosts = async () => {
const response = await
fetch(‘https://jsonplaceholder.typicode.com/posts’);
const posts = await response.json();
console.log(posts);
}
showPosts();
```

متدblob

بدنهٔ پاسخ در قالب یک فایل (Binary Large Objectیاblob) به تابع **onFulfilled** ارسال می‌شود. درنتیجه از متد blob معمولاً زمانی استفاده می‌شود که قصد داشته باشیم یک فایل را به‌صورت کامل دریافت کرده و در صفحهٔ وب درج کنیم (مانند یک فایل JPEG) .

```
const myImage = document.querySelector('img');
const p = fetch('imgcode.....jpg');
p.then(response => response.blob())
.then(function (myBlob) {
let fileURL = URL.createObjectURL(myBlob);
myImage.src = fileURL;
});
```

حالت‌های مختلف شی (Object)

- **Response.ok:** این ویژگی یک Boolean برمی‌گرداند که نشان می‌دهد آیا درخواست موفقیت‌آمیز بوده است یا خیر.
- **Response.status:** این ویژگی کد وضعیت پاسخ را برمی‌گرداند (به‌عنوان مثال عدد ۲۰۰، برای یک درخواست موفق).
- **Response.url:** این ویژگی URL درخواست را برمی‌گرداند. معمولاً هنگام ارسال درخواست، این مورد را دارید.
- **Response.clone():** یک کلون از شیء Response ایجاد می‌کند.
- **Response.text():** داده‌های موجود در پاسخ را به‌صورت متن برمی‌گرداند، که هنگام بازیابی HTML مفید است.
- **Response.json():** داده‌های موجود در پاسخ را به‌عنوان یک شیء JSON معتبر برمی‌گرداند.
- **Response.blob():** برای داده‌هایی که به شکل مرجع فایل هستند؛ (مانند URL تصویر) مفید است.

کد زیر را درنظر بگیرید:

```
<script>
    fetch('https://jsonplaceholder.typicode.com/posts')
        .then(function (response) {
            // if (response.status!== 200)
            if (!response.ok) {
                console.log('Erroradam bash!: ' + response.status);
                return;
            }
            response.json().then(function (data) {
                document.write(data);
                console.log(data);
            });
        })
        .catch(function (err) {
```

```
            document.write('Error: ' + err);
        });
</script>
```

در پارامتر دوم تابع **fetch** می‌توانیم نوع متد را مشخص کنیم، dataپاس بدیم و هدر ست کنیم.

هرگاه اطلاعاتی را از سرور نیاز داشته باشیم باید یک درخواست به آن بزنیم و پاسخ را از سمت سرور دریافت کنیم. بر این درخواست و پاسخ قوانینی حکم فرماست که به آن پروتکل (**Http Protocol**) گفته می‌شود.

هدرهایی که توسط سرور در پاسخ HTTP قرار داده می‌شوند از طریق خاصیتheaders قابل دسترسی هستند. درواقع خاصیت headers یک نمونه از شئ Headers می‌باشد. این شئ دارای تعدادی متد است که با استفاده از آنها امکان دسترسی به هدرهای ذخیره‌شده در این شئ و همچنین اضافه کردن هدرهای جدید وجود دارد.

جالب است بدانید خود Header به‌تنهایی یک شی است که می‌تواند مقادیری را در HTTP ذخیره کند. مانند مثال زیر:

```
const myHeaders = new Headers()
myHeaders.set('name', 'arsam');
document.write(myHeaders.get('name') + '&lt;hr&gt;')
```

نمونه دیگری از ست‌کردن هدر در صفحه:

```
const content = 'Hello World';
const myHeaders = new Headers();
myHeaders.append('Content-Type', 'text/plain');
myHeaders.append('name', 'arsam');
myHeaders.append('Content-Length', content.length.toString());


console.log(myHeaders.has('Content-Type')); // true
console.log(myHeaders.has('Set-Cookie')); // false
console.log(myHeaders.get('Content-Type'));

myHeaders.set('Content-Type', 'text/html');
console.log(myHeaders.get('Content-Type')); // 11
console.log(myHeaders.get('name'));

myHeaders.append('X-Custom-Header', 'AnotherValue');

console.log(myHeaders.get('Content-Length')); // 11
console.log(myHeaders.get('X-Custom-Header')); //
['ProcessThisImmediately', 'AnotherValue']

myHeaders.delete('X-Custom-Header');
console.log(myHeaders.get('X-Custom-Header')); // null
```