# اصول تکنیک آژاکس

*Foundattions of AJAX*

جابر صادقی

# اصول تکنیک آژاکس

*Foundations Of Ajax*

# اصول تکنیک آژاکش

# *Foundations Of Ajax*


گردآوری و ترجمه:

## جابر صادقی



خرداد ۸۶

تقدیم به

**پدر فرزانه و مادر فداکارم**

تقدیم به

**ستاره فروزان زندگی ام**

و

تقدیم به

**هرآنکه بخواهد**

## فهرست اجمالی مطالب

# فهرست محتوا

## با سپاس از

**استاد عزیزم جناب آقای مهندس سهیلی که با بردباری، ضعف و کاهلی اینجانب را تا به ثمر رسیدن این تلاش، تحمل نمودند و با راهنمایی های خویش، مانع از کجروی اینجانب گردیدند.**

**استاد عزیزم مهندس ملکیان که ایشان و کتابهایشان راهنمای همیگشی ام هستند.**

**دوستان عزیز و همراهان گرامی ام آقایان هادی عبدلی و محمود سنچولی که در آماده سازی این اثر یاریگر اینجانب بودند.**

**ستاره فروزانی که بی لطف و محبت او پایان این اثر امکان نداشت.**

**جابر صادقی**

# پیش گفتار

درحالی تصمیم به نگارش پیش گفتاری درخور این اثر گرفتم که هوایی که در آن نفس می کشم سرشار از یأس همراه با کینه نسبت به روزگار است. تنها نیرویی که به دستانم قدرت چرخش قلم و به ذهنم قدرت فکر داد تا آخرین صفحات این اثر را آماده سازم، نیرویی امیدی است که از نور محبت ستاره ای مهربان به جان و فکرم می تابد. خود واقفم که این جملات مناسب پیش گفتار اثری با ادعای علمی نیست، اما با کمال میل این خیط و سرکشی از قوانین را به جان می خرم، چرا که معتقدم عامل تبدیل علم به عمل چیزی جز عشق نیست. به عشقی که در جان و دل دارم افتخار می کنم و این اثر ناچیز را به ستاره فروزان زندگی ام تقدیم می دارم.

آخرین سال تحصیل اینجانب، با ظهور تکنیک آژاکس همراه بود. با توجه به اینکه در زمینه معرفی این تکنیک تلاشی درخور، تا آن زمان صورت نگرفته بود، این موضوع را به عنوان پروژه دوره کارشناسی برگزیدم و تصمیم به فهم و ارائه آن در قالب این اثر گرفتم.

در آغاز راه، به جمع آوری منابع موجود پرداختم، که همه این منابع را در DVD همراه این اثر ارائه نموده ام. این منابع شامل کتابهای الکترونیکی، فایلهای آموزشی تصویری و صوتی وصفحات آموزشی وب سایتهای مختلف می باشد. در ادامه کار به بررسی این منابع پرداخته و کتابهای مختلف را بررسی و مطالعه نمودم.

دسته ای از این منابع به بررسی اصول اساسی این تکنیک پرداخته و سعی به توضیح دنیای ماورای این تکنیک داشتند و دسته ای دیگر نیز تنها نگاهی اجمالی به اصول این تکنیک کرده و به آموزش کار با کتابخانه ها و ابزارهای برنامه نویسی موجود، جهت استفاده و تولید محصولات با قابلیت آژاکس، پرداخته بودند.

با توجه به اینکه ابزارها و کتابخانه های موجود، وابسته به زبانهای برنامه نویسی هستند و هرکس بسته به اینکه با کدام زبان برنامه نویسی آشناست، یکی از این ابزارها را انتخاب می کند، تصمیم به جمع آوری مطالب در مورد اصول تکنیک آژاکس گرفتم که بدون وابستگی به هیچ یک از ابزارها و کتابخانه های موجود، راهگشای عزیزانی باشد که سعی در فهم این تکنیک دارند تا با چشمی بازتر به انتخاب ابزارهای موجود این تکنیک بپردازند.

از بین منابع موجود، کتابی با نام Foundations Of Ajax را به عنوان مرجع اصلی برگزیدم و دلیل این انتخاب نیز نگاه دقیق و موشکافانه نویسندگان این کتاب بود.

با توجه به اینکه نویسندگان این کتاب بیشتر روی جنبه های این تکنیک متمرکز شده بودنـد و به مقدمات دنیای وب نپرداخته بودند، فصل اول که آشنایی با دنیای وب و مقدمات مورد نیـاز می باشد را از کتاب مهندسی اینترنت مهندس ملکبان که افتخار شاگردی ایـشان را داشـته ام، برگزیدم. نگاه دقیق استاد ملکیان باعث شد تا تمام این فصل را از کتاب ایشان برگزینم.
فصلهای دوم، سوم و چهارم از مرجع اصلی آورده شده است و هرجا که نیاز به توضیح بیـشتر بوده یا اینکه به مطلب مهم یا جالبی اشاره نشده بود، از دیگر منابع استفاده کردم که متاسـفانه به دلیل اینکه جمع آوری و آماده سازی مطالب این فصلها به خاطر کاهلی اینجانـب بـه درازا کشیده شد، نتوانستم این ارجاعات را مشخص نمایم.
فصل پنجم را نیز به معرفی و آشنایی با ابزار Atlas که از ابزارهـای تولیـد قابلیـت Ajax در ASP.NET می باشد، اختصاص داده ام و درهمین فصل به توضیح در مورد پروژه مـدیریت کاربران وب سایت پرداخته ام که مثالی مناسب جهت معرفی قابلیتهای ابزار Atlas می باشد.

امید آن دارم که این اثر ناچیز، بتواند راه گشای دوستان و علاقه مندان آشـنایی بـا دنیـای وب باشد و خواهش آن دارم که با نظرات و انتقادات خـود، مـانع تکـرار اشـتباهات اینجانـب در تلاشهای آینده گردند.

با آرزوی موفقیت و شادکامی
جابر صادقی
sadeghi.jaber@yahoo.com
خرداد ۸۶ - کرج

## آژاکس چیست؟

آژاکس مخفف عبارت Asynchronous Javascript And XML است به معنی استفاده از جاوااسکریپت و XML در تراکنشهای همگام. اگر بخواهیم در معنی لغات دقت کنیم باید بگوییم که آژاکس تکنولوژی جدیدی نیست، بلکه فقط تکنیک های شناخته شده برنامه نویسی وب را به گونه ای غیر معمول ترکیب می نماید تا به برنامه نویسان وب این امکان را بدهد تا برنامه های وب را به صورت جذابتر از آن چیزی که به صورت معمول رواج دارد، تولید کنند.

وقتی ما با برنامه های کاربردی که تحت وب نیستند (همان application ها) کار می کنیم انتظار داریم نتایج عملیاتی را که انجام می دهیم فوراً تولید و نمایش داده شوند، بدون نیاز به اینکه ما منتظر باشیم تا کل صفحه نمایش برنامه دوباره توسط برنامه تولید شود. مثلاً هنگامی که ما با نرم افزاری مثل اکسل کار می کنیم انتظار داریم اثر تغییراتی که در یک ستون ایجاد می کنیم وقتی که مشغول ورود اطلاعات دیگر یا کار با ماوس هستیم، بی درنگ به ستونها و سلولهای مرتبط منتقل شود.

متاسفانه اینگونه تعامل برای کاربران برنامه های تحت وب بسیار کم در دسترس بوده است. بسیاری این تجربه را داشته اند که فیلدهای اطلاعاتی یک فرم وب را پر کرده اند و روی یک دکمه یا لینک کلیک نموده و منتظر مانده اند تا صفحه وب به آرامی تغییر کند و نتایج به دست آمده را نمایش دهد. به علاوه، بسیار شاهد بوده ایم که صفحات جدید شامل مولفه های یکسانی با صفحه قبل می باشد و نیازی به بارگذاری مجدد نداشته اند. عکس های زمینه، لوگوها و منوها از این دسته مولفه ها هستند.

آژاکس نوید حل این مشکل را به ما می دهد. آژاکس به صورت یک لایه اضافی بین مرورگر کاربر و سرور وب عمل نموده و ارتباطات سرور را در پس زمینه پیاده سازی می کند، درخواست ها را به سرور ارسال و نتایج آن را پردازش می نماید. نتایج به دست آمده ممکن است با محتویات صفحه در حال نمایش ترکیب شده و به نمایش درآیند بدون اینکه نیاز به بارگذاری مجدد تمام صفحه وب باشد.

در برنامه های آژاکس حتماً نیاز نیست که درخواست ها حساس به عمل کاربر مثل کلیک دکمه یا لینک باشند. در یک برنامه آژاکس با نگارش خوب، ممکن است قبل از اینکه کاربر انجام عملی را بخواهد، درخواست مورد نظر به سرور ارسال و نتایج آن دریافت و به کاربر

نشان داده شود. این معنی کلمه غیر همگام( asynchronous ) در عبارتی که آژاکس مخفف آن است، می باشد.
قسمتی از برنامه آژاکس که در پس زمینه مرورگر کاربر انجام می شود مثل ارسال درخواست به سرور و پردازش نتایج آن، توسط جاوااسکریپت انجام می شود و XML به عنوان ابزار رایج برای کدینگ و فرمت ارسال اطلاعات توسط آژاکس استفاده می شود، تا اطلاعات بین کلاینت و سرور به صورت کارآتر منتقل شوند.
در ادامه این کتاب تمام این تکنیک ها را مورد بررسی قرار خواهیم داد.

عناوین زیادی وجود دارد که باید در این کتاب به آن ها بپردازیم. در ابتدا به این می پردازیم که WWW چیست و چگونه به وجود آمده است.

## تاریخچه مختصری از وب[1]

در اوایل دهه نود، تور جهان گستر یا وب در حالی متولد شد که شبکه اینترنت حدود بیست سال سن داشت. به جرأت می توان گفت تا قبل از تولد وب، فقط دانشگاهیان و پژوهشگران در خصوص اینترنت چیزی می دانستند یا از خدمات بهره می بردند. تا قبل از ظهور وب عمومی ترین کاربرد اینترنت، سیستم پست الکترونیکی بود. وب، اینترنت را به شبکه همگانی تبدیل کرد. سیطره نفوذ آن را به اعماق لایه های اجتماع متمدن امروز کشاند و در زندگی روزمره مردم جای خود را باز کرد. امروزه کاربران وب خود یک جامعه بزرگ و جهانی تشکیل داده اند، جامعه ای که همانند جوامع بشری دیگر، نیک و بد را در کنار هم دارد! جامعه وب بدون مرز و جهانی است.
در خلال پانزده سال اخیر، وب تحولات و پیشرفتهای شگرفی داشته است. زبان های وب بسیار قدرتمند و قابل انعطاف شده اند. سرویس دهنده های وب خدمات بسیار قدرتمند و وسیع تری را عرضه می کنند و مرورگرها روز به روز جذاب تر، مجهز به امکانات بیشتر و کارآمدتر شده اند. مجموعه این عوامل باعث شده که وب از شکل یک رسانه ارتباط جمعی خارج شده و در کلیه شئونات زندگی بشر وارد شود.امروزه صحبت از تجارت الکترونیکی،

[1] - برگرفته از کتاب مهندسی اینترنت مهندس ملکیان

دولت الکترونیکی، آموزش الکترونیکی و بسیاری از چیزهای الکترونیکی دیگر بر سر هر زبانی شنیده می شود.

## مفهوم وب

تور جهان گستر یا «وب»، یک روش معماری یا به عبارتی یک نظام برای ذخیره سازی، سازماندهی و دسترسی به مستندات به هم پیوند خورده ای است که بر روی هزاران ماشین در کل جهان پراکنده و توزیع شده اند. هر یک از این مستندات پیوند خورده حاوی متن، صدا، تصویر، کد اجرایی، اسکریپت و نظائر آن هستند و می توانند به سند یا اسناد دیگر واقع بر ماشینی متفاوت از این جهان بزرگ اشاره نمایند.

بزرگترین مزیت وب سادگی استفاده از آن است؛ کاربر با وارد کردن آدرس یک وب سایت، صفحه اصلی (Home Page) آن را بر روی کامپیوتر خود متنقل می کند؛ آن را نگاه می کند و مطالعه نموده و اگر تمایل به دریافت اطلاعات بیشتری در خصوص آیتمهایی که پررنگ هستند، داشت با موش خود روی آن کلیک می نماید.(آیتمهای پررنگ نقطه پیوند صفحه فعلی با صفحات دیگر به حساب می آیند)؛ با کلیک بر روی یکی از این پیوندها، صفحه جدید هم مثل صفحه قبلی بر روی کامپیوتر او بار گذاری شده و این روند ادامه می یابد. کاربر عموماً متوجه نیست که هر صفحه از چه نقطه ای از جهان بر روی کامپیوترش منتقل شده است.

بدیهی است که در ذیل هر آیتم پررنگ(که از این به بعد با نام ابرپیوند یا Hyperlink از آن یاد می کنیم) یک آدرس دیگر وجود دارد که ارتباط بین دو صفحه را برقرار کرده است. این آدرس خاص که URL نام دارد موقعیت دقیق صفحات ذخیره شده در هر نقطه از جهان را مشخص می کند. URL را یک آدرس استاندارد و جهانی فرض کنید که موقعیت هر منبع (Resource) همانند صفحات وب، فایلهای داده، صدا، تصویر، اسکریپتهای اجرایی و نظائر آنها را به دقت مشخص می نماید

شاید وب جذابترین جنبه استفاده از مدل انتزاعی «سرویس دهنده/مشتری» (Client/Server) در شبکه اینترنت باشد. برنامه سمت مشتری (یعنی مرورگر) تقاضایی را برای دریافت یک صفحه وب یا یک فایل به سمت سرویس دهنده دلخواه در هر نقطه از جهان ارسال می دارد. برنامه سرویس دهنده با دریافت تقاضا و در صورت داشتن مجوز، آن را پذیرفته و داده ها و پاسخ مناسب را بر می گرداند.

## مفهوم URL (Uniform Resource Locator)

اشاره کردیم که یک صفحه وب می تواند شامل یک پیوند به صفحه وب دیگر در هر نقطه از دنیا باشد. هر پیوند در حقیقت آدرس یک صفحه وب محسوب می شود.

با توجه به ناهمگون بودن سیستم های عامل و کامپیوترها در دنیا، به عنوان یک نیاز بنیادی باید بتوان فایل ها و پروسه ها (و در یک تعریف عام هر منبع) را از لحاظ سبک نام گذاری و محل استقرار آنها بر روی یک ماشین، هماهنگ و استاندارد کرد. یعنی باید باید یک روش آدرس دهی برای هر منبع انتخاب شود به گونه ای که بتواند به سه سوال زیر در دنیا پاسخ دهد:

- نام فایل (یا منبع) چیست؟
- محل دقیق ذخیره شده فایل (یا منبع) کجاست؟ (یعنی روی چه ماشینی و چه زیر شاخه ای قرار دارد؟)
- به چه روشی باید به فایل (یا منبع) دسترسی داشت و طبق چه قاعده ای می توان آن را انتقال داد؟

تعریف منبع (Resource) : طبق تعریف آقای «تیم برنرزلی» (ابداع کننده وب) منبع هر چیزی است که دارای هویت (Identity) باشد. آشناترین نوع منابع را می توان فایلهای حاوی داده معمولی مثل اسناد HTML، XML، PDF و نظائر آن یا فایلهای صدا و تصویر و برنامه های کاربردی قابل اجرا (که به صورت پویا تولید سند می کنند) یا هر شئ مشابه دانست. مشخصه یک منبع آن است که از طریق شبکه قابل بازیابی باشد بدین معنا که اگر از نوع فایل معمولی است بتوان آن را منتقل کرد و اگر برنامه کاربردی است بتوان از راه دور آن را اجرا و نتیجه را دریافت نمود.

یک روش آدرس دهی استاندارد باید هر فایلی را به طور منحصر به فرد و یکتا در دنیا مشخص نماید به نحوی که هیچ ابهامی در شناسایی آن وجود نداشته باشد. URL روشی برای آدرس دهی منابع و فایلها در دنیاست که به هر سه سوال فوق بدون هیچ ابهامی پاسخ می دهد. بنابراین URL را در ذهن خود به عنوان یک آدرس استاندارد یا یک سبک آدرس دهی مجسم کنید.

آدرس **URL** که امروزه شما حتی روی پوست یک بیسکوئیت آن را می بینید شامل سه قسمت اساسی است :

۱. شناسه پروتکل که به آن پروتکل انتقال (Transfer Protocol) هم گفته می شود.
۲. نام ماشینی که فایل روی آن قرار دارد.(نام حوزه ماشین یا آدرس IP آن)
۳. شاخه (دایرکتوری) و نام فایل

به عنوان مثال به یک آدرس به صورت http://www.ibm.com/researches/piv/paper.htm دقت نمایید. این URL از سه قسمت تشکیل شده است :

- پروتکل این فیلد (که در این مثال http است) به برنامه سمت مشتری یعنی مرورگر تفهیم می کند که برای بارگذاری و انتقال فایل از پروتکل http و مجموعه فرامین آن استفاده نماید. در حقیقت این فیلد زبان صحبت کردن مرورگر با سرویس دهنده را تعیین می کند. فیلد پروتکل با علامت :// از بقیه آدرس جدا می شود.
- نام حوزه ماشین نام حوزه ماشینی که منبع مربوطه روی آن ذخیره شده است. در مثال فوق نام حوزه ماشین www.ibm.com است. نام حوزه ماشین بین :// تا اولین / بعدی قرار می گیرد.
- نام شاخه و نام فایل در این قسمت که دقیقاً بعد از نام حوزه شروع می شود و تا انتها ادامه می یابد، نام شاخه ای که فایل درون آن قرار گرفته و همچنین نام فایل درج می شود. در مثال فوق نام فایل paper.htm است که بر روی شاخه /resource/piv/ قرار دارد.

گاهی یک URL به شکل http://www.ibm.com/cgi-bin/resp.php?sid=987jajs&name=%20ali مشاهده می کنید. این URL به یک اسکریپت اجرایی اشاره دارد که باید با آرگومانهایی که بعد از علامت ؟ آمده اند اجرا شود و نتیجه اجرا به متقاضی برگردد. در یک URL به کارکترهایی مثل ٪، : ، / ، & ، + یا ؟ اصطلاحاً «متاکاراکتر» گفته می شود.

برخی از پروتکلهای انتقال که در آدرس URL قابل تعریف هستند را به صورت مختصر در جدول بعد آورده شده اند.

جدول ۱-۱ : پروتکلهای انتقال

| نام پروتکل | مورد استفاده | مثال |
|---|---|---|
| http | انتقال صفحات ابرمتن | http://www.ibm.com |
| ftp | انتقال فایل | ftp://ftp.cs.vu.nl.minx/readm |
| file | فایلهای محلی | file://user/prog.c |
| news | گروههای خبری | news://comp.os.minix |
| gopher | گوفر | gopher://gopher.tc.mn.edu |
| telnet | تل نت | telnet://www.w3.org:80 |

## معماری سیستم وب

کلاً از دیدگاه فنی، سیستم وب در دو بخش سازماندهی می شود:

- برنامه سمت سرویس دهنده وب و برنامه سمت مشتری وب
- مجموعه اطلاعات توزیع شده از صفحات ابرمتن، فایلهای داده مثل صدا، تصویر و به طور کل هر منبع

صفحه وب چیزی نیست مگر یک فایل متنی بسیار ساده که با یکی از زبانهای نشانه گذاری «ابرمتنی» مثل HTML، XHTML، DHTML یا XML تدوین می شود. کاری که مرورگر به عنوان مشتری وب انجام می دهد آن است که تقاضای دریافت یکی از این صفحات یا فایل ها را در قالب قراردادی استاندارد ( به نام پروتکل HTTP ) به سمت سرویس دهنده ارسال می کند.در سمت مقابل سرویس دهنده وب این تقاضا را پردازش کرده و در صورت امکان، فایل مورد نظر را برای مرورگر ارسال می کند. مرورگر پس از دریافت فایل ابرمتنی، آن را تفسیر کرده و به صورت صفحه آرایی شده، روی خروجی نشان می دهد. اگر فایل ابرمتنی در جایی به فایل صدا یا تصویر پیوند خورده باشد آنها نیز توسط مرورگر تقاضا شده و پس از دریافت در جای خود قرار می گیرند.

سرویس دهنده وب را نیز باید یک برنامه سوکت در نظر گرفت که فرامین مشتری را دریافت، پردازش و در صورت امکان اجرا می کند. برنامه سمت مشتری نیز برنامه سوکتی است که

تقاضاها را در قالب فرامین استاندارد، برای سرویس دهنده وب ارسال می کند؛ در ضمن وظیفه تفسیر و نمایش داده های دریافتی را نیز برعهده دارد.

## زبانهای نشانه گذاری وب

اینترنت اجتماعی است از انواع و اقسام ماشینهای ناسازگار با سیستم های عامل متفاوت و نرم افزارهای کاربردی جورواجور!! وقتی قرار است این اجتماع رنگارنگ و ناسازگار بر سر سفره جهانی وب بنشینند بایستی زبانی مشترک برای تدوین و تبادل اطلاعات و اسناد داشته باشند، به گونه ای که این تفاوتها و ناسازگاریها احساس نشود. بنیادی ترین نیاز وب، یک زبان واحد و استاندارد برای تدوین و صفحه آرایی اسناد وب بود، به گونه ای که هیچ وابستگی به سخت افزار یا نرم افزار خاصی نداشته باشد. به عبارت دیگر صفحات وب بایستی به گونه ای تدوین و صفحه آرایی شوند تا بر روی تمام ماشینها اعم از VAX ، IBM PC ، SUN و Apple به یک شکل نمایش یابند و طریقه تفسیر و نمایش آن به هیچ عامل خارجی وابستگی نداشته باشد.

اولین زبان نشانه گذاری رسمی وب، HTML (Hyper Text Markup Language) بود که در سال ۱۹۸۹ توسط تیم برنرزلی ابداع شد. تقریباً تمام وب سایتهای دنیا به نحوی از زبان HTML برای نمایش همزمان متن، تصویر، انیمیشن، فیلم و صدا بهره می گیرند. این زبان تحت نظارت کنسرسیوم جهانی وب (با عنوان W3C یا World Wide Web Consortium) قرار دارد و در خلال دهه گذشته چندین بار بازبینی و به روز شده است و اکنون از نسخه ۴ آن استفاده می شود. برای کسب آگاهی از توصیف دقیق این زبان می توانید به آدرس http://www.w3c.org مراجعه نمایید.

صفحات HTML متون غنی شده ای هستند که مؤلفه ها و اشیای موجود در یک سند را به صورت صفحه آرایی شده و سازمان یافته، توصیف کرده و در اختیار کاربر قرار می دهند. بزرگترین حُسن این صفحات آن است که به کاربراین امکان را می دهند که به سادگی به صفحه دیگری دسترسی پیدا کند، به گونه ای که می توان توده ای انبوه از اطلاعات خام را به صورت سلسله مراتبی و سطح بندی شده در اختیار علاقه مندان قرار داد.

زبان HTML ، زبانی مانند پاسکال، C یا بیسک نیست بلکه روشی است که به واسطه آن می توان متون خالص و معمولی را صفحه آرایی کرده و عواملی مثل صدا، تصویر، فهرستهای انتخاب و عناصر ورود اطلاعات را به یک سنداضافه کرد.

HTML محموعه ای از برچسبهای خاص صفحه آرایی، عوامل و اشیا ورود و خروج اطلاعات است. برچسبهای درون متن(مشهور به tag) توسط مرورگر تشخیص داده شده و پس از تفسیر، صورت ظاهری سند و نمایش متن را تحت تاثیر قرار می دهند. برچسبهای HTML با علامت < > از متن اصلی متمایز می شوند. عملی که هر برچسب انجام می دهد در درون < > مشخص می شود. به عنوان مثال متن ساده Internet Engineering را در نظر بگیرید. با برچسبهای <I><B>Internet Engineering</B></I> می توان مرورگر را وادار کرد تا متن را به صورت پررنگ و ایتالیک به شکل ***Internet Engineering*** نمایش بدهد. فرض برآن است که تمام مرورگرهای جهان برچسبهای استاندارد HTML 4.0 را به رسمیت می شناسند و آنها را به درستی تفسیر می کنند.

کسی که با برچسبهای HTML آشنا باشد به راحتی می تواند با یک ویرایشگر ساده صفحات ساده وب مورد نظر خود را ایجاد و سازماندهی کند ولیکن برای سرعت بخشیدن به روال طراحی صفحات و انعطاف بیشتر، نرم افزارهای گوناگونی برای تولید صفحات وب به بازار عرضه شده است.

در ادامه با برخی از برچسبهای HTML آشنا می شویم. هدف از این بخش فقط آشنایی با کلیات HTML است و برای یادگیری اصولی آن باید به مراجع اصلی و کتب تفصیلی مراجعه کرد.

ساختار کلی یک صفحه وب به صورت زیر است :

```
<HTML>
    <HEAD>
        <TITLE>
        عنوان سند در اینجا درج می شود.
        </TITLE>
    </HEAD>
    <BODY>
        تمام اطلاعات صفحه وب در اینجا درج می شود.
    </BODY>
</HTML>
```

یکی از اصلی ترین برچسبها در سازماندهی متون، برچسب پاراگراف است. مرورگر، پاراگراف را براساس عرض پنجره نمایش تنظیم می نماید و درصورت لـزوم جمـلات را شکـسته و بـه خطوط بعد می برد؛ به این کار «پوشش خودکار» گفته می شود. یـک پـاراگراف بـا برچـسب <P> آغاز و با </P> خاتمه می یابد.

برای ارائه یک مطلب مهم که نظر مخاطب را به خود جلب کند از برچـسب <EM> اسـتفاده می شود؛ در این حالت متنی که تحت تاثیر این برچسب قرار می گیرد به صـورت کـج نـشان داده می شود. برای تاکید بیشتر می توان از برچسب <STRONG> استفاده کرد تا متن را بـه صورت پررنگ نشان بدهد.

برای آنکه یک سطر را قطع کرده و سطر بعدی از سـر خـط جدیـد آغـاز شـود از برچـسب <BR> استفاده می شود. این برچسب می تواند عملیات نمایش مـتن را در سـطر جدیـد بـه صورت زیر کنترل کند :

<BR CLEAR=LEFT> خاتمه سطر فعلی و شروع سطر بعدی از سمت چپ به راست

<BR CLEAR=RIGHT> خاتمه سطر فعلی و شروع سطر بعدی از سمت راست به چپ

<BR CLEAR=ALL> خاتمه سطر فعلی و شروع سطر بعدی در کل عرض پنجره نمایش

ابرپیوند (Hyperlink / Hyper Reference) قبلاً اشاره شد که در یک صفحه وب هر آیـتم از متن می تواند به صفحه دیگر اشاره کند. برای آنکه قطعه ای از متن به صورت ابرپیوند عمل کـرده و کـاربر بـا کلیـک کـردن روی آن، صـفحه دیگـری را دریافـت کنـد، از برچـسب <A HREF=.....> اسـتفاده مـی شـود. بـه مثـال زیـر دقـت کنیـد :

<A HREF=" introdoc.html"> Introduction </A>

این عبارت باعث می شود که کلمه Introduction در متن به صورت پررنگ نشان داده شـده و کـاربر بـا کلیـک آن، مرورگـر را وادار کنـد تـا فایـل introdoc.html را بارگـذاری (Download) و نمایش بدهد. در زیر، یک ابرپیوند با URL دقیق و کامل می بینید :

<A HREF="http://www.w3c.org/hypertext/datasource/www/feo.html">
Geographical.html </A>

برای نمایش تصاویر گرافیکی از برچسب <IMG...> استفاده می شود. تصاویر فشرده شـده نوع gif و jpeg یا png قابل بارگذاری در یک صفحه وب هستند :

<IMG ALIGN=TOP SRC="mypic.gif">

با برچسب بالا، مرورگر تصویر mypic.gif را در بالای خط فعلی متن قرار می دهد. گزینه های دیگر عبارتند از :

```
<IMG ALIGN=MIDDLE  SRC="mypic.gif">
<IMG ALIGN=BOTTOM  SRC="mypic.gif">
```

**نمایش فهرستها**

در HTML دو نوع فهرست قابل تعریف است:

الف) فهرست بی شماره: شروع این نوع فهرست با برچسب <UL> و پایان آن با </UL> مشخص و عناوین فهرست، با برچسبهای <LI> و </LI> تفکیک می شوند:

```
<UL>
      <LI> عنوان ۱ </LI>
      <LI> عنوان ۲ </LI>
      <LI> عنوان ۳ </LI>
</UL>
```

ب) فهرست شماره دار: این نوع فهرست مشابه فهرست قبلی است با این تفاوت که عناوین فهرست به ترتیب شماره گذاری می شود. ابتدای فهرست شماره دار با <OL> مشخص می شود. سپس با برچسب <LI> عناوین از یکدیگر جدا می شوند؛ پایان فهرست نیز با برچسب </OL> مشخص می گردد.

```
<OL>
      <LI> عنوان ۱ </LI>
      <LI> عنوان ۲ </LI>
      <LI> عنوان ۳ </LI>
</OL>
```

نوع و اندازه قلم : برای تعیین نوع و اندازه قلم (فونت) از برچسب <font size="xyz" face="Font Name"> استفاده می شودکه xyz اندازه قلم و Font Name نام قلم مورد نظر را تعیین می کند. هرگاه مرورگر قلم مورد نظر را نشناسد متن را با قلم پیش فرض نشان خواهد داد.

رسم جدول : جداول یکی از نیازهای اصلی صفحات وب تلقی می شوند در حالی که نسخه های قدیمی HTML، این امکان را فراهم نکرده بودند. از نسخه ۳ به بعد، رسم جدول با برچسب <TABLE ...> امکان پذیر شد. این برچسب به صورت زیر است :

<Table width="?" cols="?" border="?" frame="?" cellspacng="?" cellpadding="?">

هر یک از گرینه های این برچسب معنای خاصی را می دهند که در ادامه آورده شده است.

- Width عرض جدول
- Cols تعداد ستونها
- Border ضخامت حاشیه اطراف جدول برحسب پیکسل
- Frame حاشیه قابل رؤیت اطراف جدول
- Cellspacing فضای خالی میان خانه های جدول
- Cellpadding فضای خالی بین داده های درون جدول

پس از تعریف ساختار جدول، نوبت به تعریف سطر و ستونهای آن با برچسبهای زیر می رسد:

- برچسبهای <TR> و </TR> یک سطر از جدول را تعیین می کند.
- برچسبهای <TD> و </TD> محتوای هریک از خانه های جدول توسط این برچسب مشخص می شود.

در مثال زیر یک جدول ۳×۳ تعریف و به خانه های آن مقدار داده شده است :

```
<font size="6"  face="Nazanin">
<table border="3" cellpadding="3" cellspacing="3">
<tr>
        <td><p align="center"> ستون۱ سطر۱ </td>
        <td><p align="center"> ستون۲ سطر۱ </td>
        <td><p align="center"> ستون۳ سطر۱ </td>
</tr>
<tr>
        <td><p align="center"> ستون۱ سطر۲ </td>
        <td><p align="center"> ستون۲ سطر۲ </td>
        <td><p align="center"> ستون۳ سطر۲ </td>
```

```
</tr>
<tr>
        <td><p align="center"> ستون۱ سطر۳ </td>
        <td><p align="center"> ستون۲ سطر۳ </td>
        <td><p align="center"> ستون۳ سطر۳ </td>
</tr>
</table>
```

**فرمهای ورود اطلاعات در HTML : آن دسته از صفحات وب که فقط ارائه کننده اطلاعات به کاربر هستند و هیچ کنش و واکنشی با کاربر ندارد، اصطلاحاً «صفحات ایستا» ( Static Page) نامیده می شوند. کاربر با دریافت چنین صفحاتی، در زمینه موضوع دلخواه خود اطلاعاتی را از سرویس دهنده دریافت و مطالعه می کند. تمام اجزای صفحه به منظور مطالعه کاربر آراسته شده است.**

**یکی از بزرگترین مشخصه های وب آن است که کاربر می تواند با یک صفحه وب در تراکنش باشد؛ بدین معنا که قادر است در صفحه وب اطلاعاتی را وارد نموده و آن را به سمت سرویس دهنده بفرستد. سرویس دهنده نیز پس از دریافت اطلاعات و پردازش آن،مجدداً در پاسخ، اطلاعاتی را بر می گرداند. برای چنین کاربردهایی، صفحه وب باید شامل اجزایی جهت ورود اطلاعات باشد. به عنوان مثال فرض کنید یک شرکت مایل است در سایت وب خود، ضمن معرفی محصولات شرکت، از مشتریان سفارش بگیرد. بنابراین طراح صفحه وب باید نواحی ورود اطلاعات مشتری را در صفحه وب تعریف کند.پس از آنکه کاربر اطلاعات خود را در این نواحی وارد کرد، مرورگر را وادار می کند تا آنها را برای سرویس دهنده بفرستد.**

- **برای تعیین ناحیه ورود اطلاعات در یک صفحه وب، از برچسب <FORM ...> استفاده می شود. ناحیه ورود اطلاعات با برچسب </FORM> خاتمه می یابد. هر صفحه وب می تواند چند ناحیه ورود اطلاعات داشته باشد؛ اطلاعات هر ناحیه ورود اطلاعات، به طور یکجا برای سرویس دهنده ارسال خواهد شد. در هر ناحیه حداقل یک گزینه «ارسال» (Submit) وجود دارد که کاربر پس از آنکه اطلاعات این ناحیه را تکمیل کرد توسط آن به مرورگر فرمان می دهد تا آنها را برای سرویس دهنده ارسال کند. اطلاعات ارسالی در قالب مشخص و استاندارد تحویل یک پروسه خاص**

بر روی ماشین سرویس دهنده خواهد شد تا آنها را پردازش کند و پاسخی مناسب برگرداند. نام و آدرس این پروسه، درون برچسب <FORM...> مشخص شده است. این پروسه که اصطلاحاً CGI یا اسکریپت سمت سرویس دهنده نامیده می شود، توسط سرویس دهنده فراخوانی شده و اطلاعات ارسالی را دریافت می دارد.

برچسب <FORM...> دارای ویژگیها و پارامترهای زیر است :

- Action آدرس URL مربوط به محل قرار گرفتن برنامه اسکریپت سمت سرور که اطلاعات ارسالی را دریافت و پردازش خواهد کرد.
- Method یکی از متدهای HTTP که توسط آن اطلاعات به سمت سرویس دهنده ارسال می شود. این خاصیت می تواند GET یا POST باشد.
- Encrypt نوع کدگذاری داده های ارسالی را مشخص می کند. اگر این خصوصیت معرفی نشود، اطلاعات ارسالی متن معمولی فرض خواهد شد.

- برچسب <INPUT ...> با استفاده از این برچسب می توان یکی از عوامل دریافت اطلاعات در صفحه وب را تعریف کرد. ویژگی های این برچسب عبارتند از :
  - Type : در صفحه وب می توان انواع اشیا و عوامل ورود اطلاعات را تعریف کرد. با این ویژگی می توان نوع عامل ورودی را تعیین نمود. این مقدار می تواند یکی از موارد Text، Password، CheckBox، Radio، Submit، Reset باشد.

  انتخاب نام برای تمام ورودی ها به جز Submit و Reset الزامی است.زیرا وقتی مرورگر اطلاعات وارد شده توسط کاربر را برای سرویس دهنده ارسال می نماید، نام فیلد مربوطه را به همراه مقدار آن، ارسال خواهد کرد.برنامه سمت سرور، وظیفه دارد نام فیلدها و مقادیر آنها را از هم تفکیک کند.

  در تمام اشیا و عوامل ورودی، گزینه Value مقدار پیش فرض آن را تعیین می کند. برای کلیدهای فشاری Submit و Reset مقدار گزینه Value عنوانی است که بر روی کلیدها ظاهر خواهد شد.
- برچسب <Script Language=...> : از این برچسب برای اسکریپت نویسی در بطن صفحه وب استفاده می شود. به کمک این برپسب می توان پاره ای اسکریپت در درون یک صفحه وب جاسازی کرد تا در سمت مرورگر تفسیر و اجرا شود. از

اسکریپتها می توان برای ایجاد جلوه های ویژه و انجام برخی از پردازش های محلی (بر روی ماشین کاربر ) بهره گرفت. در جلوی گزینه Language زبان اسکریپتی مورد استفاده تعیین می شود؛ مهمترین زبان های اسکریپت نویسی(برای اجرا در سمت مرورگر) عبارتند از :

- ✓ Javascript
- ✓ VBscript
- ✓ Jscript
- ✓ XML

## DHTML : HTML پویا

زبان نشانه گذاری DHTML در حقیقت نسخه مبتنی بر شئ از HTML است. در این زبان نشانه گذاری، قابلیت های جدیدی تعریف شده است که بر اساس آن می توان کنترل بیشتری بر روی مؤلفه های موجود در یک صفحه وب اعمال کرد و بتوان به صفحه وب جلوه های ویژه(مبتنی بر شرایط مرورگر) بخشید. به عنوان مثال می توان بر روی حرکات ماوس در صفحه وب نظارت کرد و بر اساس موقعیت اشاره گر ماوس جلوه های خاصی ایجاد کرد؛ به عنوان مثال با حرکت ماوس از روی یک برچسب تصویر، یک انیمیشن شروع به اجرا کند یا فونتها تغییر رنگ، اندازه، نوع و حتی تغییر زبان بدهند. بیشتر امکانات DHTML در راستای ایجاد جلوه های بصری، کنترل حرکات ماوس و تغییر پویای رنگها هستند. در درون صفحات DHTML که عموماً با پسوند dhtml. ذخیره می شوند، از زبانهای اسکریپت نویسی مثل VBsript یا JavaScript استفاده می شود. خوسبختانه DHTML امکانات خوبی برای سپردن کنترل مؤلفه های صفحه وب به اسکریپتها عرضه کرده است. این اسکریپتها در هنگام نمایش صفحه وب خط به خط توسط مرورگر اجرا می شوند.

## XML

زبان نشانه گذاری HTML، یک زبان آزاد و بدون ساختار است. بدین معنا که در HTML محتوای متنی صفحه وب و برچسبهای وب در هم مخلوط هستند و دست طراح وب برای استفاده از برچسبهای وب و درج هرگونه متن دلخواه در هر کجای صفحه باز است. به عنوان مثال فرض کنید که در یک صفحه وب قیمت، شماره و نام تعدادی کالا درج شده باشد. فقط کاربر می تواند حدس بزند که کدام فیلد متنی شماره کالا، کدام قیمت و کدام نام کالاست

درحالی که از دیدگاه یک برنامه نرم افزاری تمام این فیلدها یک رشته متنی درهم تنیده هستند و تفکیک آنها از یکدیگر چندان ساده نیست. به عبارتی ماهیت و نوع اطلاعات جاسازی شده در یک صفحه وب مشخص نیست و همه از نوع متن ساده هستند. این قضیه کار را برای موتورهای جستجو دشوار می کند چرا که وقتی یک کاربر دنبال یک نام کالا می گردد، موتور جستجو مجبور است محتوای سرتاسر صفحات وب را بگردد و هر کلمه ای که با نام مورد نظر کاربر مطابقت دارد، آن را به عنوان نتیجه برگرداند. در ضمن مطمئن نیست که آیا کلمه ای که با نام مورد نظر کاربر مطابقت دارد واقعاً نام کالاست یا کلمه ای مشابه است که در خصوص موضوعی دیگر در جایی از صفحه وب ظاهر شده است. در ذهن خود فرض کنید کاربری در موتور جستجوی گوگل دنبال کالایی با نام panda بگردد تا متوجه شوید چه بلایی بر سر او می آید! در یک جمله مختصر : در صفحات وب HTML، رشته های متنی، صرفاً متنی هستند و ماهیت آنها مشخص نیست.

کنسرسیوم جهانی وب، W3C، برای ساختار دادن به صفحات HTML، استاندارد جدیدی وضع کرد که به کمک آن می توان ترکیب و ماهیت داده های متنی یک صفحه وب را مشخص کرد تا کار پردازش خودکار صفحات وب ساده تر شود. این کنسرسیوم در استاندارد جدید، دو زبان مکمل و همراه تعریف کرده است : زبان اول XML است که محتویات صفحه وب را به صورت ساختاریافته توصیف می نماید. زبان دوم XSL است که در حقیقت شیوه نمایش محتویات یک فایل XML را مشخص می نماید. این دو زبان پیچیده و مفصل هستند و برای آشنایی با آنها باید به مراجع فنی آنها مراجعه شود.

هسته زبان XML به «تعریف نوع سند» (Document Type Definition) اختصاص دارد که به اختصار DTD گفته می شود. در حقیقت DTD چیزی شبیه کتابخانه تعریف انواع داده در بانک های اطلاعاتی رابطه ای (Relational Database) است. یکی DTD برچسبهای شروع و ختم یک فایل XML را تعریف می کند و امکان آن را فراهم می آورد تا کسی که مؤلفه های درونی یک فایل XML را مشاهده می نماید احساسی از نوع و ماهیت داده به دست بیاورد. به عنوان مثال در زیر یک ساختمان داده برای معامله خودرو تعریف کرده ایم که دارای DTD ذیل است :

```
!ELEMENT Inventory (car*)>
!ELEMENT Car (Make, Model, Color, Owner*)>
!ELEMENT Make (#PCDATA)>
!ELEMENT Model (#PCDATA)>
```

```
!ELEMENT Color (#PCDATA)>
!ELEMENT Owner (#PCDATA)>
```

فایل XML که داده هایی از نوع فوق را در یک صفحه وب تعریف و مقداردهی می کنـد در زیر آمده است:

```
<?xml version=1.0" ?>
<!DOCTYPE Inventory PUBLIC   "."    "Inventory.dtd" >
<Inventory>
        <Car>
                <Make>Honda</Make>
                <Model>Civic<?Model>
                <Color>Red<?Color>
                <Owner>Jack Scanly</Owner>
                <Owner>Jane Scanly</Owner>
        </Car>
        <Car>
                <Make>Nissan</Make>
                <Model>Maxima<?Model>
                <Color>Black<?Color>
                <Owner>Mike Smith</Owner>
        </Car>
</Inventory>
```

کد فوق یک مجموعه بسیار کوچک از فهرست خودروهای موجود بـرای خریـد و فـروش را تعریف کرده است. از مجموعه داده فوق دو خودرو با مشخصات (هوندا،سیویک،قرمز رنـگ) و (نیسان، ماکسیما، سیاه رنگ) تعریف شده است. دقت کنید کـه خـودروی هونـدا دارای دو مالک مشترک است: Jack و Jane. علامت * در کنار آیتم owner اجازه می دهـد کـه یـک عضو از مجموعه داده دارای چندین ویژگی مثل <Owner>، با مقدار متفاوت باشد:

```
!ELEMENT Car (Make, Model, Color, Owner*)>
```

کل کاری که یک فایل XML انجام می دهد تعریفـی سـاختاریافته از مجموعـه داده هاسـت ولیکن درون این فایل هیچ اطلاعاتی در مورد نحوه نمایش آن بر روی صـفحه مرورگـر دیـده نمی شود. برای قالب دادن و تعریف شکل نمایش بصری این اطلاعات، بـه فایـل دیگـری بـا پسوند XSL نیاز است. این فایل در حقیقـت شـیوه نامـه ای (Casecade StyleSheet) در خصوص چگونگی ظاهر شدن این مجموعه اطلاعات روی صفحه نمایش است.

## XHTML (HTML توسعه یافته)

براساس اظهارات کنسرسیوم جهانی وب، HTML 4 آخرین نسخه زبان فراگیر و جهانی HTML خواهد بود و نسخه پنجمی برای آن ارائه نخواهد شد. در عوض، نسخه پنجم زبان HTML با عنوان XHTML معرفی شده است که درحقیقت بازنویسی و فرموله سازی HTML طبق قواعد XML است. به عنوان مثال در این زبان برچسب <h1> که در HTML وجود داشت، هیچ معنایی ندارد مگر آنکه در یک فایل XSL به درستی تعریف شده باشد.

XHTML برخلاف HTML سختگیر و دقیق است و انعطاف زیادی از خود نشان نمی دهد چراکه قرار بوده XHTML به معنای واقعی کلمه قابلیت جابجایی (portability) و عدم وابستگی به نرم افزار یا سخت افزار داشته باشد تا بتوان از آن در دستگاههایی مثل تلفنهای همراه و PDA نیز استفاده کرد و اگر قرار باشد که این زبان از خود انعطاف نشان بدهد، کار مرورگر را در تفسیر و نمایش چنین فایل هایی مشکل می کند و منجر به حجیم و و پیچیده تر شدن نرم افزار مرورگر و بروز اشکالات ناخواسته می شود.

در این زبان تمام برچسب ها و ویژگیهای زبان (Attributes) بایستی با حروف کوچک نوشته شوند و دست طراح وب برای استفاده اختیاری از حروف کوچک و بزرگ باز نیست. طبعاً این سخت گیری حجم عملیات مرورگر را کاهش می دهد. همچنین برخلاف HTML ، هر برچسبی باید اجباراً در جایی لغو شود یعنی مثلاً اگر در فایلی در جایی برچسب  وجود دارد بایستی یک  در جایی دیگر داشته باشد و گرنه فایل مربوطه از درجه اعتبار ساقط است. همچنین تودرتویی برچسب ها باید به ترتیب باشد. برای آگاهی بیشتر از XHTML به وب سایت www.w3c.com مراجعه نمایید.

## جاوااسکریپت (Java Script)

«جاوااسکریپت» زبانی است که نیازی به کامپایلر ندارد بلکه کدهای آن در بطن صفحات وب جاسازی شده و توسط یک مفسر(Interpreter) که در مرورگر قرار دارد تفسیر و اجرا می شوند. این زبان را شرکت NetScape ابداع و پیاده سازی کرده است. شاید برایتان جالب باشد که بگوییم ارتباط بین زبان اصلی جاوا و جاوااسکریپت فقط در نام آنهاست : جاوا زبانی نیازمند کامپایلر و شئ گرا و تا حدودی حرفه ای و دشوار است در حالی که جاوااسکریپت زبانی ساده، شبه شئ گرا و نیازمند مفسر است که صرفاً برای توسعه وب طراحی شده است.

**هرچند جاوااسکریپت قابلیت های چندی در خصوص شئ گرایی دارد ولیکن بیشتر به زبان های غیر شئ گرا مثل Perl شبیه است. از جاوااسکریپت برای ایجاد صفحات وب پویا شامل مؤلفه های بصری متعدد و منعطف (مثل پنجره ها، کلیدها، منوها، پنجره های انتخاب)، ارزیابی داده های ورودی کاربر(از لحاظ اعتبار سنجی) و اجرای محاسبات سمت کاربر، مورد استفاده قرار می گیرد.**

# فصل دوم : استفاده از شئ XMLHttpRequest

اکنون که تاریخچه ای از برنامه های کاربردی پویای وب[۲] ارائه کردیم و مقدمه ای بر Ajax بیان شد، وقت آن رسیده است که به اصل مطلب بپردازیم: چگونه از شئ XMLHttpRequest استفاده کنیم؟ Ajax بیش از آن که یک تکنولوژی باشد یک تکنیک است. درصورتیکه مرورگرها از شئ XMLHttpRequest پشتیبانی نمی کردند، google suggest و سایت هایی شبیه به آن به گونه ای که الان می شناسیم وجود نمی داشتند و شما هم مشغول خواندن این نوشته ها نبودید.

XMLHttpRequest به صورت اکتیوکس در مرورگر اینترنت اکسپلورر نسخه ۵ پیاده سازی شده است. این مهم است که بدانید XMLHttpRequest جزء استانداردهای W3C نیست، هرچند بیشتر کارکردهای آن در پیش نویس استاندارد جدید آن یعنی Dom level 3 load and save specification پوشش داده شده است. به خاطر همین استاندارد نبودن، رفتار این شئ از مرورگری به مرورگر دیگر ممکن است کمی متفاوت باشد هرچند بیشتر متدها و خاصیتهای آن به طور گسترده و به درستی پشتیبانی می شود. هم اکنون مرورگرهای firefox,safari,opera,IE[3] همگی رفتار شئ XMLHttpRequest را به گونه ای یکسان پیاده سازی می کنند.

بنا به آنچه گفته شد، اگر تعداد قابل توجهی از کاربران، به سایت شما از طریق مرورگرهای قدیمی دسترسی دارند، شما نیازمند بررسی گزینه های مختلفی در این زمینه هستید. آمار کاربردها نشان می دهد که جزء کمی از مرورگرهایی که امروزه استفاده می شوند فاقد پشتیبانی از XMLHttpRequest هستند و شانس اینکه این مسئله به عنوان مشکل باقی بماند کم است. به هرحال شما باید چک کنید که چه دسته هایی از کاربران در حال استفاده از سایت شما هستند.

---

[2] - dynamic web application

[3] - Internet Explorer

## بررسی شئ XMLHttpRequest

شما قبل از اینکه با استفاده از این شئ بتوانید درخواستی را بفرستید یا جوابی را پردازش کنید باید با استفاده از جاوااسکریپت یک نمونه از این شئ را بسازید. چون XMLHttpRequest استاندارد W3C نیست، می توانید از جاوااسکریپت به دو طریق برای ساخت آن استفاده کنید. IE شئ XMLHttpRequest را به صورت اکتیوکس پیاده سازی کرده است و بقیه مرورگرها نظیر firefox,opera,safari آن را به صورت اشیاء داخلی کتابخانه های جاوااسکریپت پیاده سازی کرده اند. به دلیل همین تعارض، کد جاوااسکریپتی که برای ساخت شئ XMLHttpRequest می نویسید باید شرطی جهت ساخت از طریق اکتیوکس یا اشیاء داخلی جاوااسکریپت داشته باشد. البته به این منظور لازم نیست که تشخیص دهید کاربر از کدام مرورگر استفاده می کند بلکه کافی است مشخص کنید که مرورگر وی از اشیاء اکتیوکس پشتیبانی می کند یا خیر. اگر مرورگر از اشیاء اکتیوکس پشتیبانی می کند از این طریق اکتیوکس یک شئ XMLHttpRequest بسازید در غیر اینصورت از طریق معمولی یعنی ساخت اشیاء داخلی جاوااسکریپت اقدام کنید. کد ۲-۱ ساده ترین راه جهت ایجاد یک شئ XMLHttpRequest بر اساس نوع مرورگر کاربر را نشان می دهد.

کد ۲-۱ : ساخت نمونه ای از شئ XMLHttpRequest با جاوااسکریپت

```
var xmlHttp;
function createXMLHttpRequest() {
   if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
   }
}
```

همانگونه که می بینید ساخت شئ XMLHttpRequest بسیار ساده است. ابتدا یک متغیر عمومی[۴] جهت نگهداری شئ XMLHttpRequest تعریف می کنید.(در این مثال با نام xmlHttp تعریف شده است.) متد[۵] CreateXMLHttpRequest در حقیقت تنظیماتی که جهت ساخت این شئ لازم است را انجام می دهد. این متد شامل یک شرط منطقی است که

4 - global
5 - Method

نشان می دهد از چه روشی برای ایجاد شئ استفاده شود. فراخوانی Window.activeXObject در نتیجه یک شئ یا مقدار Null را برمی گرداند و هنگامی که در شرط if استفاده شود مشخص می کند که مرورگر از اکتیوکس پشتیبانی می کند یا خیر، یعنی اینکه مرورگر کاربر IE می باشد یا نه. در صورت صحت شرط، شئ XMLHttpRequest از طریق فراخوانی سازنده اشیاء اکتیوکس و با فرستادن ورودی مشخص از نوع String که مشخص می کند شئ اکتیوکس مورد نظر باید از نوع XMLHttpRequest باشد، ایجاد می شود. در این مورد مقدار Microsoft.XMLHTTP به این سازنده فرستاده شده است، به این معنی که شئ باید از نوع XMLHttpRequest ساخته شود. اگر فراخوانی window.ActiveXObject مقدار false برگرداند قسمت else اجرا می شود که نشان دهنده استفاده از روش عادی ایجاد اشیاء در جاوااسکریپت است. اگر window.XMLHttpRequst وجود داشته باشد یک شئ از نوع XMLHttpRequest ساخته می شود.

با توجه به اینکه جاوااسکریپت به صورت داینامیک و بدون وابستگی به مرورگر اشیاء را مدیریت می کند شما می توانید بدون توجه به اینکه شئ XMLHttpRequest از طریق کدام یک از روشهای بالا ساخته شده، به خصوصیات و متدهای آن دسترسی داشته باشید. این مسئله روند پیشرفت را ساده تر می کند.

جدول ۲-۱ برخی از متدهای شئ XMLHttpRequest را نشان می دهد. در ادامه درباره هر یک از آنها به طور مفصل صحبت خواهیم کرد.

جدول ۲-۱ : متدهای استاندارد XMLHttpRequest

| شرح متد | نام متد |
|---|---|
| خاتمه درخواست[6] جاری | abort() |
| بازیابی همه سرآیندهای[7] پاسخ به صورت جفت های key/value | getAllResponseHeaders() |
| بازیابی مقدار یک سرآیند مشخص | getResponseHeader("header") |
| تنظیم مقادیر جهت فراخوانی سرور | open("method", "url") |

6 - Request
7 - Header

| | |
|---|---|
| ارسال درخواست به سرور | send(content) |
| مقداردهی سرآیند مشخص با مقدار داده شده | setRequestHeader("header","value") |

اجازه دهید نگاه دقیقتری به هریک از این متدها داشته باشیم:

void open(string method, string url, boolean asynch, string username, string password) : این متد تنظیمات لازم جهت فراخوانی سرور را انجام می دهد. این متد به منظور مقداردهی اولیه در یک درخواست استفاده می شود که دو آرگومان اصلی و سه آرگومان اختیاری دارد. در پارامار اول شما باید روش فراخوانی( Get یا Post) و در پارامتر دوم آدرس[۸] منبعی که جهت فراخوانی لازم دارید را مشخص نمایید. در پارامتر سوم می توانید به صورت دلخواه مقداری از نوع Boolean را که نشان دهنده همگام[۹] یا ناهمگام[۱۰] بودن فراخوانی است را به متد بفرستید، مقدار پیش فرض، در صورتی که مقداری را برای این پارامتر مشخص نکنید، True می باشد و نشان دهنده این است که درخواست در حالت طبیعی به صورت ناهمگام می باشد. در صورتیکه مقدار False را به این پارامتر بدهید، پردازش تا زمانی که جوابی از سرور دریافت نشود متوقف خواهد شد. از آنجایی که فراخوانی ناهمگام یکی از مهمترین فواید استفاده از آژاکس می باشد، مقدار False دادن به این پارامتر سبب نقض دلایل استفاده از شئ XMLHttpRequest می شود. البته ممکن است در بعضی مواقع نظیر چک کردن ورودی های کاربر قبل از فرستادن صفحه، این کار مفید واقع شود. دو پارامتر دیگر به اندازه کافی گویا هستند و به شما اجازه می دهند که نام کاربری و رمز عبور جهت اتصال به آدرس داده شده را مشخص نمایید.

Void Send(Content) : این متد درخواست آماده شده را به سرور می فرستد. اگر درخواست به صورت ناهمگام تعریف شده باشد، این متد فوراً نتیجه را برمی گرداند در غیر اینصورت تا زمانی که پاسخ[۱۱] مربوط به فرم موجود در آن صفحه برسد منتظر می ماند[۱۲]. مقدارآرگومان ورودی این متد می تواند نمونه ای از شئ DOM[13]، یک استریم[۱۴] ورودی یا

---

[8] - URL

[9] - synchronous

[10] - asynchronous

[11] - Response

[12] - هر صفحه شامل حداقل یک فرم فعال است که با اجرای هریک از متدهای مشخص شده آن، کل اطلاعات آن فرم به صورت یک درخواست آماده شده و به سرور ارسال می شود و سرور نیز پاسخ مربوطه را درقالب یک صفحه جدید بر می گرداند.

[13] - Document Object Model

یک رشته کارکتری[۱۵] باشد. محتوای مشخص شده(Content) به عنوان بخشی از بدنه درخواست ارسال می شود.

void setRequestHeader(string header, string value) : این متد مقدار value را به عنوان مقدار سرآیند داده شده فرار می دهد. این متد رشته کارکتری که مشخص کننده سرآیندی است که باید مقداردهی شود و رشته کارکتری که بیانگر مقدار جدید آن است را به عنوان ورودی می گیرد. توجه داشته باشید این متد باید بعد از فراخوانی متد Open، فراخوانی شود.

از بین متدهای گفته شده، بیشترین استفاده را از متدهای Open و Send خواهید داشت.

void abort() : این متد همانگونه که از نامش پیداست، ارسال درخواست را متوقف می سازد.

string getAllResponseHeaders() : کاربرد این متد برای برنامه نویسان وب آشنا است. این متد رشته ای کارکتری که حاوی سرآیندهای پاسخ یک درخواست است را برمی گرداند. سرآیندها حاوی مشخصاتی مثل Content-length, date, URI مربوطه می باشند.

string getResponseHeader(string header) : این متد تقریباً شبیه متد قبلی است با این تفاوت که آرگومان ورودی آن سرآیندی را مشخص می کند که باید مقدار آن به صورت رشته کارکتری برگردانده شود.

علاوه بر این متدها، شئ XMLHttpRequest خصوصیاتی دارد که در جدول ۲-۲ لیست شده اند. شما هنگام کار کردن با شئ XMLHttpRequest به دفعات از این خصوصیات بهره خواهید برد.

جدول ۲-۲ : خصوصیات استاندارد XMLHttpRequest

| نام خاصیت | شرح خاصیت |
|---|---|
| onreadystatechange | روالی که با تغییر حالت درخواست اجرا می شود، معمولاً نام یک تابع جاوااسکریپت می باشد. |
| readyState | وضعیت درخواست را مشخص می کند و پنج مقدار زیر را می تواند داشته باشد. 0 = uninitialized,1 = loading , |

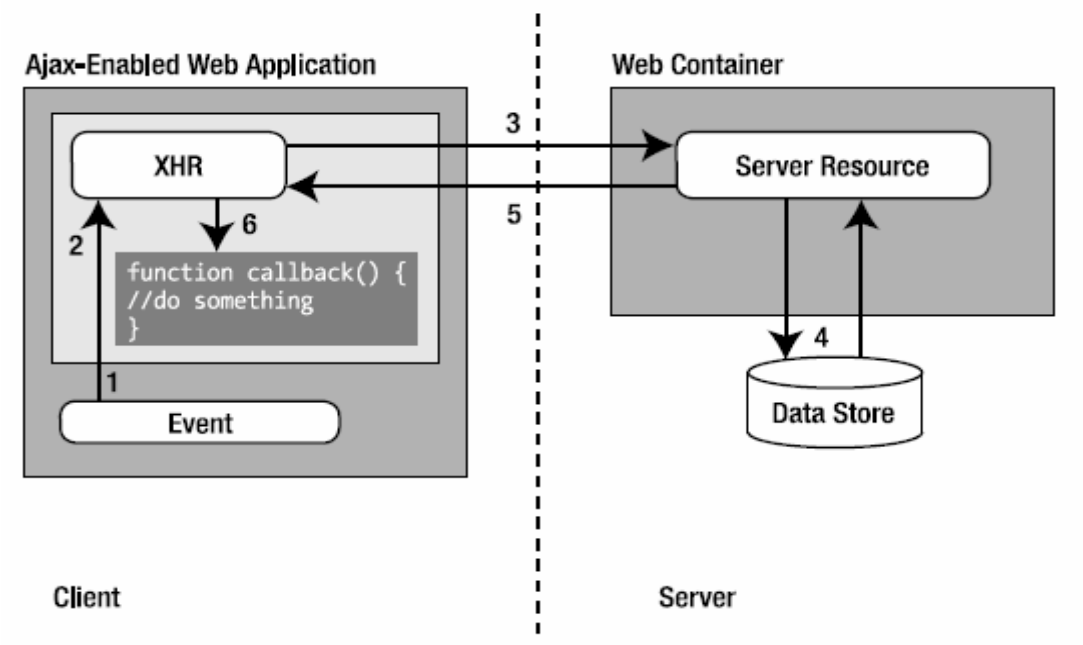
[14] Stream -
[15] String -

2 = loaded , 3 = interactive, 4=complete

| | |
|---|---|
| پاسخ سرور به صورت متنی | responseText |
| پاسخ سرور به صورت XML | responseXML |
| کد عددی وضعیت سرور در استاندارد HTTP شامل مقادیر زیر<br>200=OK , 404=Not Found , … | status |
| کد کارکتری وضعیت سرور در استاندارد HTTP شامل موارد زیر<br>OK , Not Found , … | statusText |

## یک تعامل[16] نمونه

اکنون حتماً علاقه مند هستید که بدانید تعاملات هنگام استفاده از آژاکس چگونه است. شکل ۲-۱ الگوی تعاملی استاندارد در برنامه های آژاکس را نشان می دهد.




شکل ۲-۱ : تعامل استاندارد آژاکس

[16] Interaction -

برخلاف رهیافت استاندارد درخواست/پاسخ که در کلاینت های معمول وب مشاهده می شود، برنامه آژاکس کارها را با اندکی تفاوت انجام می دهد. مراحل مختلف این تعامل که در شکل بالا نشان داده شده اند را در ادامه توضیح می دهیم.

۱. یک رویداد[۱۷] سمت کاربر، باعث فعال شدن یک رویداد آژاکس می شود. هر چیزی می تواند آن را فعال کند، از یک رویداد OnChange ساده گرفته تا اعمال خاص کاربر. ممکن است این مورد کدی شبیه زیر داشته باشد:

```
<input type="text"d="email" name="email" onblur="validateEmail()";>
```

۲. نمونه ای از شئ XMLHttpRequest ساخته می شود. با استفاده از متد Open فراخوانی مورد نظر تنظیم می شود. درخواست، هنگامی فعال می شود که متد Send اجرا شود. این مورد کدی شبیه زیر می تواند داشته باشد:

```
var xmlHttp;
function validateEmail() {
      var email = document.getElementById("email");
      var url = "validate?email=" + escape(email.value);
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
      xmlHttp.open("GET", url);
      xmlHttp.onreadystatechange = callback;
      xmlHttp.send(null);
}
```

۳. درخواست به سرور فرستاده می شود. درخواست ممکن است شامل فراخوانی یک سرولت، یک اسکریپت CGI , ASP.NET یا هر تکنیک سمت سرور باشد.

۴. در سمت سرور هر عملی که بخواهید می توانید انجام دهید مثل دسترسی به پایگاه داده یا هر سیستم اطلاعاتی دیگر.

۵. پاسخ درخواست به سمت مرورگر مربوطه برگردانده می شود. محتوای پاسخ می تواند Text یا XML باشد. شئ XMLHttpRequest نتایج را فقط به این دو صورت می تواند پردازش کند. در نمونه های پیچیده تر پاسخ می تواند شامل کد جاوااسکریپت، تنظیمات DOM یا دیگر تکنولوژی های ارتباطی باشد. توجه داشته باشید که شما باید مقادیر سرآیندها

[17] - trigger

را تنظیم کنید زیرا مرورگرها نتایج را به صورت محلی از سرآیندها دستیابی می کنند. کد این تنظیمات در زیر آمده است:

```
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
```

۶. در مراحلی که طی شد شئ XMLHttpRequest به گونه ای پیکربندی گردید که هنگام بازگشت پاسخ سرور، تابع CallBack فراخوانی شود. این تابع مقدار خاصیت ReadyState شئ XMLHttpRequest را چک می کند و سپس مقدار کد status را که سرور برگردانده چک می کند. با توجه به نتایج به دست آمده تابع Callback کارهای مورد نیاز را روی کلاینت اجرا می کند. به عنوان مثال تابع callback می تواند به صورت زیر باشد:

```
function callback() {
      if (xmlHttp.readyState = = 4) {
            if (xmlHttp.status = = 200) {
                  //do something interesting here
            }
      }
}
```

همانگونه که دیدید، این روش با الگوی معمولی درخواست/پاسخ تفاوت دارد اما نه به قدری که کلاً برای برنامه نویسان وب عجیب و غریب به نظر برسد. در آینده قسمت های ساخت و تنظیم شئ XMLHttpRequest و تابع Callback را فراوان تکرار خواهید کرد، هرچند که می توانید آنها را به کتابخانه ای تبدیل کرده و از آن در برنامه هایتان استفاده کنید یا اینکه از کتابخانه های نوشته شده و آماده که روی وب یافت می شوند استفاده کنید. این یک فیلد کاری و مطالعاتی جدید است و فعالیت های قابل توجه مفیدی در گروههای طرفدار Open Source برای آن انجام گرفته است.

به طور کلی، Framework ها و ابزارهای متنوعی روی وب قابل دسترس است که شما را از نوشتن کدهای ساده تکراری بی نیاز می کنند، تعدادی از آنها چندین کامپوننت رابط کاربر را نیز شامل می شوند. تعداد زیادی از این فریم ورک ها در مراحل اولیه توسعه هستند. کتابخانه های کاملی مثل libXmlRequest, RSLite sarissa, JavaScript Object Notation (JSON), JSRS و Atlas در دسترس هستند.

### Post یا Get :

ممکن است علاقه مند باشید که تفاوت های Get و Post را بدانید و اینکه چه زمانی باید از هر یک از آنها استفاده کرد. به صورت نظری، هنگامی درخواست به صورت idempotent باشد، به این معنی که چندین درخواست نتیجه یکسانی داشته باشد، از روش **Get** استفاده می کنیم. تفاوت عمده این دو روش در مفهومی به نام payload نهفته است. در بسیاری موارد، مرورگرها و سرورها محدودیت هایی را روی اندازه URL که برای فرستادن داده ها به سرور استفاده می شود، قرار می دهند. به طورکلی، از Get برای بازیابی اطلاعات از سرور استفاده کنید و از تغییر حالت سرور با استفاده از فراخوانی متد Get خودداری کنید. از متد Post هنگامی که می خواهید حالت سرور را تغییر دهید استفاده کنید.

تفاوت دیگز این دو روش این است که روش Post محدودیتی روی حجم اطلاعاتی که به سرور فرستاده می شود ایجاد نمی کند، در حالیکه هنگام استفاده از روش Get در حجم اطلاعات ارسالی به سرور محدودیت وجود دارد.

## نحوه ارسال یک درخواست ساده

اکنون شما برای استفاده از شئ XMLHttpRequest آماده اید. تا اینجا فقط توضیح دادیم که چگونه یک شئ XMLHttpRequest بسازیم و اکنون می توانیم نشان دهیم که چگونه یک درخواست به سرور بفرستیم و پاسخ آن را پردازش کنیم.

ساده ترین درخواستی که می توانید بسازید، درخواستی است که هیچ گونه اطلاعاتی در قالب پارامترهای QueryString به سمت سرور نمی فرستد. به مرور خواهید دید معمولاً به فرستادن اطلاعاتی به سمت سرور نیاز خواهید داشت. مراحل ابتدایی جهت ارسال درخواست با استفاده از شئ XMLHttpRequest به قرار زیر است :

داشتن نمونه ای از شئ XMLHttpRequest، حال یا با ایجاد یک شئ جدید یا با استفاده از متغیری که از قبل نمونه ای از شئ XMLHttpRequest در خود نگهداری می کند.

مشخص کردن تابعی که تغییرات حالت شئ XMLHttpRequest را کنترل می کند. این کار را از طریق تنظیم خصوصیت onReadyStateChange مربوطه به شئ XMLHttpRequest به نام یک تابع جاوااسکریپت انجام می دهیم.

مقداردهی کردن خصوصیات درخواست. متد open شئ XMLHttpRequest این کار را انجام می دهد. قبلاً گفته شد که این متد سه آرگومان ورودی دارد، رشته کارکتری که روش ارسال(post,get) را مشخص می کند، رشته کارکتری که آدرس مقصد را نشان می دهد و مقدار boolean که نشان می دهد که آیا درخواست باید به صورت همگام یا ناهمگام باشد.
و در نهایت، ارسال درخواست به سرور. متد send درخواست را به مقصد مشخص شده ارسال می کند. متد send یک آرگومان ورودی دارد که معمولاً یک رشته کارکتری یا یک شئ DOM می باشد. این پارامتر به عنوان بخشی از بدنه درخواست به آدرس مقصد ارسال می شود. هنگامی که آرگومانی را جهت ارسال به سرور به متد send می فرستید، مطمئن شوید که روش ارسال در متد open حتماً نوع post انتخاب شده باشد. در صورتیکه نیازی به ارسال چیزی به سرور ندارید مقدار آرگومان متد send را Null قرار دهید.
به نظر می رسد دادن توضیحاتی در مورد خصوصیت onReadyStateChange لازم باشد. این خصوصیت نام یک تابع را می پذیرد. این تابع هنگامی اجرا می شود که حالت داخلی شئ XMLHttpRequest تغییر کند. اگر درخواست از نوع ناهمگام باشد، درخواست فوراً ارسال و پاسخ مربوطه جهت پردازش مورد استفاده قرار می گیرد. وقتی که درخواست ارسال شد خصوصیت readyState ممکن است مقادیر عددی مختلفی بگیرد که هریک نشان می دهد درخواست در چه حالت و مرحله ای قرار دارد (به جدول ۲-۲ مراجعه شود) اما حالتی که بیشتر چک می شود این است که آیا سرور ارسال پاسخ به این درخواست را تمام کرده است یا خیر. به طور خلاصه، با مقداردهی کردن خصوصیت onReadyStateChange به طور عامیانه به شئ XMLHttpRequest می گویید که این تابع مشخص را هنگام رسیدن پاسخ سرور اجرا کن.

## یک نمونه درخواست ساده

اولین مثال ما بسیار ساده است، یک صفحه ساده با یک دکمه[۱۸] روی آن. فشردن دکمه باعث ایجاد و ارسال یک درخواست به سمت سرور می شود. سرور با ارسال یک فایل متنی ساده به سمت کلاینت پاسخ می دهد. پاسخ سرور با نمایش محتویات فایل متنی رسیده، در یک پنجره

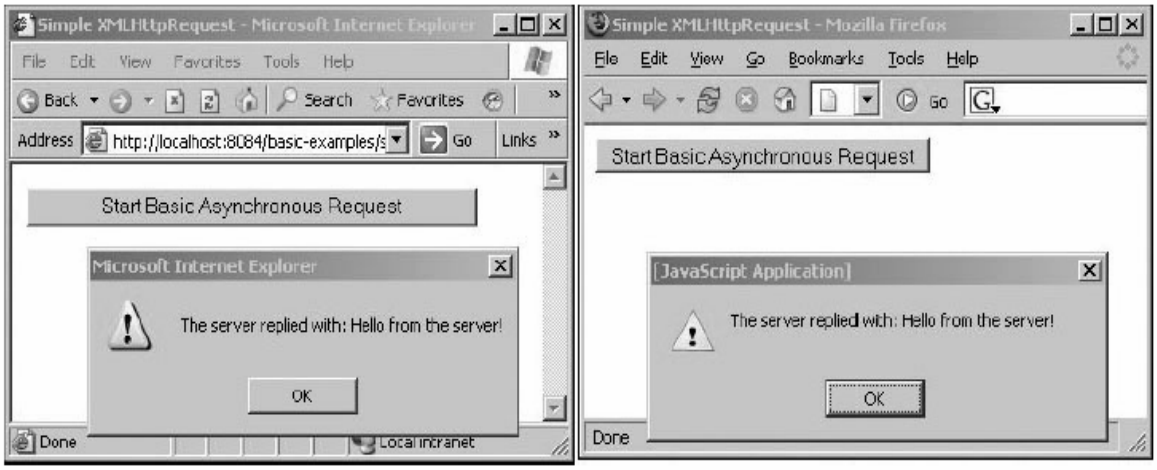
[18] - button

پیام جاوااسکریپت، مشخص می شود. کد ۲-۲ فایل html مربوطه و کـدهای جاوااسـکریپت لازم را نشان می دهد.

کد ۲-۲ : صفحه وب simpleRequest.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple XMLHttpRequest</title>
<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest() {
   if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
   }
}
function startRequest() {
   createXMLHttpRequest();
   xmlHttp.onreadystatechange = handleStateChange;
   xmlHttp.open("GET", "simpleResponse.xml", true);
   xmlHttp.send(null);
}
function handleStateChange() {
   if(xmlHttp.readyState == 4) {
      if(xmlHttp.status == 200) {
         alert("The server replied with: " + xmlHttp.responseText);
      }
   }
}
</script>
</head>
<body>
<form action="#">
        <input type="button" value="Start Basic Asynchronous Request"
        onclick="startRequest();"/>
   </form>
</body>
</html>
```

فایل پاسخ سرور، simpleResponse.xml، فقط حاوی یک خط متن دلخواه است. فـشردن دکمه روی صفحه، باید پنجره پیام که حاوی متن درون این فایل است را نـشان دهـد. شـکل ۲-۴ پاسخ سرور را در مرورگرهای FireFox و IE نشان می دهد.




**شکل ۲-۴ : اولین درخواست ناهمگام**

**ارسال درخواست به صورت ناهمگام به سمت سرور به مرورگر اجازه می دهد تا تعامل با کاربر تا رسیدن پاسخ سرور که در پشت صحنه منتظر آن است، ادامه یابد. درصورتی که اگر از روش همگام استفاده شود و رسیدن پاسخ سرور چندین ثانیه به طول بیانجامد، در این مدت مرورگر قادر به ادامه تعامل با کاربر نخواهد بود. روش اول (ناهمگام) به کاربر اجازه می دهد به کارش با مرورگر هنگامی که مرورگر در پشت صحنه منتظر پاسخ سرور است، ادامه دهد.**

**قابلیت ارتباط با سرور بدون متوقف کردن جریان کاری کاربر با مرورگر، تکنیکهای متنوعی را برای افزایش تجربه و علاقه کاربر هنگام کار با برنامه ها ایجاد می کند. به عنوان مثال برنامه ای را جهت کنترل ورودی های کاربر فرض کنید. هنگامی که کاربر مشغول پر کردن فیلدها در فرم ورود اطلاعات است مرورگر می تواند به صورت متناوب مقادیر وارد شده را به منظور چک کردن به سرور بفرستد، بدون ایجاد وقفه در ورود بقیه اطلاعات توسط کاربر. اگر تعیین اعتبار یک ورودی با شکست مواجه شود، کاربر فوراً مطلع می شود، قبل از اینکه تمام فرم جهت پردازش به سرور فرستاده شود. صرفه جویی در زمان کاربر و کاهش بار روی سرور از فواید استفاده از این روش است چون دیگر لازم نیست فرمهایی که حاوی اطلاعات نادرست هستند کلاً به سرور فرستاده شوند و پس از پردازش، خطای مربوطه بدون هیچ گونه نتیجه قابل توجه دیگری به سوی کاربر فرستاده شود.**

## سخنی کوتاه درباره امنیت

هر بحثی در رابطه با تکنولوژی های وابسته به مرورگرها بدون اشاره به مبحث امنیت، کامل نیست. هر منبعی که توسط شئ XMLHttpRequest درخواست می شود، باید در دامنه ای که فراخوانی اولیه از آن ایجاد شده، مستقر باشد. این محدودیت امنیتی از فراخوانی منبعی که در دامنه مربوط به صفحه ای که شئ XMLHttpRequest ساخته، جلوگیری می کند. ( به طور عامیانه در صورتی که شئ XMLHttpRequest توسط صفحه ای از وب سایت با آدرس **a** ایجاد شده باشد، این شئ نمی تواند به منابعی غیر از منابع موجود در آدرس **a** دسترسی داشته باشد.)

شدت این محدودیت در مرورگرهای مختلف فرق می کند. اینترنت اکسپلورر در اینگونه مواقع با نمایش پنجره حاوی اخطار به کاربر اجازه می دهد که ارتباط را ادامه دهد یا متوقف سازد در صورتیکه مرورگر firefox به سادگی با اعلام پیغام خطا در کنسول جاوااسکریپت خود، درخواست را متوقف می کند. هرچند firefox هم راهکارهایی به منظور اجازه دادن به شئ XMLHttpRequest برای دسترسی به منابع خارج از دامنه ارائه می کند اما در کل بهتر است روی این نمونه دستیابی حساب باز نکنید.

# فصل سوم : ارتباط با سرور،ارسال درخواست و پردازش پاسخ

قسمت مفرح داستان آژاکس اکنون شروع می شود. اکنون وقت آن است که دانش جدیدی که از شئ XMLHttpRequest به دست آورده اید، به کار ببندید. با استفاده از چندین مثال ساده نشان خواهیم داد با استفاده از شئ XMLHttpRequest چگونه یک درخواست را به سرور ارسال کرده و چگونه با جاوااسکریپت پاسخ سرور را پردازش کنیم.

مثالهای این فصل از برنامه نویسی مشکل سمت سرور جهت پردازش درخواست و تهیه پاسخ استفاده نمی کنند. در عوض، مثالها از فایلهای متنی ساده جهت شبیه سازی پاسخ سرور استفاده می کنند. حذف این پیچیدگی ها به شما اجازه می دهد تا روی اینکه در مثال در مرورگر چه اتفاقی می افتد، متمرکز شوید.

## پردازش پاسخ سرور

شئ XMLHttpRequest دو خاصیت[19] جهت دسترسی به پاسخ سرور دارد. خاصیت اول، responseText ، پاسخ را به عنوان رشته کارکتری در دسترس قرار می دهد و خاصیت دوم، responseXML ، پاسخ را به صورت شئ XML در دسترس قرار می دهد. بازگرداندن پاسخ به صورت متن ساده جهت کارهای ساده مناسب است مثل هنگامی که پاسخ سرور باید در پنجره پیام نمایش داده شود یا هنگامی که پاسخ سرور تنها شامل یک عبارت است که نشان دهنده موفقیت یا شکست در یک شرط می باشد.

مثال قبل در فصل ۲، به پاسخ سرور به صورت متن دسترسی داشت و آن را در پنجره پیام نشان داد.

## استفاده از خاصیت innerHTML جهت ساخت محتوای پویا[20]

دستیابی به پاسخ سرور به صورت متن ساده، انعطاف زیادی را ایجاد نمی کند. متن ساده فاقد ساختار است و پردازش و تجزیه اطلاعات موجود در آن توسط جاوااسکریپت مشکل است. همچنین استفاده از آن ایجاد محتوای پویا در صفحات وب را مشکل می سازد.

خاصیت responseText هنگامی که به همراه خاصیت innerHTML مولفه های[21] HTML به کار گرفته شود می تواند مفید واقع شود. خاصیت innerHTML، یک خاصیت

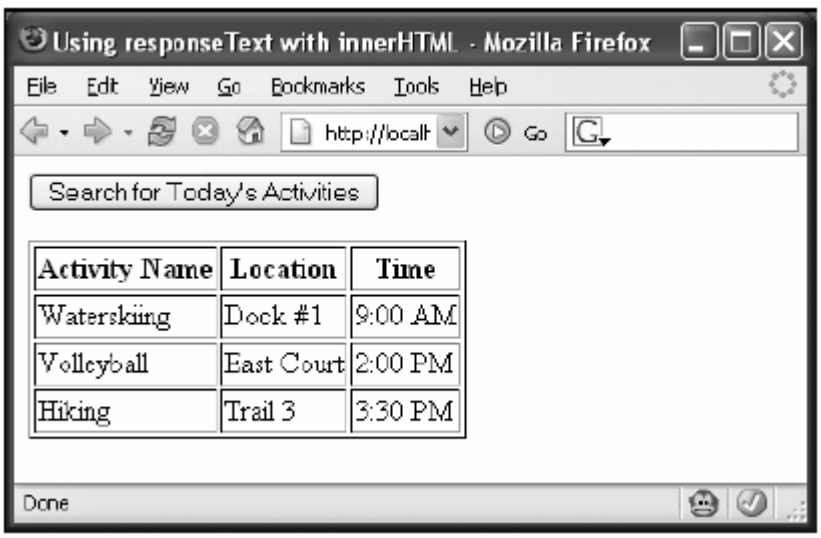
---

[19] - property

[20] - dynamic content

غیر استاندارد در مولفه های HTML است که اولین بار توسط اینترنت اکسپلورر پیاده سازی شد و بعدها توسط بسیاری از مرورگرهای دیگر نیز پیاده سازی شد.

با استفاده از responseText و innerHTML سرور می تواند محتوای html که مورد استفاده مرورگرهاست را تولید نماید. مثال زیر یک عمل جستجوی ساده با استفاده از شئ XMLHttpRequest و خاصیت responsetext آن و خاصیت innerhtml مولفه های HTML را نشان می دهد. فشردن دکمه جستجو عمل جستجو روی سرور را فراخوانی می کند. سرور با تولید جدول نتایج، پاسخ می دهد. مرورگر با تنظیم خاصیت innerhtml یک مولفه div در صفحه به مقدار خاصیت responsetext شئ XMLHttpRequest ، پاسخ سرور را نشان می دهد. شکل ۳-۱ پنجره مرورگر را بعد از فشردن دکمه جستجو نشان می دهد.( جدول نتایج به محتوای صفحه افزوده شده است.)




شکل ۳-۱ : مرورگر نتایج بازیابی شده را با استفاده از شئ XMLHttpRequest و خاصیت innerhtml نشان می دهد.

کد این مثال نیز به کد مثال قصل ۲ که نتیجه را در پنجره پیام نشان می داد، بسیار شبیه می باشد. مراحل به صورت زیر هستند:

element - [21]

کلیک دکمه جستجو تابع جاوااسکریپتی startRequest را در کد صفحه جاری فراخوانی می کند که این تابع نیز در ابتدا تابع createxmlhttprequest را جهت مقداردهی به نمونه ای از شئ XMLHttpRequest ، فراخوانی می کند.

تابع startrequest تابع handlestatechange را به عنوان تابع callback ( تابعی که هنگام رسیدن پاسخ سرور اجرا می شود ) قرار می دهد.

تابع startrequest با استفاده از متد open، روش ارسال ( get ) و آدرس مقصد و نحوه عملکرد ( ناهمگام ) شئ XMLHttpRequest را مشخص می سازد.

سپس درخواست با استفاده از متد send به سمت سرور در آدرس مشخص شده ارسال می شود.

هر زمان که حالت داخلی شئ XMLHttpRequest تغییر کند، تابع handleStateChange فراخوانی می شود. وقتی که خاصیت readyState مقدار ۴ داشته باشد و مقدار status آن برابر ۲۰۰ باشد، خاصیت innerHTML مولفه div در صفحه با استفاده از خاصیت responseText تنظیم می شود.

کد ۳-۱ فایل innerHTML.html را نشان می دهد و کد ۳-۲ فایل innerHTML.xml را که نشان دهنده نتایج تولید شده جستجو روی سرور می باشد، را نشان می دهد

کد ۳-۱ : innerHTML.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Using responseText with innerHTML</title>
<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest() {
   if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
   }
}
function startRequest() {
   createXMLHttpRequest();
   xmlHttp.onreadystatechange = handleStateChange;
   xmlHttp.open("GET", "innerHTML.xml", true);
   xmlHttp.send(null);
}
function handleStateChange() {
```

```
   if(xmlHttp.readyState == 4) {
      if(xmlHttp.status == 200) {
            document.getElementById("results").innerHTML =
            xmlHttp.responseText;
     }
   }
}
</script>
</head>
<body>
<form action="#">
   <input type="button" value="Search for Today's Activities"
      onclick="startRequest();"/>
</form>
<div id="results"></div>
</body>
</html>
```

کد ۳-۲: innerHTML.xml

```
<table border="1">
   <tbody>
      <tr>
         <th>Activity Name</th>
         <th>Location</th>
         <th>Time</th>
      </tr>
      <tr>
         <td>Waterskiing</td>
         <td>Dock #1</td>
         <td>9:00 AM</td>
      </tr>
      <tr>
         <td>Volleyball</td>
         <td>East Court</td>
         <td>2:00 PM</td>
      </tr>
      <tr>
         <td>Hiking</td>
         <td>Trail 3</td>
         <td>3:30 PM</td>
      </tr>
   </tbody>
</table>
```

**دیدید که در این مثال استفاده از responseText و innerHTML افزودن محتوای پویا به صفحات را بسیار آسان کرد. متاسفانه این روش اشکال های زیادی نیز دارد. مثلاً خاصیت innerHTML یک خاصیت استاندارد برای مولفه های HTML نیست و پشتیبانی آن توسط مرورگرها اختیاری است، هرچند که بسیاری از مرورگرهای امروزی از آن پشتیبانی می کنند. اینترنت اکسپلورر، مرورگری که پیشگام پیاده سازی این خاصیت است، بیشترین محدودیت ها را روی استفاده از آن قرار داده است. بسیاری از مرورگرهای امروزی این خاصیت را به**

صورت یک خاصیت خواندنی/ نوشتنی[۲۲] روی همه مولفه های HTML قرار داده اند. اما در جهت عکس این حرکت، اینترنت اکسپلورر این خاصیت را در بعضی از مولفه های HTML مثل جدول و سطرهای جدول به صورت فقط خواندنی پیاده سازی کرده است که گاهی فواید این خاصیت را محدود می سازد.

## پردازش پاسخ به صورت XML

دیدید که لازم نیست سرور حتماً پاسخ را به صورت XML بفرستد. ساختارهای داده ای پیچیده گزینه های خوبی جهت ارسال در قالب XML هستند. مرورگرهای مدرن امروزی به خوبی از راهبری[۲۳] در یک سند[۲۴] XML و تغییر ساختار محتوای سند XML پشتیبانی می کنند.

مرورگر چگونه پاسخ در قالب XML را به کار می برد؟ مرورگرهای امروزی با پاسخ XML به صورت سند XML مطابق با W3C DOM رفتار می کنند. مرورگرهای موافق DOM، با پیاده سازی API های مربوطه و رفتارهای تعیین شده، قابلیت انتقال اسکریپت ها بین مرورگرها را افزایش می دهند.

## جاوااسکریپت و W3C DOM

به راحتی ممکن است جاوااسکریپت و W3C DOM اشتباه گرفته شوند. DOM یک API برای HTML و XML است که یک ارائه ساختاری[۲۵] از این نمونه اسناد در دسترس قرار می دهد و چگونگی دسترسی به ساختار اسناد از طریق زبانهای اسکریپتی را تعریف می کند. جاوااسکریپت زبانی است که جهت دسترسی و دستکاری DOM استفاده می شود. بدون DOM، جاوااسکریپت هیچ فهمی از صفحات وب و مؤلفه های سازنده این صفحات نخواهد داشت. هر مؤلفه ای در یک سند، جزئی از DOM است که خاصیت ها و متدهای قابل دستیابی توسط جاوااسکریپت را تامین می کند.

DOM به هیچ زبانی وابسته نیست. معمولاً DOM توسط جاوااسکریپت به کارگرفته می شود اما این اجباری نیست. شما به لطف API پایدار DOM با استفاده از هر زبان اسکریپتی

---

[22] - read/write
[23] - navigate
[24] - document
[25] - structural representation

می توانید با آن کار کنید. جدول ۳-۱ لیستی از خاصیت های مفید مؤلفه های DOM و جدول ۳-۲ متدهای مفید آن را ارائه می کند.

جدول۳-۱ : خاصیت های مفید مؤلفه های DOM جهت پردازش اسناد XML

| Property Name | Description |
|---|---|
| childNodes | Returns an array of the current element's children |
| firstChild | Returns the first direct child of the current element |
| lastChild | Returns the last child of the current element |
| nextSibling | Returns the element immediately following the current element |
| nodeValue | Specifies the read/write property representing the element's value |
| parentNode | Returns the element's parent node |
| previousSibling | Returns the element immediately preceding the current element |

جدول ۳-۲ : متدهای مفید مؤلفه های DOM جهت پیمایش اسناد XML

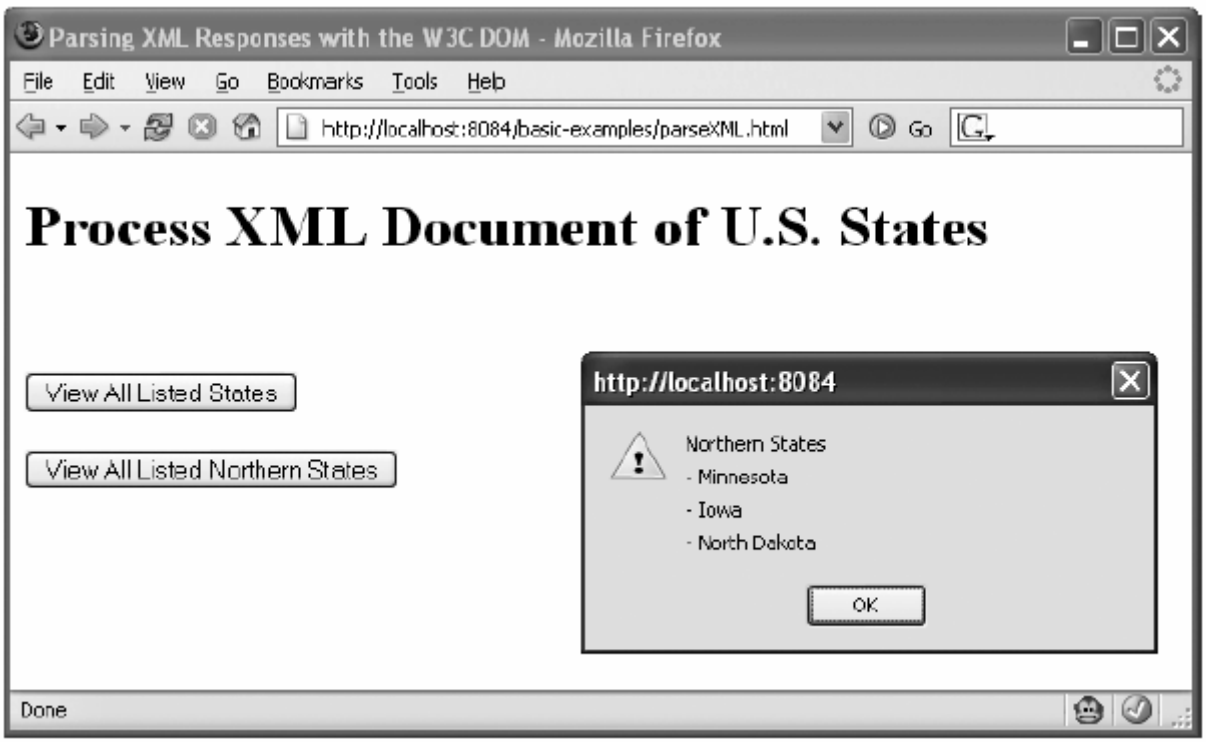
| Method Name | Description |
|---|---|
| getElementById(id) (document) | Retrieves the element in the document that has the specified unique ID attribute value |
| getElementsByTagName(name) | Returns an array of the current element's children that have the specified tag name |
| hasChildNodes() | Returns a Boolean indicating whether the element has any child elements |
| getAttribute(name) | Returns the value of the element's attribute specified by name |

به لطف W3C DOM شما می توانید قدرت و انعطاف XML را به عنوان ابزار ارتباطی بین مرورگر و سرور با نوشتن اسکریپتهای ساده و مستقل از مرورگر، تحت کنترل خود درآورید.
مثال زیر نشان می دهد که چگونه می توانید با استفاده از جاوااسکریپت یک سند XML را به سادگی بخوانید. کد ۳-۳ محتوای فایل XML که توسط سرور به مرورگر پاسخ داده می شود، را نشان می دهد که لیستی ساده از ایالتهای مختلف آمریکاست که بر اساس مناطق چهارگانه جغرافیایی دسته بندی شده اند.

کد ۳-۳ : لیست ایالتهای آمریکا که توسط سرور برگردانده شده است.

```
<?xml version="1.0" encoding="UTF-8"?>
<states>
	<north>
		<state>Minnesota</state>
		<state>Iowa</state>
		<state>North Dakota</state>
	</north>
	<south>
		<state>Texas</state>
		<state>Oklahoma</state>
		<state>Louisiana</state>
	</south>
	<east>
		<state>New York</state>
		<state>North Carolina</state>
		<state>Massachusetts</state>
	</east>
	<west>
		<state>California</state>
		<state>Oregon</state>
		<state>Nevada</state>
	</west>
</states>
```

در مرورگر، یک صفحه ساده با دو دکمه خواهیم دید. کلیک اولین دکمه، سند XML را از سرور بازیابی کرده و تمام ایالتها را در پنجره پیام نشان می دهد و دکمه دوم، سند XML را از سرور بازیابی و تنها ایالتهای شمالی را در پنجره پیام نشان می دهد. (شکل ۳-۲ را ببینید.)




شکل ۳-۲ : کلیک هر کدام از دکمه ها سند XML را از سرور بازیابی و نتایج مربوطه را در پنجره پیام نشان می دهد.

کد ۳-۴ محتویات فایل parseXML.html را نشان می دهد.

کد۳-۴ : parseXML.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Parsing XML Responses with the W3C DOM</title>

<script type="text/javascript">
var xmlHttp;
var requestType = "";

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}
function startRequest(requestedList) {
      requestType = requestedList;
      createXMLHttpRequest();
      xmlHttp.onreadystatechange = handleStateChange;
      xmlHttp.open("GET", "parseXML.xml", true);
      xmlHttp.send(null);
}

function handleStateChange() {
      if(xmlHttp.readyState == 4) {
            if(xmlHttp.status == 200) {
                  if(requestType == "north") {
                        listNorthStates();
                  }
                  else if(requestType == "all") {
                        listAllStates();
                  }
            }
      }
}

function listNorthStates() {
      var xmlDoc = xmlHttp.responseXML;
      var northNode =       xmlDoc.getElementsByTagName("north")[0];
      var out = "Northern States";
      var northStates = northNode.getElementsByTagName("state");
      outputList("Northern States", northStates);
}

function listAllStates() {
      var xmlDoc = xmlHttp.responseXML;
      var allStates = xmlDoc.getElementsByTagName("state");
      outputList("All States in Document", allStates);
}
```

```
function outputList(title, states) {
      var out = title;
      var currentState = null;
      for(var i = 0; i < states.length; i++) {
            currentState = states[i];
            out = out + "\n- " +
      currentState.childNodes[0].nodeValue;
      }
      alert(out);
}
</script>
</head>
<body>
<h1>Process XML Document of U.S. States</h1>
<br/><br/>
<form action="#">
<input type="button" value="View All Listed States"
      onclick="startRequest('all');"/>
<br/><br/>
<input type="button" value="View All Listed Northern States"
onclick="startRequest('north');"/>
</form>
</body>
</html>
```

کد اسکریپت جهت بازیابی سند XML از سرور و پردازش آن شبیه مثال قبل که مشاهده کردید می باشد ( پردازش پاسخ سرور به صورت متن ساده ). تفاوت کلیدی این دو مثال در توابع listAllState و listNorthState می باشد. مثال قبل پاسخ سرور را به صورت متن ساده با استفاده از خاصیت responseText شئ XMLHttpRequest بازیابی می کرد. اما دو تابع listAllState و listNorthState از خاصیت responseXML این شئ جهت بازیابی پاسخ سرور به صورت سند XML استفاده می کنند و به شما اجازه می دهند تا از متدهای W3C DOM جهت پیمایش[۲۶] سند XML استفاده کنید.

به تابع listAllState دقت کنید. اولین کاری که انجام می دهد، ایجاد یک متغیر محلی به نام xmlDoc و مقداردهی آن با استفاده از سند XML بازیابی شده توسط خاصیت responseXML ، می باشد. از متد getElementByTagName سند XML جهت بازیابی همه مؤلفه های سند که تگی به نام state دارند، استفاده می شود. این متد آرایه ای از همه مؤلفه های state برمی گرداند که توسط متغیری محلی به نام allState نگهداری می شوند.

[26] traverse -

بعد از بازیابی همه مؤلفه های state از سند XML، تابع listAllState جهت نمایش آنها در پنجره پیام، تابع outputList را فراخوانی می کند. تابع outpitList تمام اعضای آرایه state را مرور کرده و به ازای هر عضو از این آرایه نام مربوط به آن را به انتهای یک رشته کارکتری می افزاید. در پایان، این رشته کارکتری توسط پنجره پیام نمایش داده خواهد شد.

نکته ای که در اینجا باید به آن توجه شود این است که چگونه نام ایالات از مؤلفه های state بازیابی می شود. ممکن است انتظار داشته باشید که مؤلفه state متد یا خاصیتی جهت بازیابی متن خود داشته باشد، اما این درست نیست. متنی که نام ایالت را نشان می دهد، در حقیقت یک فرزند[۲۷] مؤلفه state در سند XML می باشد و برای به دست آوردن مقدار آن باید فرزند را بازیابی و سپس محتوای متنی آن فرزند را که نتیجه مورد نظر است، مورد استفاده قرار گیرد. تابع outputList این عمل را انجام می دهد. این تابع همه اعضای آرایه را بررسی می کند و هر بار عضو جاری را در متغیری به نام currentState نگهداری می کند. به دلیل اینکه متنی که نشان دهنده نام ایالت است همواره اولین فرزند مؤلفه می باشد، از خاصیت childNodes به صورت childNodes[0]، جهت بازیابی آن استفاده می شود سپس خاصیت nodeValue متنی را که نشان دهنده نام ایالت می باشد را بر می گرداند.

تابع listNorthStates بسیار شبیه تابع listAllState می باشد با این تفاوت که در این تابع تنها ایالتهای شمالی بازیابی خواهند شد، نه همه ایالتها. به این منظور، ابتدا با استفاده از متد getElementByTagName تمام مؤلفه های north از سند XML بازیابی می شوند. به یاد داشته باشید که خروجی این متد آرایه ای از مؤلفه هاست و از آنجایی که سند XML مربوطه تنها حاوی یک north می باشد از [0] در انتهای این متد استفاده می کنیم. اکنون که مؤلفه north به طور کامل بازیابی شده است، می توان با فراخوانی متد getElementByTagName تمام مؤلفه های state که فرزندان مؤلفه north هستند را به صورت آرایه به دست آورد. در پایان از متد outputList جهت نمایش محتوای متنی اعضای این آرایه در پنجره پیام استفاده می شود.

---

[27] - child

## ویرایش محتوا با استفاده از W3C DOM به صورت پویا

اکنون وب از رسانه ای جهت توزیع اسناد متنی، به بستری برای توسعه نرم افزار تبدیل شده است. کم کم شبکه های محلی که به منظور ارتباط کلاینت هایی که فقط برنامه های متنی را پردازش می کنند، به وجود آمده اند جای خود را به رسانه قوی وب خواهند داد و برنامه های کلاینت ها نیز همان مرورگرهای وب خواهند شد.

کاربران نهایی نرم افزارها هر روز تمایل بیشتری به استفاده از نرم افزارهای مبتنی بر وب نشان می دهند. طولی نخواهید کشید که آنها دیگر تحمل مشاهده بارگذاری های مکرر صفحات وب را بعد از هر ویرایش روی محتوای صفحات نخواهند داشت. آنها خواهان آن خواهند بود که نتیجه را همان لحظه ببینند، بدون انتظار جهت ارسال کامل صفحه وب به سرور و رسیدن نتیجه آن.

تا اینجا مشاهده کردید که چگونه می توان به سادگی پیام های XML را که توسط سرور ارسال شده اند مورد استفاده قرار داد. W3C DOM خاصیت ها ومتدهایی را در دسترس شما قرار داده است که به شما امکان می دهند ساختار XML را پیمایش و داده های مورد نیاز را استخراج نمایید.

مثال قبل عمل قابل توجهی را روی پاسخ سرور که به صورت XML فرستاده شده بود، انجام نمی داد. نمایش مقادیر داده ای یک قایل XML در پنجره پیام عملاً در دنیای واقعی وب ارزش پندانی ندارد. شما باید به دنبال آن باشید که صفحه ای از سایتتان را که کاربر با آن کار می کند به سادگی نیاز به بارگذاری کلی نداشته باشد. این ویژگی نه تنها به کاربر احساس خوشایندی می دهد بلکه چرخه های پردازنده سرور جهت ساخت کل یک صفحه از وب سایت را کم می کند و به طبع پهنای باند کمتری جهت این امور پیش پا افتاده به هدر خواهد رفت.

خوب بهترین راه حل این است که به جای اینکه کل یک صفحه را بارگذاری مجدد کنید، در حالی که می دانید بیشتر اطلاعات آن صفحه هر بار تغییر نخواهند کرد، آن است که آن قسمتی از صفحه که اطلاعات آن تغییر نموده را مجدداً بارگذاری کنید.

اما انجام این عمل با استفاده از قابلیتهای عادی مرورگرها کار دشواری است. مرورگر در اصل برنامه ای است که یک سری تگ HTML را ترجمه کرده و آنها را بر اساس قوانین مشخصی نمایش می دهد. در اصل، وب و مرورگرهای وب جهت نمایش اطلاعات پایا روی کار

آمده اند. ( اطلاعاتی که بدون درخواست یک صفحه کامل جدید از سرور، تغییر نخواهند کرد.)

مرورگرهای مدرن، به جز چند استثنا، محتوای صفحات وب را با استفاده از W3C DOM ارائه می کنند. این باعث می شود که صفحات وب توسط مرورگرهای مختلف یکسان دیده شوند و کدهای اسکریپتی که به منظور تغییر محتوای صفحات به کار رفته اند در مرورگرهای مختلف یکسان عمل کنند. با ادامه تکامل پیاده سازی های W3C DOM و جاوااسکریپت توسط مرورگرها، عملیات ایجاد صفحات کاملاً پویا که محتویات آنها بدون نیاز به بارگذاری مجدد کل صفحه تغییر می کنند، بسیار ساده و آسان خواهد شد. جدول ۳-۳ لیستی از خاصیتها و متدهای مفید DOM که برای ایجاد محتوای پویا به کار می روند، را نشان می دهد.

جدول ۳-۳ : خاصیت ها و متدهای مفید DOM جهت ایجاد محتوای پویا

| Property/Method | Description |
|---|---|
| `document.createElement(tagName)` | The `createElement` method on the document object creates the element specified by `tagName`. Providing the string `div` as the method parameter produces a `div` element. |
| `document.createTextNode(text)` | This document object's `createTextNode` method creates a node containing static text. |
| `<element>.appendChild(childNode)` | The `appendChild` method adds the specified node to the current element's list of child nodes. For example, you can add an `option` element as a child node of a `select` element. |
| `<element>.getAttribute(name)`<br>`<element>.setAttribute(name, value)` | These methods, respectively, get and set the value of the attribute `name` of the element. |
| `<element>.insertBefore(newNode, targetNode)` | This inserts the node `newNode` before the element `targetNode` as a child of the current element. |
| `<element>.removeAttribute(name)` | This removes the attribute `name` from the element. |
| `<element>.removeChild(childNode)` | This removes the element `childNode` from the element. |
| `<element>.replaceChild(newNode, oldNode)` | This method replaces the node `oldNode` with the node `newNode`. |
| `<element>.hasChildnodes()` | This method returns a Boolean indicating whether the element has any child elements. |

## سخنی کوتاه در مورد ناسازگاری مرورگرها

با وجود اینکه پیاده سازی W3C DOM و جاوااسکریپت در مرورگرهای مدرن دائماً پیشرفت می کند، برخی تغییرات ناگهانی و ناسازگاری ها همچنان هنگام کار با DOM و جاوااسکریپت دردسرساز می شوند. اینترنت اکسپلورر بیشترین محدودیت ها را در پیاده سازی DOM و جاوااسکریپت داراست. سال ۲۰۰۰ اینترنت اکسپلورر ۹۵٪ از بازار مرورگرها در اختیار داشت، در چنین شرایطی که هیچ رقابت نزدیکی برای آن وجود ندارد، مایکروسافت به طور کامل همه استانداردهای وب را پیاده سازی نمی کند.

شما می توانید بدون در نظرگرفتن این ناسازگاری ها به کار خود بپردازید، هرچند که اینگونه کدنویسی دچار آشفتگی می شود و اصلاً استاندارد نیست. به عنوان مثال، تگ <tr> که مستقیماً توسط دستور <appendChild> به تگ <table> اضافه می شود، در مرورگر اینترنت اکسپلورر نشان داده نخواهد شد، در حالی که در بقیه مرورگرها بدون نقص نشان داده می شود. در این زمینه روشی که همیشه و در هر مرورگری کار کند، افزودن تگ <tr> به تگ <tbody> است.

اینترنت اکسپلورر همچنین با متد setAttribute مشکل دارد. اینترنت اکسپلورر صفات یک کلاس را با استفاده از این دستور به درستی تنظیم نخواهد کرد. راه عملی و مستقل از مرورگر در این زمینه استفاده از دستور به صورت های زیر است:

```
setAttribute("class","newClassName") یا
setAttribute("className","newClassName")
```

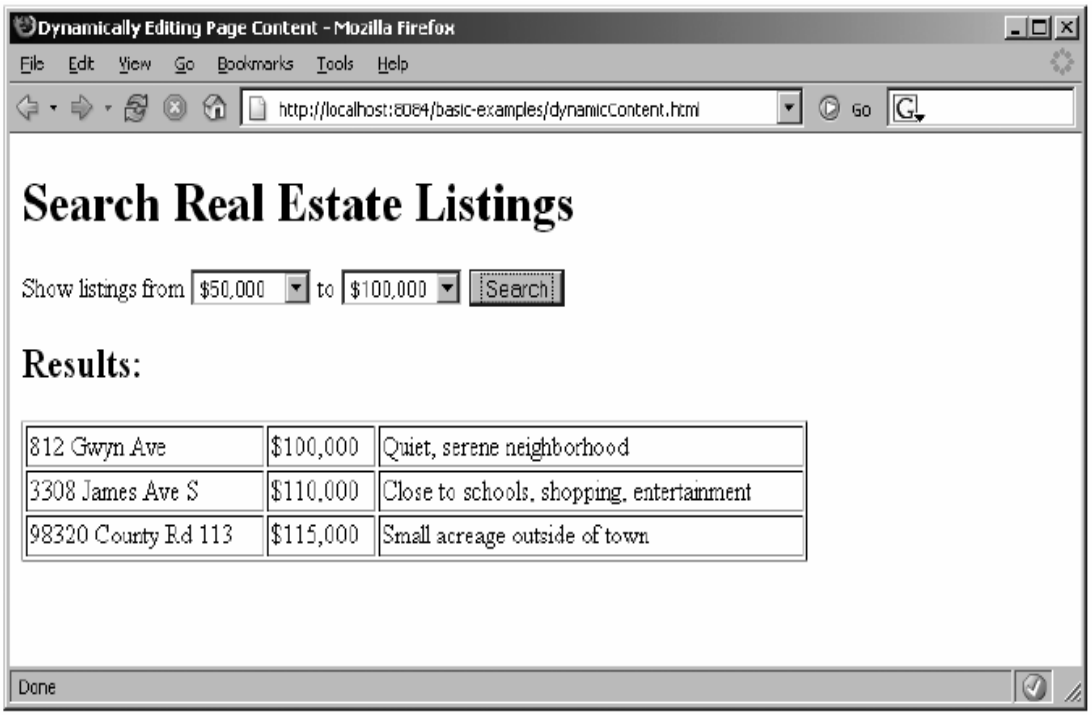
همچنین شما نمی توانید صفت style را با استفاده از setAttribute در اینترنت اکسپلورر تنظیم کنید. به جای استفاده از <element>.setAttribute("style, "font-weight:bold;") از تکنیک همیشه سازگار زیر استفاده کنید:

```
<element>.style.cssText = "font-weight:bold;"
```

مثال هایی که در ادامه می بینید سعی بر استفاده از استانداردهای W3C DOM به نحو احسن دارند اما هرگاه که با مشکل ناسازگاری در مرورگرها مواجه شده اند، از برخی استانداردهای DOM و جاوااسکریپت صرف نظر شده است.

مثال زیر نشان می دهد که چونه می توانید با استفاده از W3C DOM و جاوااسکریپت به صورت پویا محتوای صفحات وب را بسازید یا تغییر دهید. مثال شامل یک جستجو روی لیستی از املاک است. فشردن دکمه جستجو در صفحه، نتیجه را با استفاده از از شئ

XMLHttpRequest در قالب XML برمی گرداند. پاسخ در قالب XML است توسط جاوااسکریپت پردازش می شود تا جدولی که شامل لیستی از نتایج جستجو است تولید و در صفحه نمایش داده شود. ( شکل ۳-۳ را ببینید.)




شکل ۳-۳: نتیجه جستجو که به صورت پویا توسط متدهای DOM و جاوااسکریپت تولید شده.

فایل XML که توسط سرور به عنوان نتیجه بازگردانده می شود بسیار ساده است. گره[۲۸] اصلی ریشه، properties، همه مؤلفه های property، که به عنوان نتیجه باید برگردانده شوند را شامل می شود. هر مؤلفه property سه مؤلفه فرزند دارد: address, price, comments .

کد ۳-۵ : dynamicContent.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
	<property>
		<address>812 Gwyn Ave</address>
		<price>$100,000</price>
		<comments>Quiet, serene neighborhood</comments>
	</property>
	<property>
		<address>3308 James Ave S</address>
```

[28] node -

```
            <price>$110,000</price>
            <comments>Close to schools, shopping,
      entertainment</comments>
      </property>
      <property>
            <address>98320 County Rd 113</address>
            <price>$115,000</price>
            <comments>Small acreage outside of town</comments>
      </property>
</properties>
```

کد جاوااسکریپتی که به منظور ارسال درخواست به سرور و دریافت پاسخ استفاده شده است مانند مثالهای قبل است. تفاوت این مثال در تابع handleStateChange است. فرض کنید درخواست با موفقیت به اتمام رسیده است. اولین عملی که اتفاق می افتد این است که محتوایی که نتیجه جستجوهای قبلی است توسط فراخوانی تابع clearPreviousResult حذف شود. این تابع دو عمل انجام می دهد: متن سرآیند "Result" را که در بالای جدول نمایان می شود را حذف می کند و همه سطرهای جدول نمایش نتیجه را پاک می کند. برای حذف سرآیند ابتدا تگ span توسط متد hasChildNodes چک می شود که آیا فرزندی دارد یا خیر(مقدار متنی یک تگ به صورت فرزند آن تگ ارزیابی می شود). مشخص است اگر این متد مقدار true برگرداند، سرآیند حاوی متن است و در اینصورت تنها فرزند این تگ (span) که همان متن تگ است، حذف می شود. عمل دوم تابع clearPreviousResult حذف سطرهای موجود جدول نمایش است که نتیجه جستجوی قبلی را نشان می دهند. تمام سطرهای نتیجه، فرزندان گره tbody هستند، پس با نگهداری این گره در یک متغیر، با استفاده از متد document.getElementById، کار را شروع می کند. سپس به ازای تمام فرزندان گره tbody عملی تکراری را انجام می دهد می دانیم که فرزندان این گره مؤلفه های tr (ردیف های جدول) هستند. در هر بار تکرار اولین فرزند گره در مجموعه childNodes از بدنه جدول حذف می شود. این تکرار هنگامی که دیگر سطری در بدنه جدول باقی نمانده باشد، تمام می شود.

جدول نتیجه جستجو توسط تابع parseResults ساخته می شود. این تابع با تعریف یک متغیر محلی به نام results که محتوای خاصیت responseXML شئ XMLHttpRequest را در خود نگه خواهد داشت، شروع می شود. با استفاده از متد getElementByTagName همه مؤلفه های prperty در قالب یک آرایه از سند XML نتیجه، در متغیری محلی به نام properties ذخیره خواهند شد. هنگامی که آرایه ای از مؤلفه های property دارید

می توانید با تکرار روی هر عضو آن آرایه، مؤلفه های address, price, comments هر یک اعضا را به دست آورید.

```
var properties = results.getElementsByTagName("property");
for(var i = 0; i < properties.length; i++) {
      property = properties[i];
      address=property.getElementsByTagName("address")[0]
            .firstChild.nodeValue;
      price=property.getElementsByTagName("price")[0].
            firstChild.nodeValue;
      comments=property.getElementsByTagName("comments")
            [0].firstChild.nodeValue;
      addTableRow(address, price, comments);
}
```

بیایید دقیقتر به این تکرار که در اصل قسمت اصلی تابع parseResults است، نگاه کنیم. در حلقه for اولین عملی که اتفاق می افتد این است که عضوی از آرایه را در متغیری محلی به نام property نگهداری می شود. سپس به ازای هر کدام از فرزندان این مؤلفه ( address, price, comments) مقدار گره مربوطه توسط متغیرهایی بازیابی می شوند. مؤلفه address به عنوان فرزندی از مؤلفه property را بررسی می کنیم. در ابتدا با استفاده از متد getElementByTagName مربوط به مؤلفه proprty، آن را به دست می آوریم. متد اخیر نتیجه را به صورت آرایه برمی گرداند اما می دانیم که فقط یک مؤلفه address وجود دارد که با نماد [0] به آن دسترسی خواهیم داشت. اکنون که اشاره گری به مؤلفه address در اختیار است و می خواهیم محتوای متنی آن را به دست آوریم با استفاده از متد firstChild به آن دسترسی خواهیم داشت. تاکنون به گرهی رسیده ایم که متن مورد نظر(address) را شامل می شود پس با استفاده از خاصیت nodeValue آن گره، به محتوای متنی موردنظر خواهیم رسید. با استفاده از همین روند مقادیر متنی مؤلفه های price و comments را به دست آورده و در متغیرهای محلی مربوطه نگهداری خواهیم کرد. مقادیر به دست آمده به تابعی کمکی به نام addTableRow فرستاده می شوند. این تابع با استفاده از از این مقادیر، یک سطر جدید به جدول نتایج می افزاید.

تابع addTableRow با استفاده از متدهای W3C DOM و جاوااسکریپت یک سطر جدول می سازد. در این تابع شئ row با استفاده از متد document.createElement ساخته می شود. بعد از ساخت شئ row با استفاده از تابع کمکی به نام createCellWithText برای هر یک از مقادیر address, price, comments یک ستون ایجاد می شود. تابع

createCellWithText یک شئ cell می سازد که محتوای آن، همان متن مشخص شده در ورودی اش است، سپس این شئ cell را برمی گرداند. تابع createCellWithText با ایجاد یک مؤلفه td توسط متدdocument.createElement شروع می شود. سپس یک گره متنی که حاوی متن مورد نظر است با استفاده از متد document.createTextNode ساخته می شود و به مؤلفه td اضافه می شود. سپس تابع مؤلفه td ساخته شده را به عنوان نتیجه به تابع احضارکننده اش برمی گرداند. تابع addTableRow فراخوانی تابع createCellWithText را برای هر یک از مؤلفه های address, price, comments و مقادیر مربوط به آنها تکرار می کند و در هر تکرار یک ستون ( مؤلفه td ) به سطر در حال ساخت ( مؤلفه tr ) می افزاید. هنگامی که هر سه ستون به سطر اضافه شدند، سطر مربوطه به مؤلفه tbody جدول افزوده می شود.

این مثال با موفقیت سند XML که توسط سرور به عنوان پاسخ برگردانده شده بود را می خواند و به صورت پویا جدول نتایج را ساخته و نمایش می دهد. کد ۳-۶ کد جاوااسکریپت و XHTML این مثال را نشان می دهد.

کد۳-۶ : dynamicContent.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Dynamically Editing Page Content</title>
<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
         xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function doSearch() {
      createXMLHttpRequest();
      xmlHttp.onreadystatechange = handleStateChange;
      xmlHttp.open("GET", "dynamicContent.xml", true);
      xmlHttp.send(null);
}

function handleStateChange() {
      if(xmlHttp.readyState == 4) {
            if(xmlHttp.status == 200) {
                  clearPreviousResults();
```

```
                parseResults();
            }
        }
}

function clearPreviousResults() {
        var header = document.getElementById("header");
        if(header.hasChildNodes()) {
            header.removeChild(header.childNodes[0]);
        }
        var tableBody = document.getElementById("resultsBody");
        while(tableBody.childNodes.length > 0) {
            tableBody.removeChild(tableBody.childNodes[0]);
        }
}

function parseResults() {
        var results = xmlHttp.responseXML;
        var property = null;
        var address = "";
        var price = "";
        var comments = "";
        var properties = results.getElementsByTagName("property");
        for(var i = 0; i < properties.length; i++) {
            property = properties[i];
            address=
            property.getElementsByTagName("address")[0].
                firstChild.nodeValue;
            price = property.getElementsByTagName("price")[0].
                firstChild.nodeValue;
            comments = property.getElementsByTagName("comments")[0].
                firstChild.nodeValue;

            addTableRow(address, price, comments);
        }
        var header = document.createElement("h2");
        var headerText =
            document.createTextNode("Results:");
        header.appendChild(headerText);
        document.getElementById("header").appendChild(header
            );
        document.getElementById("resultsTable").setAttribute
            ("border", "1");
}

function addTableRow(address, price, comments) {
        var row = document.createElement("tr");
        var cell = createCellWithText(address);
        row.appendChild(cell);
        cell = createCellWithText(price);
        row.appendChild(cell);
        cell = createCellWithText(comments);
        row.appendChild(cell);
        document.getElementById("resultsBody").appendChild(row);
}

function createCellWithText(text) {
        var cell = document.createElement("td");
        var textNode = document.createTextNode(text);
        cell.appendChild(textNode);
        return cell;
```

```
}
</script>
</head>
<body>
<h1>Search Real Estate Listings</h1>
<form action="#">
      Show listings from
<select>
      <option value="50000">$50,000</option>
      <option value="100000">$100,000</option>
      <option value="150000">$150,000</option>
</select>
to
<select>
      <option value="100000">$100,000</option>
      <option value="150000">$150,000</option>
      <option value="200000">$200,000</option>
</select>
<input type="button" value="Search" onclick="doSearch();"/>
</form>
      <span id="header">
</span>
<table id="resultsTable" width="75%" border="0">
      <tbody id="resultsBody">
      </tbody>
</table>
</body>
</html>
```

## ارسال پارامتر به همراه درخواست

خوب تاکنون دیدید که چگونه با استفاده از تکنیکهای آژاکس یک درخواست را به سرور بفرستید و با روشهای مختلف پردازش پاسخ سرور توسط کلاینت آشنا شدید. تنها کاستی مثالهای قبل این است که به همراه درخواست هیچ داده ای به سرور فرستاده نمی شد. در بیشتر موارد کاربردی، ارسال درخواست بدون پارامتر به سرور چندان استفاده ای ندارد. بدون پارامترهای درخواست، سرور هیچ داده ای جهت تولید پاسخ ویژه برای کلاینت درخواست دهنده، ندارد و در اصل سرور پاسخ های پیش فرض و یکسانی را به همه کلاینت ها می فرستد.

مشاهده قدرت واقعی تکنیکهای آژاکس مستلزم این است که داده های وابسته به شرایط را به سرور بفرستید. فرم ورود داده ای را تصور کنید که مشخصات مختلفی را ساخت ایمیل جدید از کاربر می خواهد. شما می توانید از تکنیک های آژاکس استفاده کنید و هنگامی که کاربر کد شهر خود را وارد کرد، نام آن شهر را نمایش دهید. خوب البته برای جستجو در کد شهرها، سرور کد شهری را کاربر وارد کرده است را نیاز دارد. شما باید به طریقی کدی را که

ماربر وارد کرده به سرور بفرستید. خوشبختانه، شئ XMLHttpRequest خیلی شبیه به تکنیک های قدیمی HTTP که در آنها از Get و Post استفاده می شود، عمل می کند.

متد Get مقادیر را به صورت جفت های name/value به عنوان قسمتی از آدرس[۲۹] درخواست، به سرور ارسال می کند. انتهای آدرس مقصد با علامت ؟ مشخص می شود و بعد از آن جفت های name/value قرار می گیرند. جفت های name/value به صورت name=value قرار می گیرند و جفت ها با علامت & از یکدیگر جدا می شوند.

در زیر مثالی از درخواست به روش Get را می بینید. این درخواست دو پارامتر به نام های firstName و middleName به برنامه ای به نام yourApp روی سروری به نام localhost می فرستد. دقت کنید که آدرس مقصد و پارامترها توسط علامت ؟ و خود پارامترها نیز توسط علامت & جدا می شوند.

```
http://localhost/yourApp?firstName=Adam&middleName=Christopher
```

سرور می داند که چگونه پارامترها را از آدرس URL به دست آورد. بیشتر زبان های برنامه نویسی سمت سرور، روشها و توابع ویژه ای را که اجازه می دهند به این نمونه پارامترها دسترسی داشته باشید، در اختیار برنامه نویسان قرار داده ند.

روش ارسال پارامتر توسط متد post نیز بسیار شبیه روش استفاده از متد get است. متد post نیز پارامترها را به صورت جفت های name/value در فرمت name=value به کار می گیرد که هر جفت توسط علامت & از یکدیگر جدا شده اند. تفاوت اصلی بین این دو روش این است که متد post پارامترها را به عنوان قسمتی از بدنه درخواست به سرور ارسال می کند در حالیکه متد get آنها را به آدرس مقصد می افزاید و چیزی به بدنه درخواست نمی افزاید.

از لحاظ تکنیکی، خصوصیات کاربردی HTML توصیه می کند که هنگامی که پردازش داد های ارسالی تغییری روی حالت مدل داده ای صفحه ایجاد نمی کند، از متد get استفاده شود به این معنی که هنگام بازیابی داده و تطلاعات از سرور بهتر است از متد get استفاده شود. متد post هنگامی توصیه می شود که حالت مدل داده ای صفحه به کل تغییر می کند مثل ذخیره یا ثبت داده ها.

هر روش فواید خاص خود را دارد. مثلاً به دلیل اینکه پارامترهای متد get در آدرس URL مقصد قرار می گیرند، کاربر می تواند توسط مرورگر خود این آدرس را ذخیره[۳۰] کرده و به

---

[29] - URL

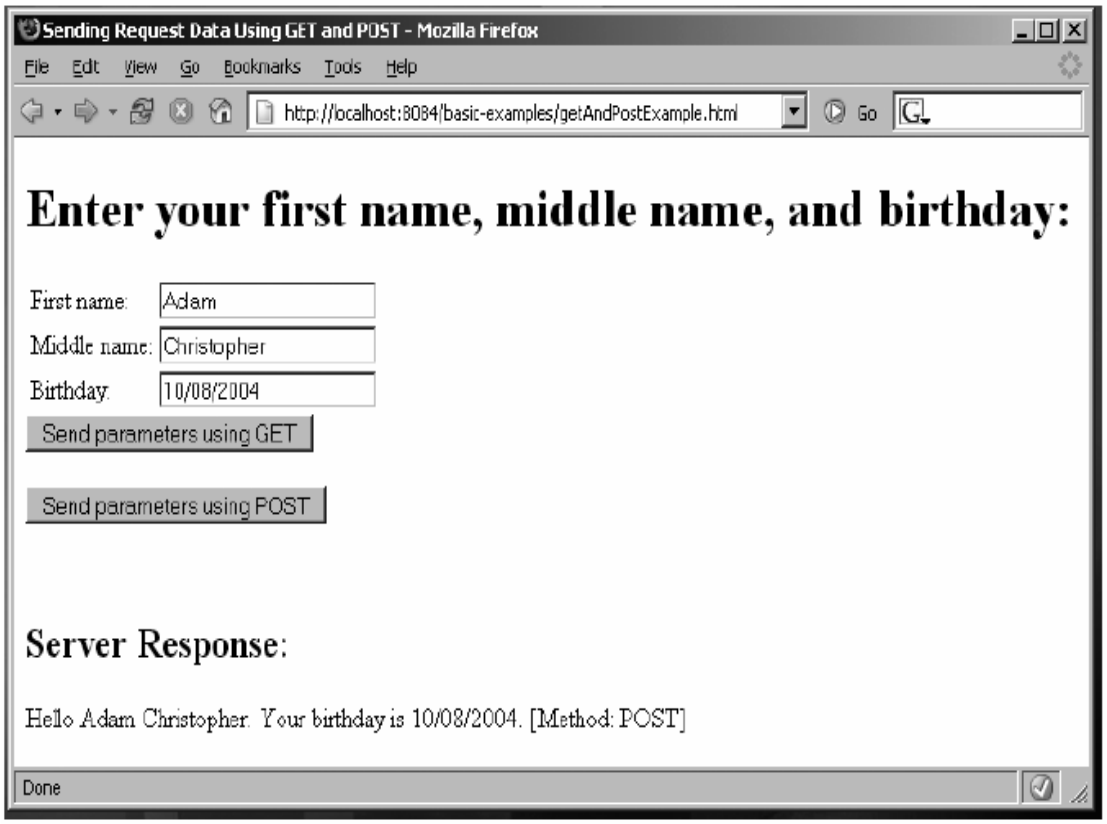
[30] - bookmark

سادگی این درخواست را تکرار کند. هرچند که هنگام استفاده از درخواست ها به صورت ناهنگام این قابلیت استفاده خاصی ندارد. متد post قابلیت انعطاف بیشتری روی مقدار داده ای که باید به سرور ارسال شود، دارد. مقدار داده ای که توسط کتد get می تواند به سرور ارسال شود بسته به مرورگرها، مقدار محدودی است که این محدودیت بین مرورگرهای مختلف متغیر است، در صورتی که متد post این محدودیت را ندارد و هر مقدار داده ای را می تواند به سرور ارسال کند.

مؤلفه form در HTML اجازه می دهد با استفاده از تنظیم مقدار خاصیت method خود، روش مطلوب خود را از بین get با post انتخاب کنید. مؤلفه form به صورت اتوماتیک مقادیر مؤلفه های ورودی را بر اساس روشی که در خاصیت method خود دارد هنگام submit شدن فرم در آدرس مقصد قرار داده و ارسال می کند. اما شئ XMLHttpRequest رفتار اتوماتیکی به این صورت ندارد. در عوض برنامه نویس مسئولیت ساخت رشته درخواست[۳۱] که حاوی داده های ارسالی است، را به عهده دارد. تکنیک ساخت رشته درخواست بدون توجه به نوع روش استفاده شده ( post یا get ) همواره ثابت و یکسان است. تنها تفاوت این است که هنگام ارسال درخواست به روش get مقدار رشته درخواست به انتهای آدرس مقصد باید اضافه شود، در حالیکه هنگام استفاده از روش post این مقدار هنگام فراخوانی متد send شئ XMLHttpRequest به عنوان قسمتی از بدنه درخواست به سرور ارسال می شود و نیازی به افزودن آن به آدرس مقصد نیست.

شکل ۳-۴ صفحه نمونه ای را نشان می دهد که چگونه پارامترهای درخواست به سرور ارسال می شوند. این صفحه ای ساده است که مقادیر اطلاعات شخصی را از کاربر می گیرد. فرم نشان داده شده، دو دکمه ئارد . هر دکمه داده های مربوط به نام و نام خانوادگی و تاریخ تولد را به سرور می فرستد با این تفاوت که یکی به روش post و دیگری به روش get این عمل را انجام می دهد. پاسخ سرور عبارت است از نمایش مقادیر وارد شده توسط کاربر. این چرخه با نمایش پاسخ ورودی توسط مرورگر پایان می یابد. (شکل ۳-۴ را ببینید.)

[31] - query string




**شکل۳-۴ : مرمرگر داده های ورودی را با استفاده از روشهای post و get به سرور می فرستد و سرور این مقادیر را جهت نمایش به مرورگر برمی گرداند.**

**کد ۳-۷ فایل getAndPostExample.html را نشان می دهد و کد ۳-۸ کد جاواسرولت سمت سرور است که مقادیر دریافت شده را در قالب یک عبارت جهت نمایش به مرورگر برمی گرداند.**

**کد۳-۷ :** getAndPostExample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Sending Request Data Using GET and POST</title>
<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
       }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}
```

```
function createQueryString() {
	var firstName = document.getElementById("firstName").value;
	var middleName = document.getElementById("middleName").value;
	var birthday = document.getElementById("birthday").value;
	var queryString = "firstName=" + firstName + "&middleName=" +
middleName
		+ "&birthday=" + birthday;
	return queryString;
}

function doRequestUsingGET() {
	createXMLHttpRequest();
	var queryString = "GetAndPostExample?";
	queryString = queryString + createQueryString()
		+ "&timeStamp=" + new Date().getTime();
	xmlHttp.onreadystatechange = handleStateChange;
	xmlHttp.open("GET", queryString, true);
	xmlHttp.send(null);
}

function doRequestUsingPOST() {
	createXMLHttpRequest();
	var url = "GetAndPostExample?timeStamp=" + new
Date().getTime();
	var queryString = createQueryString();
	xmlHttp.open("POST", url, true);
	xmlHttp.onreadystatechange = handleStateChange;
	xmlHttp.setRequestHeader("Content-Type",
		"application/x-www-form-urlencoded;");
	xmlHttp.send(queryString);
}

function handleStateChange() {
	if(xmlHttp.readyState == 4) {
		if(xmlHttp.status == 200) {
			parseResults();
		}
	}
}

function parseResults() {
	var responseDiv = document.getElementById("serverResponse");
	if(responseDiv.hasChildNodes()) {
		responseDiv.removeChild(responseDiv.childNodes[0]);
	}
	var responseText =
document.createTextNode(xmlHttp.responseText);
	responseDiv.appendChild(responseText);
}

</script>

</head>
<body>
<h1>Enter your first name, middle name, and birthday:</h1>
<table>
	<tbody>
		<tr>
			<td>First name:</td>
			<td><input type="text" id="firstName"/>
```

```
            </tr>
            <tr>
                  <td>Middle name:</td>
                  <td><input type="text" id="middleName"/>
            </tr>
            <tr>
                  <td>Birthday:</td>
                  <td><input type="text" id="birthday"/>
            </tr>
      </tbody>
</table>
<form action="#">
      <input type="button" value="Send parameters using GET"
            onclick="doRequestUsingGET();"/>
<br/><br/>
      <input type="button" value="Send parameters using POST"
            onclick="doRequestUsingPOST();"/>
</form>
<br/>
<h2>Server Response:</h2>
<div id="serverResponse"></div>
</body>
</html>
```

**کد ۳-۸ : برگرداندن مقادیر نام و نام خانوادگی و تاریخ تولد به مرورگر**

```
package ajaxbook.chap3;

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetAndPostExample extends HttpServlet {
      protected void processRequest(HttpServletRequest request,
            HttpServletResponse response, String method)
      throws ServletException, IOException {

            //Set content type of the response to text/xml
            response.setContentType("text/xml");

            //Get the user's input
            String firstName = request.getParameter("firstName");
            String middleName = request.getParameter("middleName");
            String birthday = request.getParameter("birthday");

            //Create the response text
            String responseText = "Hello " + firstName + " " +
            middleName+ ". Your birthday is " + birthday + "."
                  + " [Method: " + method + "]";

            //Write the response back to the browser
            PrintWriter out = response.getWriter();
            out.println(responseText);

            //Close the writer
            out.close();
      }

      protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
```

```
        throws ServletException, IOException {
            //Process the request in method processRequest
            processRequest(request, response, "GET");
        }

        protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
            //Process the request in method processRequest
            processRequest(request, response, "POST");
        }
}
```

اجازه دهید ابتدا کد سمت سرور را بررسی کنیم. این مثال از جاواسرولت برای پردازش درخواست کاربر استفاده نموده است. بدیهی است که می توان از زبان های دیگر مثل PHP,CGI,.NET استفاده کرد. این کد جاوا بایستی متدهای doGet و doPost را به منظور پاسخ دهی به دو نمونه درخواست رسیده، پیاده سازی نماید. در این مثال هر دو متد، متد یکسانی به نام processRequest را به منظور پردازش درخواست استفاده می نمایند.

متد processRequest با تنظیم نوع محتوای پاسخ به نوع text/xml شروع می شود، هرچند که در این مثال از XML استفاده نشده است. سه فیلد ورودی از شئ request با استفاده از متد getParameter بازیابی می شوند. یک جمله ساده با استفاده از مقتدیر این سه فیلد و نوع روش ارسال درخواست توسط کاربر، ساخته می شود. این جمله توسط مکانیزم مشخصی توسط سرور در مرورگر نوشته می شود.

کد جاوااسکریپت سمت کلاینت بسیار شبیه مثال های قبلی است، فقط کمی پیچیدگی جدید به آن افزوده شده است که توضیح داده می شود. تابعی به نام createQueryString مسئول ساخت رشته درخواست با استفاده از مقادیر وارد شده توسط کاربر، می باشد. این تابع مقادیر firstName, middleName, birthday را به صورت جفت های name/value به یکدیگیر الحاق نموده و به عنوان خروجی خود برمی گرداند. هر جفت با علامت & از یکدیگر جدا می شوند. خروجی این تابع توسط توابع مربوط به روشهای post و get استفاده خواهد شد.

فشردن دکمه send Request Using Get تابع doRequestUsingGet را فراخوانی می نماید. این تابع نیز مانند توابع دیگری که در مثال های قبل دیدید کارش را با ساخت نمونه ای از شئ XMLHttpRequest آغاز می کند، سپس مقدار رشته درخواست که حاوی مقادیر وارد شده توسط کاربر است، ساخته می شود.

آدرس مقصد در این مثال کدی به نام getAndPostExample می باشد. مقدار رشته درخواست با الحاق نتیجه تابع createQueryString به آدرس مقصد که توسط علامت ؟ از یکدیگر جدا شده اند، کامل خواهد شد.

کد جاوااسکریپت مانند مثال های قبل ادامه داده می شود. مقدار خاصیت onReadyStateChange به نام تابع handleStateChange تنظیم می شود. متد Open مشخص می کند این درخواست از نوع get می باشد و آدرس مقصد که در این مثال حاوی پارامترهایی می باشد را تنظیم می نماید. متد send درخواست کامل شده را به سرور می فرستد و تابع handleStateChange پاسخ سرور در هر لحظه را چک و پردازش می کند.

تابع handleStateChange در صورتی که درخواست با موفقیت توسط سرور پاسخ داده شود، تابع parseResult را فراخوانی می کند. تابع اخیر یک مؤلفه div به عنوان خروجی برمی گرداند که حاوی پاسخ سرور است و این پاسخ را در متغیر محلی به نام responsediv ذخیره می کند. تمام پاسخ های قبلی سرور در ابتدا توسط متد removeChild از responseDiv حذف می شوند. سرانجام، یک گره متنی جدید که حاوی پاسخ سرور است ساخته شده و به مؤلفه responseDiv افزوده می شود.

تکنیک استفاده از روش post شبیه روش get است فقط در چگونگی ارسال پارامترها به سرور با آن تفاوت دارد. قبلاً گفتیم که روش post، مقدار رشته درخواست را به عنوان قسمتی از بدنه درخواست ارسال می کند و آن را به انتهای URL مقصد نمی افزاید.

فشردن دکمه SendParametersUsingPost تابع doRequestUsingPost را فراخوانی می کند. این تابع همانند تابع doRequestUsingGet با ساخت نمونه ای از شئ XMLHttpRequest آغاز می شود. تابع سپس مقدار رشته درخواست را که حاوی پارامترهای ورودی است را می سازد. دقت کنید که در این روش مقدار رشته درخواست به انتهای آدرس مقصد افزوده نمی شود. سپس متد open شئ XMLHttpRequest فراخوانی می شود و روش ارسال درخواست، post مشخص شده و آدرس مقصد بدون الحاق چیزی به آن استفاده می شود. مقدار خاصیت onReadyState به نام تابع handleStateChange تنظیم می شود تا پردازش پاسخ سرور همانند روش get انجام شود. به منظور اینکه اطمینان حاصل شود که سرور می داند پارامترهای درخواست را می تواند از بدنه درخواست پیدا کند، متد setRequestHeader فراخوانی می شود و مقدار خاصیت Content-Type به

application/x-www-from-urlencoded تنظیم می شود. سرانجام، متد send که مقدار رشته درخواست را به عنوان پارامتر ورودی پذیرفته، فراخوانی می شود.

خروجی حاصل از فشردن هر دو دکمه یکسان است. چاپ رشته ای کارکتری روی صفحه که شامل فیلدهای نام و نام خانوادگی و تاریخ تولد و روش ارسال درخواست توسط کاربر، نتیجه این مثال است.

**چرا فیلد زمان[۳۲] به انتهای آدرس مقصد اضافه می شود؟**

در بعضی شرایط، بعضی از مرورگرها، نتایج چندین درخواست شئ XMLHttpRequest را به یک آدرس می فرستند. این موضوع می تواند باعث ایجاد نتایج نامطلوب، برای درخواستهای مختلف شود. افزودن فیلد زمان به انتهای آدرس مقصد، این آدرس را منحصر به فرد می سازد و از این خطا توسط مرورگرها، جلوگیری می کند.

[32] - timestamp

## ارسال پارامترهای درخواست به صورت XML

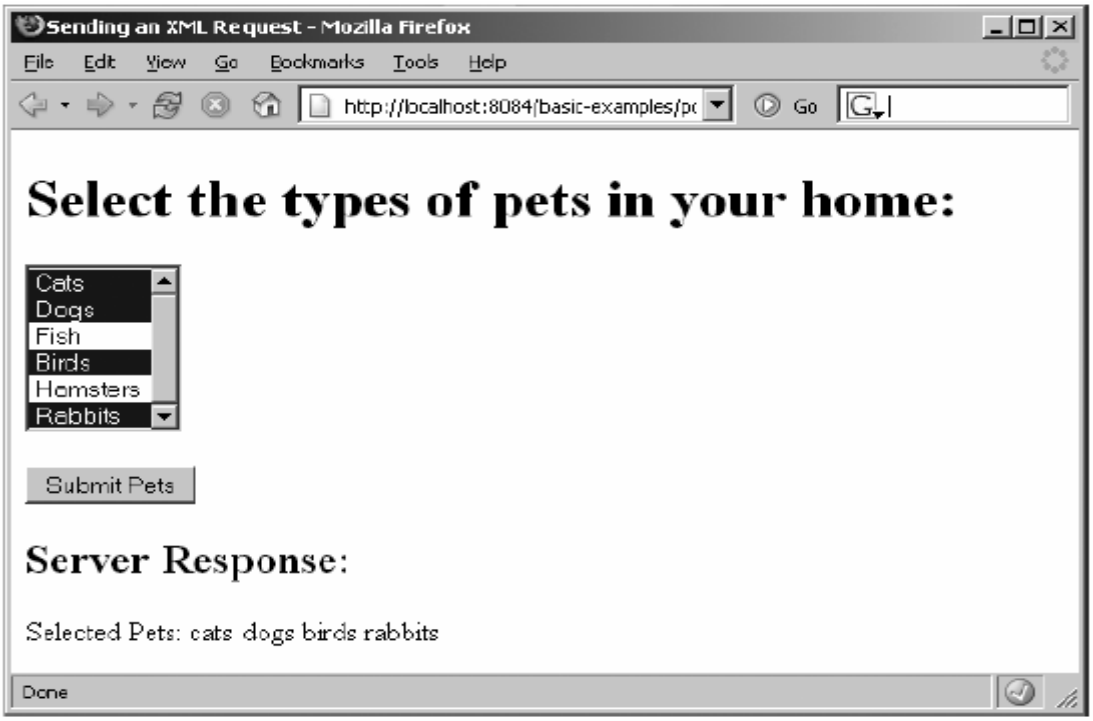
سازگاری جاوااسکریپت در مرورگرهای امروزی روزهای روشنی را در پیش روی تکنیک آژاکس قرار می دهد. با افزایش توانایی های ابزارهای توسعه جاوااسکریپت و تکنیک های آن، ممکن است تصمیم به استفاده از مرورگرهای وب به عنوان بستر توسعه برنامه های خود بگیرید.

یک رشته درخواست ساده که فقط می تواند شامل جفت های name/value باشد، برای ارتباط و ارسال داده های زیاد، که حاصل تغییر حالت سرور و کلاینت می باشند، به اندازه کافی قوی نیست. راه حل بهتر این است که تغییرات مدل داده ای به صورت XML به سرور ارسال شود.

چگونه می توان اطلاعات را به صورت XML به سرور ارسال کرد؟

شما می توانید داده XML را به عنوان قسمتی از بدنه درخواست، به سرور ارسال کنید، همانگونه که مقدار رشته درخواست را با استفاده از درخواست های نوع post به سرور ارسال کردید. سرور می تواند XML ارسالی را از بدنه درخواست بخواند و پردازش نماید.

مثال بعد نشان می دهد که چگونه با استفاده از درخواست های آژاکس، اطلاعات XML به سرور ارسال کنید. شکل ۳-۵ صفحه ای را نشان می دهد که شامل لیست انتخابی است که کاربر می تواند حیوانات خانگی را که نگهداری می کند از آن انتخاب نماید. این یک مثال بسیار ساده است اما به خوبی نشان می دهد که چگونه می توان مقادیر XML را به سرور ارسال نمود.




**شکل ۳-۵ : گزینه های انتخاب شده در لیست، به صورت XML به سرور ارسال می شوند.**

**کد ۳-۹ فایل postingXML.html را نشان می دهد.**

**کد ۳-۹ :** postingXML.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Sending an XML Request</title>
<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function createXML() {
      var xml = "<pets>";
      var options = document.getElementById("petTypes").childNodes;
      var option = null;
      for(var i = 0; i < options.length; i++) {
            option = options[i];
            if(option.selected) {
                  xml = xml + "<type>" + option.value + "<\/type>";
            }
      }
      xml = xml + "<\/pets>";
```

```
	return xml;
}

function sendPetTypes() {
	createXMLHttpRequest();
	var xml = createXML();
	var url = "PostingXMLExample?timeStamp=" + new
Date().getTime();
	xmlHttp.open("POST", url, true);
	xmlHttp.onreadystatechange = handleStateChange;
	xmlHttp.setRequestHeader("Content-Type", "application/x-www-
form-urlencoded;");
	xmlHttp.send(xml);
}

function handleStateChange() {
	if(xmlHttp.readyState == 4) {
		if(xmlHttp.status == 200) {
			parseResults();
		}
	}
}

function parseResults() {
	var responseDiv = document.getElementById("serverResponse");
	if(responseDiv.hasChildNodes()) {
		responseDiv.removeChild(responseDiv.childNodes[0]);
	}
	var responseText =
document.createTextNode(xmlHttp.responseText);
	responseDiv.appendChild(responseText);
}

</script>
</head>
<body>
<h1>Select the types of pets in your home:</h1>
<form action="#">
	<select id="petTypes" size="6" multiple="true">
		<option value="cats">Cats</option>
		<option value="dogs">Dogs</option>
		<option value="fish">Fish</option>
		<option value="birds">Birds</option>
		<option value="hamsters">Hamsters</option>
		<option value="rabbits">Rabbits</option>
	</select>
	<br/><br/>
	<input type="button" value="Submit Pets"
onclick="sendPetTypes();"/>
</form>
<h2>Server Response:</h2>
<div id="serverResponse"></div>
</body>
</html>
```

**این مثال بسیار شبیه مثال قبل در مورد ارسال به روش post می باشد. تنها تفاوت آن این است که به جای ارسال رشته درخواست که شامل جفت های name/value بود، یک رشته XML تولید شده و به سرور ارسال می شود.**

فشردن دکمه submit Pets در صفحه، تابع sendPetTypes را فراخوانی می نماید. مانند مثال های قبل، این تابع ابتدا یک نمونه شئ XMLHttpRequest می سازد . سپس این تابع، تابعی کمکی به نام createXML را فراخوانی می کند که با استفاده از مقادیر انتخاب شده از لیست، یک رشته کارکتری XML می سازد.

تابع createXML با استفاده از متد document.getElementById اشاره گری به مؤلفه select (لیست انتخاب) را در اختیار می گیرد. سپس این تابع همه فرزندان این مؤلفه را بررسی می کند و برای هر مؤلفه فرزند که توسط کاربر انتخاب شده است یک تگ XML ساخته و آن را به انتهای رشته XML نهایی می افزاید. پس از بررسی همه مؤلفه های فرزند، تگ پایانی pets را به انتهای رشته XML افزوده می شود و مقدار XML به دست آمده به عنوان خروجی، برگردانده می شود.

وقتی رشته XML به دست آمد، تابع sendPetTypes شئ XMLHttpRequest را تنظیم می کند و مقدار XML به دست آمده را به صورت پارامتر متد send به سرور ارسال می نماید.

نکته: به استفاده از '\' قبل از '/' در تگ پایانی هنگام تولید رشته XML دقت کنید. از این روش برای جلوگیری از بروز خطاهای احتمالی توسط مرورگرها استفاده می شود. بیشتر مرورگرها نیازی به این روش برای درست کار کردن، ندارند اما بعضی از مرورگرها ممکن است علامت '/' را با علامت شروع توضیح اشتباه[۳۳] گرفته و تولید خطا نمایند.

خوانندگان دقیق به این نکته توجه خواهند داشت که که متد send هم می تواند مقادیر رشته کارکتری و هم سند XML به سرور ارسال نماید، پس چرا در این مثال، مقدار XML نیز با استفاده از الحاق رشته های کارکتری به یکدیگر ایجاد شد و چرا مستقیماً یک سند XML ساخته نشد که به راحتی با استفاده از متد send به سرور ارسال شود؟ در پاسخ باید بگوییم که متاسفانه، تاکنون تکنیک مستقل از مرورگر به این منظور(ساخت سند XML به صورت پویا) وجود ندارد. اینترنت اکسپلورر این قابلیت را با استفاده از اشیاء اکتیوکس و مرورگر Mozilla آن را با استفاده از اشیاء جاوااسکریپت انجام می دهند اما بقیه مرورگرها یا به کلی آن را پشتیبانی نمی کنند و یا به روشهای متفاوتی آن را پیاده سازی کرده اند.

کد سمت سرور برای خواندن XML که در کد ۳-۱۰ نشان داده شده است، کمی پیچیده به نظر می رسد. در این مثال از کد جاوا سرولت استفاده شده است، هرچند که می توان از

[33] - comment

زبانهای دیگر نیز استفاده کرد. متد doPost در این کد هنگام در یافت درخواست، فراخوانی و فعال می شود. این تابع، از تابعی کمکی به نام createXMLFromRequestBody برای جداسازی XML از بدنه درخواست استفاده می کند. سپس رشته XML را با استفاده از اینترفیس[۳۴] JAXP به شئ document تبدیل می نماید. توجه کنید که شئ document نمونه ای از اینترفیس Document است که توسط W3C مشخص شده است. بنابراین، این شئ همانند شئ document در مرورگرها، متدهایی مثل getElementByTagName را داراست. شما می توانید از این متد برای به دست آوردن لیستی از مؤلفه های type در document استفاده کنید. برای هر مؤلفه type مقدار متنی آن به دست می آید و به انتهای یک رشته کارکتری اضافه می شود.(توجه داشته باشید که مقدار یک مؤلفه به عنوان اولین فرزند آن مؤلفه قابل دسترس است) بعد از اینکه همه مؤلفه های type بررسی شدند، مقدار رشته کارکتری حاصل، به عنوان پاسخ به مرورگر بازگردانده می شود.

کد ۳-۱۰ : postingXMLExample.java

```
package ajaxbook.chap3;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class PostingXMLExample extends HttpServlet {
      protected void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
            String xml = readXMLFromRequestBody(request);
            Document xmlDoc = null;
            try {
                  xmlDoc =
                  DocumentBuilderFactory.newInstance().newDocumentBui
            lder()
                  .parse(new ByteArrayInputStream(xml.getBytes()));
            }
            catch(ParserConfigurationException e) {
                  System.out.println("ParserConfigurationException: "
            + e);
            }
            catch(SAXException e) {
                  System.out.println("SAXException: " + e);
            }
```

[34] - interface

```
      /* Note how the Java implementation of the W3C DOM has the same
methods
      * as the JavaScript implementation, such as
getElementsByTagName and
      * getNodeValue.
      */
      NodeList selectedPetTypes =
xmlDoc.getElementsByTagName("type");
      String type = null;
      String responseText = "Selected Pets: ";
      for(int i = 0; i < selectedPetTypes.getLength(); i++) {
            type =
      selectedPetTypes.item(i).getFirstChild().getNodeValue();
            responseText = responseText + " " + type;
      }
      response.setContentType("text/xml");
      response.getWriter().print(responseText);
}

private String readXMLFromRequestBody(HttpServletRequest request){
      StringBuffer xml = new StringBuffer();
      String line = null;
      try {
            BufferedReader reader = request.getReader();
            while((line = reader.readLine()) != null) {
                  xml.append(line);
            }
      }
      catch(Exception e) {
            System.out.println("Error reading XML: " + e.toString());
      }
      return xml.toString();
}
}
```

## ارسال اطلاعات به سرور با استفاده از JSON

حالا که بیشتر با جاوااسکریپت و قابلیت های آن کار کردید، حتماً مایلید که بیشتر با امکانات سمت مرورگرها کار کنید. البته بعد از مشاهده مثال قبل که از XML به منظور ارسال ساختارهای پیچیده به سرور استفاده می کرد ممکن است علاقه شما به این مورد کمتر شده باشد. ساخت رشته های XML با استفاده از الحاق رشته های کوچک تکنیک قوی و خوشایندی برای ساخت و تغییر ساختارهای داده ای در قالب XML نیست. اجازه دهید با تکنیک جدیدی به نام JSON آشنا شویم.

JSON گونه ای جدید از ساختارهای XML است که می توانید در آدرس www.json.org آن را بیابید. JSON یک فرمت متنی است که مستقل از زبانهای برنامه نویسی است که از آن استفاده می کنند اما از قراردادهایی استفاده می کند که به خانواده

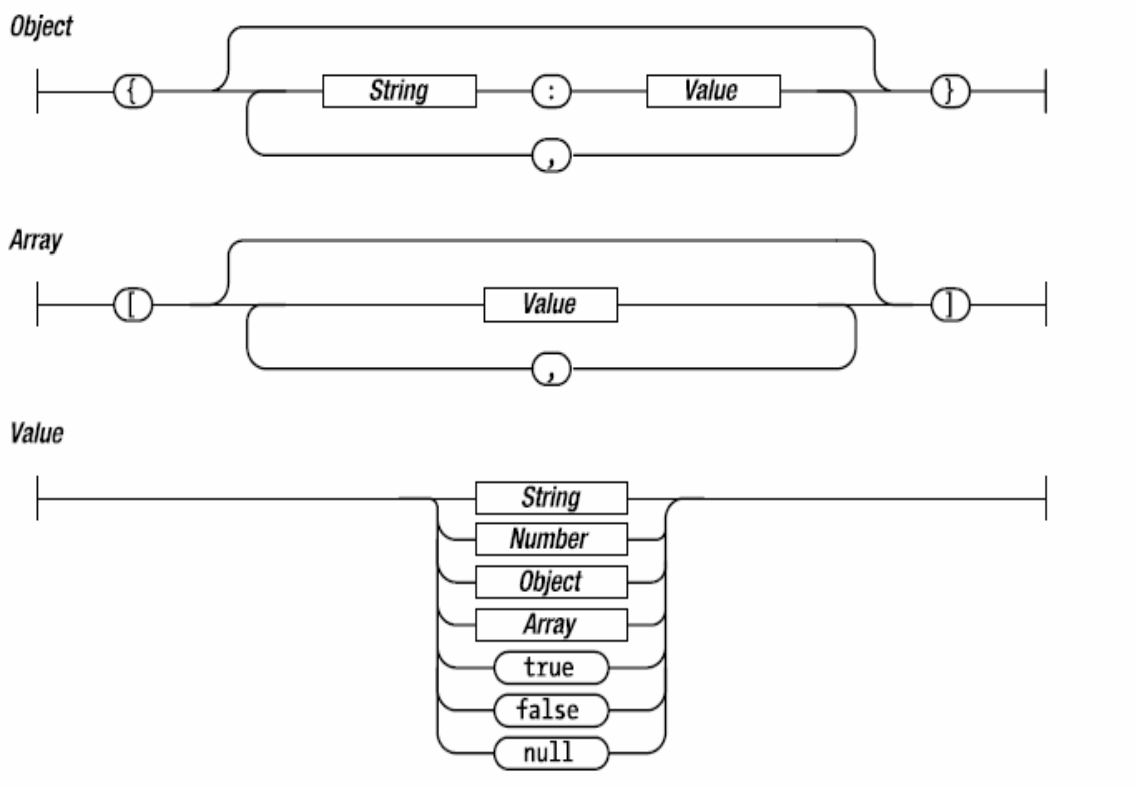
زبان C مثل C, C#, javascript و غیره شبیه است. JSON از دو ساختمان داده زیر که توسط همه زبانهای برنامه نویسی پشتیبانی می شوند، ساخته شده است.

- مجموعه ای از جفت های name/value . در زبانهای برنامه نویسی مدرن این مجموعه ها به صورت های شئ، رکورد یا دیکشنری[۳۵] پیاده سازی شده اند.
- یک لیست مرتب از مقادیر، که معمولاً به صورت آرایه پیاده سازی می شوند.

از آنجایی که این ساختارهای داده ای به وسیله بسیاری از زبان های برنامه نویسی پشتیبانی می شوند، JSON گزینه مناسبی برای قالب تبادل داده بین سیستم های ناهمگون می باشد. علاوه بر این، از آنجایی که JSON براساس استانداردهای جاوااسکریپت بنا شده است، تقریباً با همه مرورگرهای مدرن سازگاری دارد.

یک شئ JSON یک مجموعه نامرتب از جفت های name/value است. این شئ با علامت { شروع و با علامت } خاتمه می یابد. برای جداکردن name و value از علامت : (کولن) استفاده می شود. یک آرایه JSON مجموعه ای مرتب از مقادیر است که توسط علامت [ شروع و با علامت ] خاتمه می یابد. برای جداکردن مقادیر این آرایه از , (کاما) استفاده می شود. یک عضو این آرایه می تواند یک رشته کارکتری (که با علامت " " محصور شده باشد)، عدد، مقدار بولین، شئ یا یک آرایه باشد. این به شما امکان می دهد تا ساختارهای تودرتو ایجاد کنید. شکل ۳-۶ راهنمای خوبی برای نمایش چگونگی ساخت شئ JSON است.

[35] - dictionary




شکل ۳-۶ : توضیح گرافیکی از ساختار شئ JSON (منبع : سایت www.json.org)

یک مثال ساده از شئ Employee را با JSON بررسی می کنیم. یک شئ Employee (کارمند) ممکن است شامل اطلاعاتی مثل نام، نام خانوادگی، شماره کارمندی و سِمت باشد. با استفاده از JSON می توان این شئ را اینگونه توصیف کرد:

```
var employee = {
      "firstName" : John
      , "lastName" : Doe
      , "employeeNumber" : 123
      , "title" : "Accountant"
}
```

بعد از تعریف این شئ، با استفاده از علامت گذاری استاندارد اشیاء، یعنی نقطه، می توان به خواص و اعضای آن دسترسی داشت. مثل کد زیر :

```
var lastName = employee.lastName;  //Access the last name
var title = employee.title;         //Access the title
employee.employeeNumber = 456;            //Change the employee
                                          number
```

JSON یک فرمت تبادل داده سبک است و از این بابت به خود افتخار می کند. همان شئ employee که توسط JSON ارائه شد، توسط XML به گونه زیر ارائه خوهد شد:

```
<employee>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <employeeNumber>123</employeeNumber>
      <title>Accountant</title>
</employee>
```

روشن است که ساختار کدینگ JSON کوچکتر از ساختار کدینگ XML است. حجم کمتر کدینگ به روش JSON کارآیی و سرعت را هنگام ارسال داده های حجیم و بزرگ روی شبکه، بالا می برد.

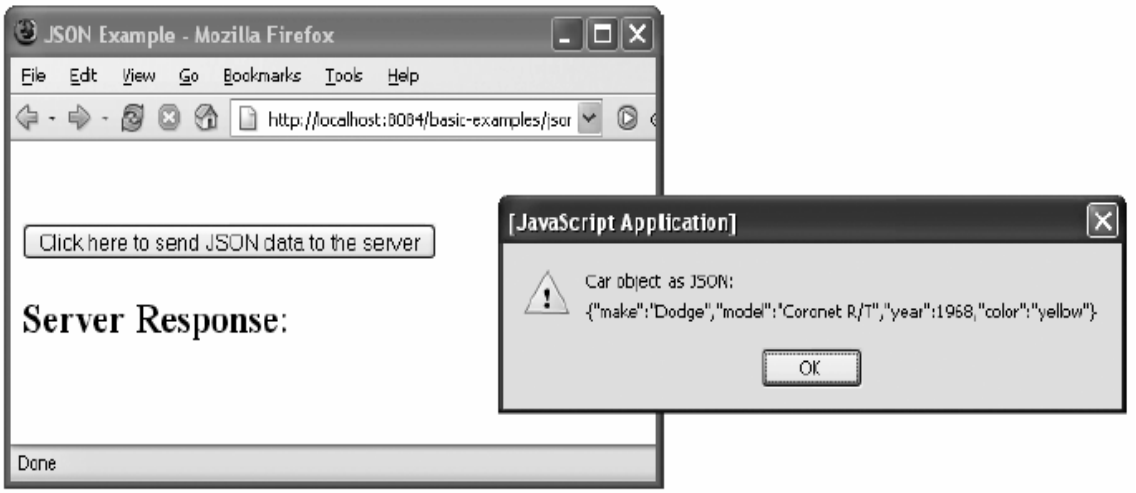
وب سایت www.json.org حداقل ۱۴ نمونه مختلف از کتابخانه های JSON را برای زبانهای برنامه نویسی مختلف تهیه کرده است تا کاربران بتوانند با هر زبان سمت سروری که مایل هستند، از طریق JSON با مرورگرها تبادل داده داشته باشند.

## مثال استفاده از JSON

مثال زیر نشان می دهد که چگونه با استفاده از JSON اشیاء جاوااسکریپت را به رشته کارکتری تبدیل کنید و آن را با استفاده از تکنیک های آژاکس به سرور ارسال نمایید تا سرور با استفاده از آن رشته، یک شئ جهت پردازش بسازد. این مثال منطق کاری خاصی ندارد و از تعامل کمی با کاربر، برخوردار است تا بیشتر روی جنبه های استفاده از JSON، هم سمت کلاینت و هم سمت سرور، تمرکز کنید. شکل ۳-۷ نمایش اطلاعات مربوط به یک شئ ماشین را توسط روش JSON نشان می دهد.

به دلیل اینکه این مثال بسیار شبیه به مثال های قبلی در مورد متد post می باشد، بیشتر روی تکنیک های ویژه JSON در آن متمرکز می شویم. فشردن دکمه روی صفحه، تابع doJSON را فراخوانی می کند. این تابع در ابتدا تابع getCarObject را به منظور برگرداندن نمونه ای جدید از شئ car فراخوانی می کند. سپس با استفاده از توابع کتابخانه جاوااسکریپت JSON (قابل دانلود از سایت www.json.org) شئ car به رشته کارکتری JSON تبدیل می شود و با استفاده از شئ XMLHttpRequest به سرور فرستاده می شود.

در سمت سرور، با استفاده از کتابخانه های تبدیل اشیاء JSON به جاوا، تبدیل درخواست از نوع JSON به سرویس های قابل فهم توسط جاوا بسیار آسان است. با استفاده از کتابخانه های موجود جهت تبدیل اشیاء JSON به سرویس های قابل فهم توسط زبان های مختلف، می توان این مثال را با استفاده از هر زبان سمت سرور، پیاده سازی کرد.




**شکل ۳-۷ : نمایش اطلاعات شئ car در پنجره پیام**

**متد doPost در کد ۳-۱۲ درخواست نوع JSON را پردازش می نماید. این متد ابتدا با استفاده از تابع readJSONFromRequestBody رشته کارکتری JSON را از بدنه درخواست به دست می آورد. سپس با ارسال مقدار به دست آمده به سازنده شئ JSONObject نمونه ای جدید از این شئ را می سازد. این شئ به صورت اتوماتیک هنگام ایجاد، رشته ورودی به سازنده اش را تجزیه می کند. بعد از ایجاد شئ JSONObject با استفاده از متدهای get آن می توان مقدار خاصیت های مختلف آن را به دست آورد. می توان با استفاده از متدهای getInt و getString مقدار خاصیت های سال و مدل و رنگ شئ ماشین مربوطه را به دست آورد. این خواص به دست آمده به صورت رشته کارکتری به یکدیگر الحاق شده و جهت نمایش به مرورگر ارسال می شوند. شکل ۳-۸ پاسخ سرور بعد از خواندن خاصیت های شئ JSON نشان می دهد.**


**شکل ۳-۸ : پاسخ سرور بعد از خواندن رشته کارکتری JSON**

**کد ۳-۱۱ فایل JSONExample.html و کد ۳-۱۲ فایل JSONExample.java را نشان می دهند.**

**کد ۳-۱۱ :** JSONExample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JSON Example</title>

<script type="text/javascript" src="json.js"></script>
<script type="text/javascript">

var xmlHttp;

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function doJSON() {
      var car = getCarObject();
      //Use the JSON JavaScript library to stringify the Car object
      var carAsJSON = JSON.stringify(car);
      alert("Car object as JSON:\n " + carAsJSON);
      var url = "JSONExample?timeStamp=" + new Date().getTime();
      createXMLHttpRequest();
      xmlHttp.open("POST", url, true);
      xmlHttp.onreadystatechange = handleStateChange;
```

```
	xmlHttp.setRequestHeader("Content-Type",
		"application/x-www-form-urlencoded;");
	xmlHttp.send(carAsJSON);
}

function handleStateChange() {
	if(xmlHttp.readyState == 4) {
		if(xmlHttp.status == 200) {
			parseResults();
		}
	}
}

function parseResults() {
	var responseDiv = document.getElementById("serverResponse");
	if(responseDiv.hasChildNodes()) {
		responseDiv.removeChild(responseDiv.childNodes[0]);
	}
	var responseText =
document.createTextNode(xmlHttp.responseText);
	responseDiv.appendChild(responseText);
}

function getCarObject() {
	return new Car("Dodge", "Coronet R/T", 1968, "yellow");
}

function Car(make, model, year, color) {
	this.make = make;
	this.model = model;
	this.year = year;
	this.color = color;
}

</script>
</head>
<body>
<br/><br/>
<form action="#">
	<input type="button" value="Click here to send JSON data to the
server"
		onclick="doJSON();"/>
</form>
<h2>Server Response:</h2>
	<div id="serverResponse"></div>
</body>
</html>
```

کد ۳-۱۲ : JSONExample.java

```
package ajaxbook.chap3;

import java.io.*;
import java.net.*;
import java.text.ParseException;
import javax.servlet.*;
import javax.servlet.http.*;
import org.json.JSONObject;

public class JSONExample extends HttpServlet {
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      String json = readJSONStringFromRequestBody(request);
      //Use the JSON-Java binding library to create a JSON object in
Java
      JSONObject jsonObject = null;
      try {
            jsonObject = new JSONObject(json);
      }
      catch(ParseException pe) {
            System.out.println("ParseException: " + pe.toString());
      }
      String responseText = "You have a " + jsonObject.getInt("year")
+ " "
            + jsonObject.getString("make") + " " +
      jsonObject.getString("model")
            + " " + " that is " + jsonObject.getString("color") + "
      in color.";
      response.setContentType("text/xml");
      response.getWriter().print(responseText);
}

private String readJSONStringFromRequestBody(HttpServletRequest
request){
      StringBuffer json = new StringBuffer();
      String line = null;
      try {
            BufferedReader reader = request.getReader();
            while((line = reader.readLine()) != null) {
                  json.append(line);
            }
      }
      catch(Exception e) {
            System.out.println("Error reading JSON string: " +
      e.toString());
      }
      return json.toString();
}
}
```

# فصل چهارم : پیاده سازی تکنیک های مقدماتی توسط آژاکس

تاکنون با تکنیک آژاکس آشنا شده اید و می دانید که چگونه از شئ XMLHttpRequest استفاده کنید. حال باید همه این آموخته ها را به کار ببندید. اما چگونه؟ این فصل چندین موقعیت را که می توانید در آنها از آژاکس استفاده کنید به شما نشان می دهد، تا با استفاده از آنها برنامه های قوی تری بنویسید. بعضی از آنها واضح و ساده هستند و بعضی خیر، چیزی که مهم است این است که شما با مشاهده روش حل مسئله توسط ما می توانید تکنیک خود را توسعه دهید. در این فصل نیز مثالها از کد جاواسرولت در سمت سرور بهره می گیرند، اما هرکدام از آنها را می توان به راحتی با زبانهای دیگر مثل ASP.NET و PHP یا هر زبان دیگر سمت سرور پیاده سازی کرد.

## اعتبار سنجی (Performing Validation)

قانون جامع بهره وری می گوید که باید از وقوع خطاها جلوگیری کرد، اما به جز آن، شما باید به زودی نحوه گزارش خطا به کاربرانتان را تغییر دهید. قبل از آژاکس، برنامه های تحت وب، به منظور تعیین صحت داده های وارد شده توسط کاربر، مجبور به ارسال کل صفحه به سرور بودند یا از جاوااسکریپت به منظور چک کردن داده های هر فرم استفاده کنند. اما نوشتن بعضی از روتین های چک کننده در جاوااسکریپت آسان نمی باشد. البته هر روتین چک کننده که در سمت کلاینت با جاوااسکریپت نوشته می شود باید در سمت سرور هم نوشته شده باشد چراکه ممکن است قابلیت اجرای کدهای جاوااسکریپت توسط مرورگر کاربر غیرفعال شده باشد.

با آژاکس دیگر مجبور نیستید خودتان را به روتین های ساده اعتبارسنجی سمت کلاینت و دوباره نویسی آنها در سمت سرور، محدود کنید. بلکه شما می توانید به سادگی روالهای نوشته شده در سمت سرور را فراخوانی نمایید. در بیشتر موارد کار کردن به این روش ساده تر است.

وقتی از ما سوال می شود که از کجا برای شروع کاربرد آژاکس در برنامه ها باید استفاده کرد، توصیه می کنیم که از اعتبار سنجی فرمها شروع کنید. در این قسمت ما یکی از پر کاربردترین اعتبارسنجی ها را که تعیین اعتبار تاریخ هاست در قالب یک مثال آورده ایم.

کد HTML این مثال بسیار ساده است.(کد ۴-۱ را ببینید.) در این مثال یک Input Box با رویداد OnChange وجود دارد (البته شما می توانید از هر رویداد دیگری که مایلید به جای این رویداد استفاده نمایید) که روال اعتبار سنجی را فعال می کند. همانطور که می بینید متد آشنای ما یعنی CreateXMLHttpRequest فراخوانی و سپس مقدار وارد شده توسط کاربر به فایل ValidationServlet ارسال شده است. تابع CallBack نتایج را از سرور دریافت کرده و به عنوان ورودی به متد SetMessage می دهد. متد اخیر پاسخ سرور را با استفاده از رنگ مشخص روی صفحه نمایش می دهد.

کد ۴-۱ : Validation.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html>
<head>
<title>Using Ajax for validation</title>
<script type="text/javascript">
      var xmlHttp;
function createXMLHttpRequest() {
      if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
      xmlHttp = new XMLHttpRequest();
      }
}
function validate() {
      createXMLHttpRequest();
      var date = document.getElementById("birthDate");
      var url = "ValidationServlet?birthDate=" + escape(date.value);
      xmlHttp.open("GET", url, true);
      xmlHttp.onreadystatechange = callback;
      xmlHttp.send(null);
}
function callback() {
      if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
            var mes =
            xmlHttp.responseXML
            .getElementsByTagName("message")[0].firstChild.data;
            var val =xmlHttp.responseXML
            .getElementsByTagName("passed")[0].firstChild.data;
            setMessage(mes, val);
            }
      }
}
function setMessage(message, isValid) {
      var messageArea = document.getElementById("dateMessage");
      var fontColor = "red";
      if (isValid == "true") {
            fontColor = "green";
      }
      messageArea.innerHTML = "<font color=" + fontColor + ">" å
```

```
      + message + " </font>";
}
</script>
</head>
<body>
<h1>Ajax Validation Example</h1>
Birth date: <input type="text" size="10" id="birthDate"
onchange="validate();"/>
<div id="dateMessage"></div>
</body>
</html>
```

کد سمت سرور این مثال نیز همانند کد سمت کلاینت آن بسیار ساده است. (کد ۴-۲ را ببینید.)

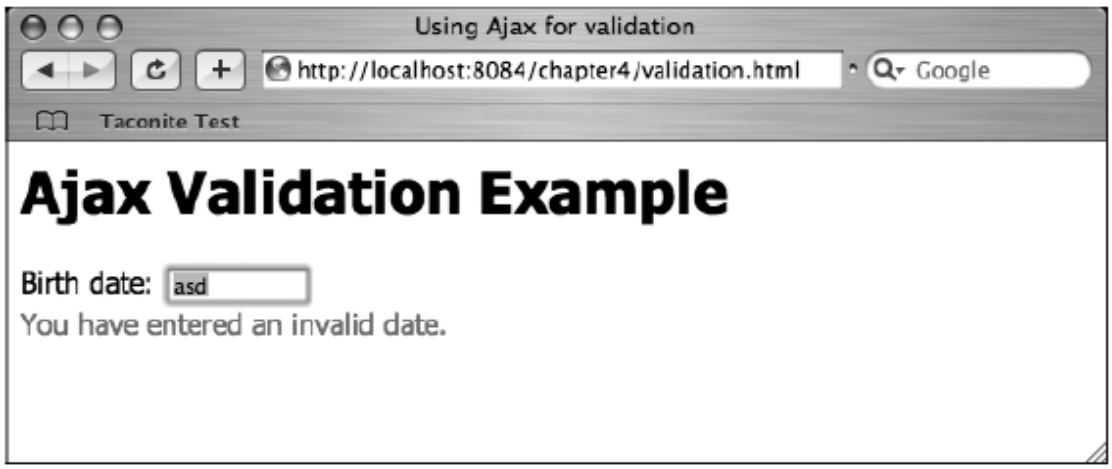
کد ۴-۲ : ValidationServlet.java

```
package ajaxbook.chap4;
import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import javax.servlet.*;
import javax.servlet.http.*;
public class ValidationServlet extends HttpServlet {
/** Handles the HTTP <code>GET</code> method.
* @param request servlet request
* @param response servlet response
*/
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      boolean passed =
      validateDate(request.getParameter("birthDate"));
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      String message = "You have entered an invalid date.";
      if (passed) {
            message = "You have entered a valid date.";
      }
      out.println("<response>");
      out.println("<passed>" + Boolean.toString(passed) +
      "</passed>");
      out.println("<message>" + message + "</message>");
      out.println("</response>");
      out.close();
}
/**
* Checks to see whether the argument is a valid date.
* A null date is considered invalid. This method
* used the default data formatter and lenient
* parsing.
*
* @param date a String representing the date to check
* @return message a String representing the outcome of the check
*/
private boolean validateDate(String date) {
      boolean isValid = true;
      if(date != null) {
```

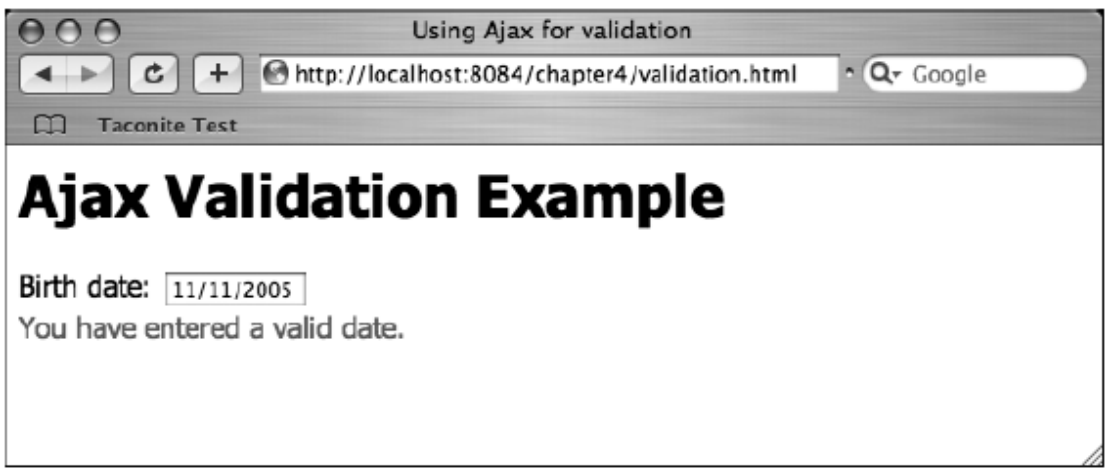
```
      SimpleDateFormat formatter= new SimpleDateFormat("MM/dd/yyyy");
      try {
            formatter.parse(date);
      } catch (ParseException pe) {
      System.out.println(pe.toString());
      isValid = false;
      }
      } else {
      isValid = false;
      }
      return isValid;
      }
}
```

**نتیجه اجرای این مثال را در شکل های ۴-۱ و ۴-۲ می بینید.**

**شکل ۴-۱ : ورود تاریخ غیر معتبر**




**شکل ۴-۲ : ورود تاریخ معتبر**



## پردازش سرآیندهای پاسخ سرور

گاهی اوقات شما نیاز به بازیابی تمام محتوای صفحه ندارید، به عنوان مثال، وقتی که فقط می خواهید فعالیت سرور را چک کنید. ممکن است فقط بخواهید سرآیندهای پاسخ ارسال شده توسط سرور را چک کنید و از نمایش محتوای پاسخ خودداری کنید. با خواندن سرآیندهای پاسخ، شما می توانید اطلاعاتی مثل نوع محتوای پاسخ، اندازه آن و حتی تاریخ آخرین تغییرات را به دست آورید.

روش استاندارد ارسال درخواست در این مورد(دستیابی به سرآیندهای پاسخ) به جای استفاده از درخواست به روش Post یا Get که قبلاً توضیح داده شدند، استفاده از درخواست نوع HEAD می باشد. هنگامی که سرور به درخواست نوع HEAD پاسخ می دهد، فقط سرآیندها را به مرورگر کاربر می فرستد. با توجه به حذف محتوا از پاسخ، پاسخ این نوع درخواست خیلی کوچکتر از پاسخ درخواست های نوع Post یا Get می باشد.

کد ۳-۴ روشهای مختلفی را که می توانیدسرآیندهای پاسخ را از شئ XMLHttpRequest به دست آورید و برای اهداف خود استفاده کنید، نشان می دهد. صفحه این مثال شامل چهار لینک است که هرکدام از متدهای مختلف شئ XMLHttpRequest برای خواندن سرآیندهای پاسخ رسیده، استفاده می کند.

کد ۳-۴ : readingResponseHeader.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Reading Response Headers</title>

<script type="text/javascript">
var xmlHttp;
var requestType = "";

function createXMLHttpRequest() {
  if (window.ActiveXObject) {
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else if (window.XMLHttpRequest) {
    xmlHttp = new XMLHttpRequest();
  }
}

function doHeadRequest(request, url) {
  requestType = request;
  createXMLHttpRequest();
  xmlHttp.onreadystatechange = handleStateChange;
```

```
   xmlHttp.open("HEAD", url, true);
   xmlHttp.send(null);
}

function handleStateChange() {
   if(xmlHttp.readyState == 4) {
      if(requestType == "allResponseHeaders") {
         getAllResponseHeaders();
      }
      else if(requestType == "lastModified") {
         getLastModified();
      }
      else if(requestType == "isResourceAvailable") {
         getIsResourceAvailable();
      }
   }
}

function getAllResponseHeaders() {
   alert(xmlHttp.getAllResponseHeaders());
}

function getLastModified() {
   alert("Last Modified: " + xmlHttp.getResponseHeader("Last-
   Modified"));
}

function getIsResourceAvailable() {
   if(xmlHttp.status == 200) {
      alert("Successful response");
   }
   else if(xmlHttp.status == 404) {
      alert("Resource is unavailable");
   }
   else {
      alert("Unexpected response status: " + xmlHttp.status);
   }
}

</script>
</head>

<body>
<h1>Reading Response Headers</h1>

<a href="javascript:doHeadRequest('allResponseHeaders',
   'readingResponseHeaders.xml');">Read All Response Headers</a>
<br/>

<a href="javascript:doHeadRequest('lastModified',
   'readingResponseHeaders.xml');">Get Last Modified Date</a>
<br/>

<a href="javascript:doHeadRequest('isResourceAvailable',
   'readingResponseHeaders.xml');">Read Available Resource</a>
<br/>

<a href="javascript:doHeadRequest('isResourceAvailable',
   'not-available.xml');">Read Unavailable Resource</a>
</body>
</html>
```

لینک اول در صفحه از متد getAllResponseHeader مربوط به شئ XMLHttpRequest استفاده می کند.این متد همه سرآیندهای پاسخ را به صورت رشته کاراکتری برمی گرداند. در این مثال، سرآیندهای پاسخ در پنجره پیام نمایش داده می شوند. این متد در عمل کمتر استفاده می شود زیرا همه سرآیندها را در قالب رشته کاراکتری بر می گرداند و بازیابی یک سرآیند مشخص با استفاده از خروجی این متد نیاز به تجزیه و تحلیل رشته خروجی دارد.

متد getResponseHeader این مشکل را با برگرداندن مقدار مربوط به یک سرآیند مشخص در پاسخ حل کرده است. این متد فقط یک ورودی دارد که نام سرآیندی است که مقدار آن مورد توجه است. در این مثال از این متد به منظور نمایش تاریخ آخرین تغییرات سند ( Last Modified) در یک پنجره پیام استفاده شده است. یک نمونه برنامه واقعی برای این متد می تواند چک کردن یک منبع روی سرور در زمان مشخص باشد. مرورگر هنگامی محتوای صفحه جاری را با استفاده از یک منبع روی سرور تغییر خواهد داد که مقدار این سرآیند بعد از آخرین سرکشی تغییر کرده باشد.

دو لینک آخر در صفحه با استفاده از شئ XMLHttpRequest کدهای وضعیت HTTP را که توسط سرور برگردانده می شوند را بررسی می کنند. متدهای مربوط به وضعیت شئ XMLHttpRequest مقدار کد وضعیت سرور را به صورت مقدار عدد صحیح برمی گرداند. کد وضعیت ۲۰۰ نشان دهنده پاسخ طبیعی و موفقیت آمیز از سمت سرور است. برعکس، کد وضعیت ۵۰۰ نشان می دهد که خطایی هنگامی که سرور جهت پاسخ دادن به درخواست تلاش می کرده است، اتفاق افتاده است.

این مثال از کدهای وضعیت HTTP به منظور تشخیص وجود یک منبع[۳۶] مشخص روی سرور استفاده می کند. کد وضعیت ۴۰۴ نشان می دهد که منبع خواسته شده روی سرور موجود نمی باشد. لینک "Read Available Resource" روی صفحه یک فایل XML را که روی سرور موجود می باشد، درخواست می نماید. به خاطر وجود فایل خواسته شده روی سرور، کد وضعیت ۲۰۰ که نشان دهنده پاسخ موفقیت آمیز است برگردانده می شود. آخرین لینک روی صفحه که با عبارت "Read Unavailable Resource" مشخص شده است، فایلی را را که روی سرور نیست، درخواست می نماید. سرور با کد وضعیت ۴۰۴ پاسخ می دهد. تابع جاوااسکریپت پاسخ سرور را بررسی می کند و با مشاهده کد وضعیت ۴۰۴، یک

[36] - resource

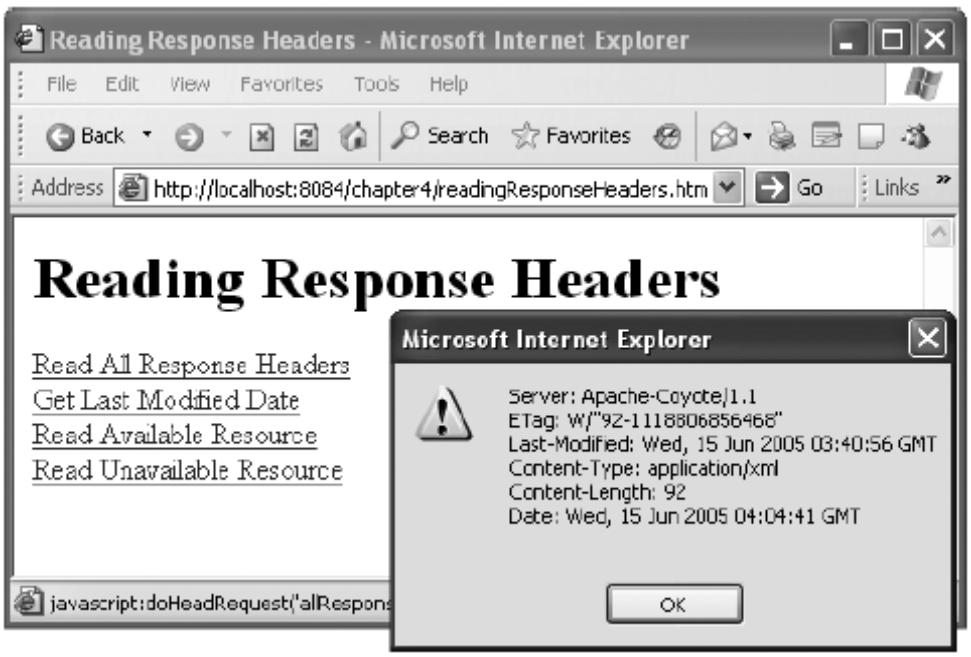
پنجره پیام را که بیانگر عدم وجود فایل خواسته شده روی سرور است، به کاربر نشان می دهد.

کد ۴-۴ فایل ReadingResponseHeader.xml را نشان می دهد.

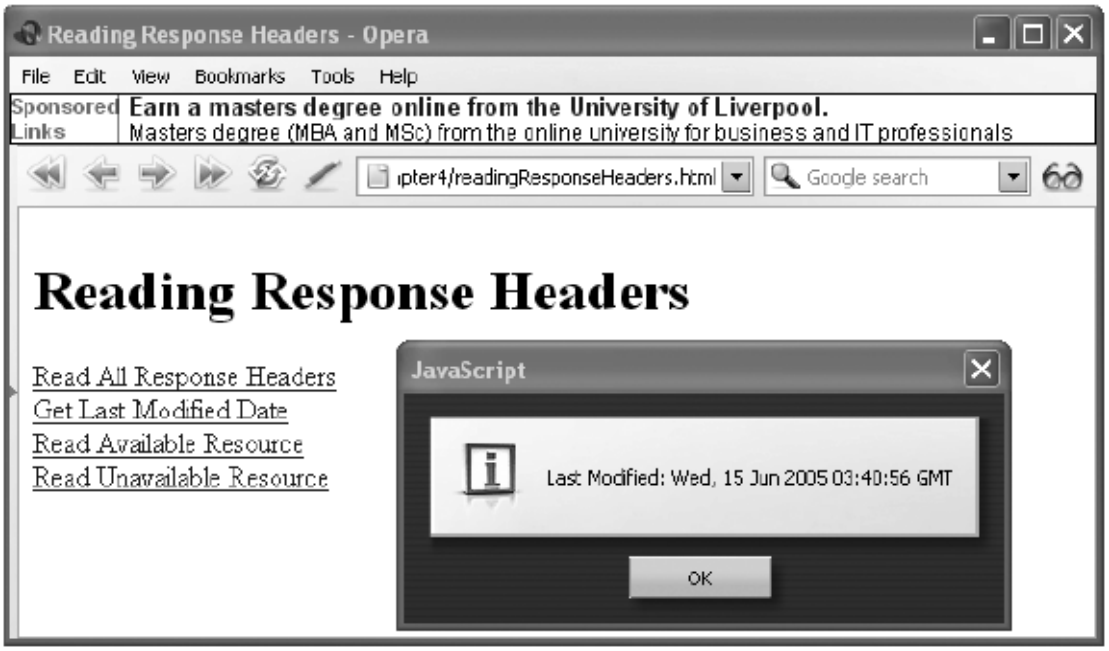
کد ۴-۴ : ReadingResponseHeader.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<readingResponseHeaders>
</readingResponseHeaders>
```

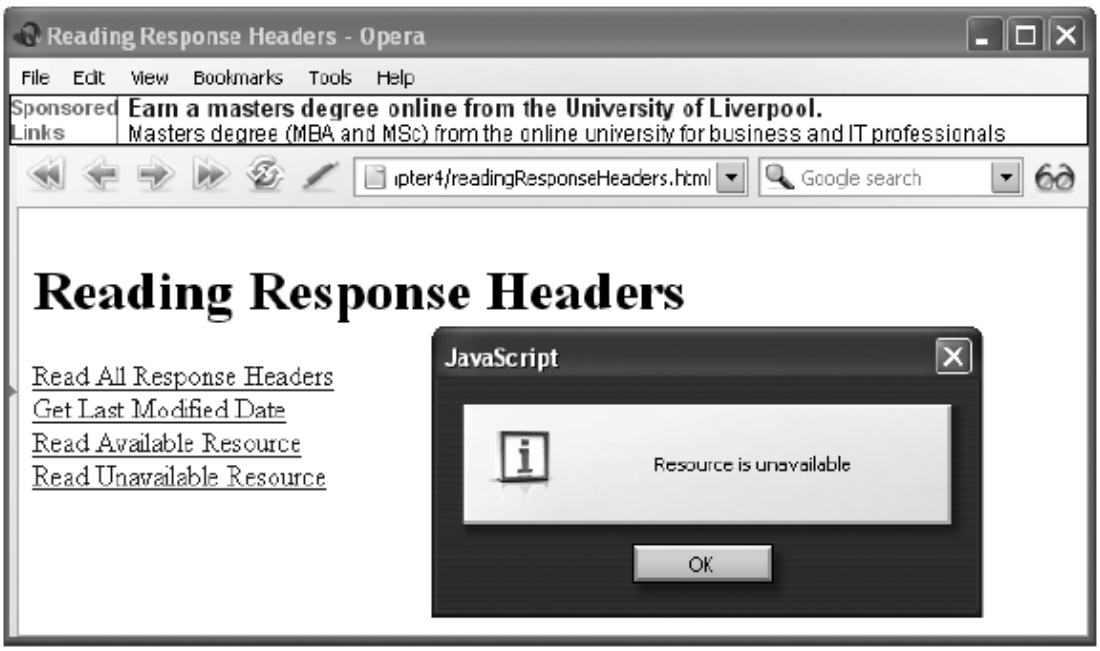
شکل ۳-۴ نتیجه نمایش همه سرآیندهای پاسخ سرور را نشان می دهد. شکل ۴-۴ نتیجه بررسی سرآیند Last-Modified را نشان می دهد و شکل ۵-۴ نتیجه وجود یک فایل مشخص روی سرور را نشان می دهد.




شکل ۳-۴ : نمایش همه سرآینهای پاسخ سرور




شکل ۴-۴ : بررسی یک سرآیند مشخص، در این مثال سرآیند Last-Modified




شکل ۵-۴ : بررسی وجود یک منبع خاص روی سرور

## بارگذاری[۳۷] لیست به صورت پویا

برنامه های وب معمولاً براساس روش طراحی ویزارد[۳۸] ساخته می شوند، بدین ترتیب که هر صفحه در حال نمایش، از کاربر مقداری اطلاعات به صورت ورودی دریافت می کند و هر صفحه جدید با اطلاعات دریافت شده در صفحات قبل، ساخته می شود. این روش طراحی در مواقعی که کاربر می خواهد عملی را انجام دهد که آن عمل ماهیت مرحله به مرحله دارد، مفید است. متاسفانه بسیاری از وب سایتها از این رهیافت بهره می برند چون تاکنون راهی جز این نداشتند.پیش از ظهور آژاکس اگر نگوییم غیرممکن بود، بسیار سخت بود که یک صفحه را بدون بازیابی مجدد[۳۹] همه صفحه، هنگامی که فقط قسمت مشخصی از صفحه بر اساس ورودی کاربر نیاز به تغییر دارد، به صورت پویا تغییر داد.

یک تکنیک به منظور جلوگیری از بازیابی مجدد کل صفحه این است که اطلاعات روی صفحه به صورت مخفی نگه داشته شوند و هنگام نیاز نمایش داده شوند. مثلاً هنگامی که مقادیر درون جعبه انتخاب[۴۰] B به مقدار انتخاب شده در جعبه انتخاب A بستگی دارد، مقادیر مورد نیاز جعبه انتخاب B می توانند درون جعبه های انتخاب مخفی نگهداری شوند. هنگامی که مقدار انتخاب شده در جعهبه انتخاب A تغییر کند، با استفاده از جاوااسکریپت می توان مشخص کرد که کدام یک از جعبه های انتخاب مخفی باید نمایش داده شود. تکنیک دیگر حل این مشکل این است که مقدار خاصیت Option مربوط به جعبه انتخاب B را با استفاده از مؤلفه های یک جعبه انتخاب مخفی تنظیم کرد. این تکنیک ها مفیدند اما کاربرد آنها محدود به مواردی است که تغییرات صفحه به مجموعه متناهی و ثابت، براساس انتخاب کاربر، وابسته باشد.

فرض کنید می خواهید سرویس تبلیغات اتومبیل به صورت دسته بندی شده راه اندازی نمایید. معمولاً خریداران، اتومبیل مورد نظر خود را بر اساس سال ساخت، کارخانه سازنده و مدل آن جستجو می کنند. به منظور جلوگیری از خطاهای تایپی توسط کاربر تصمیم می گیرید که سال ساخت، کارخانه سازنده و مدل به صورت جعبه های انتخاب در اختیار کاربران قرار گیرند. تغییر گزینه جاری در جعبه انتخاب سال ساخت یا کارخانه سازنده باید مدلهای مربوط به آن سال و کارخانه را در جعبه انتخاب مربوطه نمایش دهد.

---

37 - Loading
38 - Wizard
39 - Refreshing
40 - Select Box

توجه داشته باشید که با انتخاب هر سال ساخت، کارخانه های سازنده نیز تغییر خواهند کرد و همچنین مدلهای هر کارخانه سازنده در هر سال متفاوت خواهد بود. می بینید که با افزایش تعداد سالهای ساخت، کارخانه ها و مدلها حالات زیادی به منظور انتخاب ایجاد می شوند که استفاده از دو تکنیک قبل را غیر ممکن می سازد.

شما می توانید این مشکل را به راحتی توسط آژاکس حل کنید. هرگاه که سال ساخت یا کارخانه سازنده در جعبه های انتخاب مربوطه تغییر کنند، یک درخواست ناهمگام به سرور ارسال می شود که لیست مدلهای موجود مربوط به آن سال و کارخانه سازنده را جستجو می کند. کد سمت سرور مسئول جستجو و مشخص کردن مدلهای خواسته شده می باشد. در سمت سرور معمولاً با استفاده از پایگاه داده های رابطه ای عمل جستجو روی مدلها صورت می گیرد. هنگامی که که مدلهای مورد نظر یافت شدند، سرور آنها را درون فایل XML قرار داده و به مرورگر برمی گرداند. مرورگر مسئول پردازش پاسخ سرور است و باید مقادیر جعبه انتخاب مربوط به مدلها را را با استفاده از مقادیر ارسال شده توسط سرور، به روزرسانی نماید.

کد ۵-۴ نشان می دهد که چگونه می توان محتوای یک جعبه انتخاب را براساس انتخاب های دو جعبه انتخاب دیگر، به صورت پویا تغییر داد. این مثال فقط از ۴ سال ساخت و ۳ کارخانه سازنده و ۴ مدل به ازای هر سال و کارخانه، استفاده می کند. بدین ترتیب ۴۸ حال ترکیب برای سال، کارخانه و مدل وجود خواهد داشت. این مثال را نمی توان با استفاده از تکنیک پنهان سازی انجام داد چونکه نیاز به ۴۸ حالت مخفی دارد که در هر لحظه تنها یکی از آنها باید نمایش داده شوند.

کد ۵-۴ : DynamicLists.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Dynamically Filling Lists</title>

<script type="text/javascript">
var xmlHttp;

function createXMLHttpRequest() {
   if (window.ActiveXObject) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   else if (window.XMLHttpRequest) {
   xmlHttp = new XMLHttpRequest();
   }
}
```

```
function refreshModelList() {
   var make = document.getElementById("make").value;
   var modelYear = document.getElementById("modelYear").value;
   if(make == "" || modelYear == "") {
      clearModelsList();
      return;
   }
   var url = "RefreshModelList?"
      + createQueryString(make, modelYear) + "&ts=" + new
      Date().getTime();
   createXMLHttpRequest();
   xmlHttp.onreadystatechange = handleStateChange;
   xmlHttp.open("GET", url, true);
   xmlHttp.send(null);
}

function createQueryString(make, modelYear) {
   var queryString = "make=" + make + "&modelYear=" + modelYear;
   return queryString;
}

function handleStateChange() {
   if(xmlHttp.readyState == 4) {
      if(xmlHttp.status == 200) {
         updateModelsList();
      }
   }
}

function updateModelsList() {
   clearModelsList();
   var models = document.getElementById("models");
   var results = xmlHttp.responseXML.getElementsByTagName("model");
   var option = null;
   for(var i = 0; i < results.length; i++) {
      option = document.createElement("option");
      option.appendChild
         (document.createTextNode(results[i].firstChild.nodeValue));
      models.appendChild(option);
   }
}

function clearModelsList() {
   var models = document.getElementById("models");
   while(models.childNodes.length > 0) {
      models.removeChild(models.childNodes[0]);
   }
}

</script>

</head>
<body>

   <h1>Select Model Year and Make</h1>
   <form action="#">
      <span style="font-weight:bold;">Model Year:</span>
      <select id="modelYear" onchange="refreshModelList();">
         <option value="">Select One</option>
         <option value="2006">2006</option>
```

```
      <option value="1995">1995</option>
      <option value="1985">1985</option>
      <option value="1970">1970</option>
    </select>
    <br/><br/>
    <span style="font-weight:bold;">Make:</span>

    <select id="make" onchange="refreshModelList();">
      <option value="">Select One</option>
      <option value="Chevrolet">Chevrolet</option>
      <option value="Dodge">Dodge</option>
      <option value="Pontiac">Pontiac</option>
    </select>
    <br/><br/>

    <span style="font-weight:bold;">Models:</span>
    <br/>

    <select id="models" size="6" style="width:300px;">
    </select>
  </form>
</body>
</html>
```

به روزرسانی صفحه با استفاده از رویدار OnChange مربوط به جعبه های انتخاب سال و کارخانه اتفاق می افتد. هنگامی که یکی از جعبه های انتخاب تغییر کند، مرورگر یک درخواست ناهمگام به سرور ارسال می کند. درخواست شامل رشته ای است که حاوی سال ساخت و کارخانه انتخاب شده می باشد.

در سمت سرور کد RefreshModelList درخواست مرورگر را دریافت کرده و لیست مدلهای مربوط به سال و کارخانه مورد نظر را مشخص می نماید. این کد در ابتدا رشته درخواست را به منظور مشخص کردن سال و کارخانه، پردازش می نماید. وقتی که این مقادیر به دست آمدند، این کد درون مجموعه ای از مدلها به جستجو می پردازد. درصورتی که موردی یافت شود آن را به رشته XML نتیجه، می افزاید. وقتی همه مدلهای مورد نظر یافت شد، رشته XML آماده شده به عنوان پاسخ به مرورگر برگردانده می شود.

توجه داشته باشید که در دنیای واقعی کد سمت سرور اینگونه مثالها هرگز به این سادگی نخواهد بود و در واقع با استفاده از پایگاه داده و مکانیزم های آن عمل جستجو انجام خواهد شد. در این مثال به منظور سادگی و تمرکز روی اصل بحث از آوردن کدهای سنگین سمت سرور خودداری شده است.

کد ۶-۴ فایل RefreshModelListServlet.java را نشان می دهد.

کد ۶-۴ : RefreshModelListServlet.java

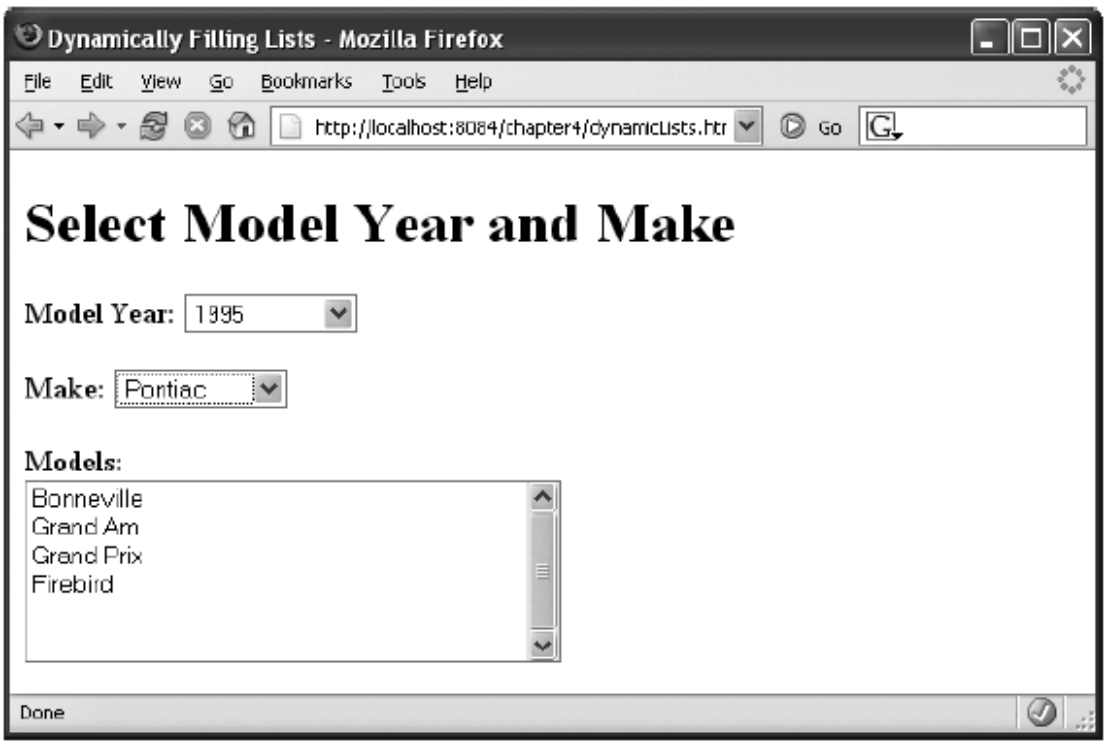
```
package ajaxbook.chap4;

import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.servlet.*;
import javax.servlet.http.*;

public class RefreshModelListServlet extends HttpServlet {

private static List availableModels = new ArrayList();

protected void processRequest(HttpServletRequest request
   , HttpServletResponse response)
throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   int modelYear =
   Integer.parseInt(request.getParameter("modelYear"));
   String make = request.getParameter("make");
   StringBuffer results = new StringBuffer("<models>");
   MakeModelYear availableModel = null;
   for(Iterator it = availableModels.iterator(); it.hasNext();) {
      availableModel = (MakeModelYear)it.next();
      if(availableModel.modelYear == modelYear) {
         if(availableModel.make.equals(make)) {
            results.append("<model>");
            results.append(availableModel.model);
            results.append("</model>");
         }
      }
   }
   results.append("</models>");
   response.setContentType("text/xml");
   response.getWriter().write(results.toString());
}

protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
   processRequest(request, response);
}

public void init() throws ServletException {
   availableModels.add(new MakeModelYear(2006, "Dodge", "Charger"));
   availableModels.add(new MakeModelYear(2006, "Dodge", "Magnum"));
   availableModels.add(new MakeModelYear(2006, "Dodge", "Ram"));
   availableModels.add(new MakeModelYear(2006, "Dodge", "Viper"));
   availableModels.add(new MakeModelYear(1995, "Dodge", "Avenger"));
   availableModels.add(new MakeModelYear(1995, "Dodge", "Intrepid"));
   availableModels.add(new MakeModelYear(1995, "Dodge", "Neon"));
   availableModels.add(new MakeModelYear(1995, "Dodge", "Spirit"));
   availableModels.add(new MakeModelYear(1985, "Dodge", "Aries"));
   availableModels.add(new MakeModelYear(1985, "Dodge", "Daytona"));
   availableModels.add(new MakeModelYear(1985, "Dodge", "Diplomat"));
   availableModels.add(new MakeModelYear(1985, "Dodge", "Omni"));
   availableModels.add(new MakeModelYear(1970, "Dodge",
   "Challenger"));
   availableModels.add(new MakeModelYear(1970, "Dodge", "Charger"));
```

```
    availableModels.add(new MakeModelYear(1970, "Dodge", "Coronet"));
    availableModels.add(new MakeModelYear(1970, "Dodge", "Dart"));
    availableModels.add(new MakeModelYear(2006, "Chevrolet",
    "Colorado"));
    availableModels.add(new MakeModelYear(2006, "Chevrolet",
    "Corvette"));
    availableModels.add(new MakeModelYear(2006, "Chevrolet",
    "Equinox"));
    availableModels.add(new MakeModelYear(2006, "Chevrolet", "Monte
    Carlo"));
    availableModels.add(new MakeModelYear(1995, "Chevrolet",
    "Beretta"));
    availableModels.add(new MakeModelYear(1995, "Chevrolet",
    "Camaro"));
    availableModels.add(new MakeModelYear(1995, "Chevrolet",
    "Cavalier"));
    availableModels.add(new MakeModelYear(1995, "Chevrolet",
    "Lumina"));
    availableModels.add(new MakeModelYear(1985, "Chevrolet",
    "Cavalier"));
    availableModels.add(new MakeModelYear(1985, "Chevrolet",
    "Chevette"));
    availableModels.add(new MakeModelYear(1985, "Chevrolet",
    "Celebrity"));
    availableModels.add(new MakeModelYear(1985, "Chevrolet", "Citation
    II"));
    availableModels.add(new MakeModelYear(1970, "Chevrolet", "Bel
    Air"));
    availableModels.add(new MakeModelYear(1970, "Chevrolet",
    "Caprice"));
    availableModels.add(new MakeModelYear(1970, "Chevrolet",
    "Chevelle"));
    availableModels.add(new MakeModelYear(1970, "Chevrolet", "Monte
    Carlo"));
    availableModels.add(new MakeModelYear(2006, "Pontiac", "G6"));
    availableModels.add(new MakeModelYear(2006, "Pontiac", "Grand
    Prix"));
    availableModels.add(new MakeModelYear(2006, "Pontiac",
    "Solstice"));
    availableModels.add(new MakeModelYear(2006, "Pontiac", "Vibe"));
    availableModels.add(new MakeModelYear(1995, "Pontiac",
    "Bonneville"));
    availableModels.add(new MakeModelYear(1995, "Pontiac", "Grand
    Am"));
    availableModels.add(new MakeModelYear(1995, "Pontiac", "Grand
    Prix"));
    availableModels.add(new MakeModelYear(1995, "Pontiac",
    "Firebird"));
    availableModels.add(new MakeModelYear(1985, "Pontiac", "6000"));
    availableModels.add(new MakeModelYear(1985, "Pontiac", "Fiero"));
    availableModels.add(new MakeModelYear(1985, "Pontiac", "Grand
    Prix"));
    availableModels.add(new MakeModelYear(1985, "Pontiac",
    "Parisienne"));
    availableModels.add(new MakeModelYear(1970, "Pontiac",
    "Catalina"));
    availableModels.add(new MakeModelYear(1970, "Pontiac", "GTO"));
    availableModels.add(new MakeModelYear(1970, "Pontiac", "LeMans"));
    availableModels.add(new MakeModelYear(1970, "Pontiac",
    "Tempest"));
}
```

```
  private static class MakeModelYear {

     private int modelYear;
     private String make;
     private String model;

     public MakeModelYear(int modelYear, String make, String model) {
        this.modelYear = modelYear;
        this.make = make;
        this.model = model;
     }
      }
}
```

شکل ۶-۴ نشان می دهد که انتخاب مقادیر مختلف در جعبه های انتخاب سال و کارخانه، لیست مدلهای موجود را تغییر می دهد.

شکل ۶-۴ : انتخاب مقادیر مختلف در جعبه های انتخاب

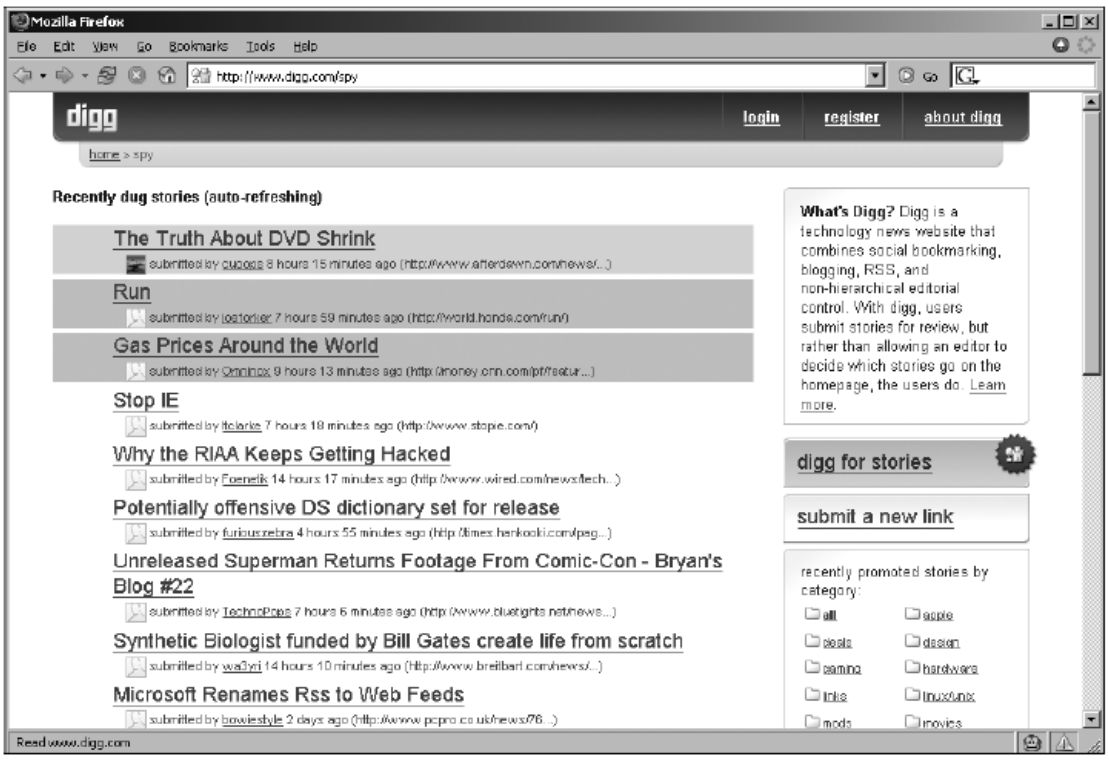


## ساخت یک صفحه با قابلیت بارگذاری خودکار[41]

هنگامی که سایتی مثل CNN.com در بازه های زمانی مختلف بارگذاری مجدد می شود، می توانید ببینید که کل صفحه مجدداً بارگذاری شده است تا فقط یک یا چند تیتر خبر و چند عکس تغییر یابند. همچنین با بارگذاری مجدد کل صفحه، تشخیص اینکه کدام مطلب به تازگی به صفحه افزوده شده است گاهی مشکل می باشد.

با استفاده از آژاکس می توانید خیال کاربران خود را در مورد فشردن دکمه بارگذاری مجدد مرورگر، جهت دریافت آخرین تغییرات، راحت نمایید.یک نمونه وب سایت که از این تکنیک استفاده می نماید وب سایت خبری جدیدی به نام Digg است.( http://digg.com/spy ) این وب سایت با استفاده از رهیافت بارگذاری مجدد خودکار و سیستم رنگ مناسب به صورت گرافیکی به کاربر نشان می دهد که کدام تیتر خبری جدید می باشد. (شکل ۷-۴ را ببینید.)

شکل ۷-۴ : مثالی از صفحه با قابلیت بارگذاری خودکار( Digg.com )




Autorefreshing - [41]

بارگذاری خودکار صفحه وب بسیار آسان می باشد. برای مثال همانطور که در کد ۷-۴ نشان داده شده است، از یک دکمه برای شروع بارگذاری مجدد استفاده می شود، البته در دنیای واقعی، این مسئله توسط متد OnLoad صفحه انجام می شود. در این مثال متد doStart() آغاز کننده عملیات است، اما قسمت جالب کار متد setTimeout در متد pollCallbacl() می باشد که اجازه می دهد متد اشاره شده در بازه های زمانی مشخص که می تواند در حد میلی ثانیه باشد، ارزیابی و اجرا گردد. متد createRow متدی است که از مزایای DOM به منظور ایجاد محتوای صفحه به صورت پویا استفاده می کندو متد refreshTime نیز مقدار زمان بارگذاری مجدد را کنترل و تنظیم می نماید.

کد ۷-۴ : dynamicUpdate.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Ajax Dynamic Update</title>
<script type="text/javascript">
var xmlHttp;
function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function doStart() {
      createXMLHttpRequest();
      var url = "DynamicUpdateServlet?task=reset";
      xmlHttp.open("GET", url, true);
      xmlHttp.onreadystatechange = startCallback;
      xmlHttp.send(null);
}

function startCallback() {
      if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
            setTimeout("pollServer()", 5000);
            refreshTime();
            }
      }
}

function pollServer() {
      createXMLHttpRequest();
      var url = "DynamicUpdateServlet?task=foo";
      xmlHttp.open("GET", url, true);
      xmlHttp.onreadystatechange = pollCallback;
      xmlHttp.send(null);
}
```

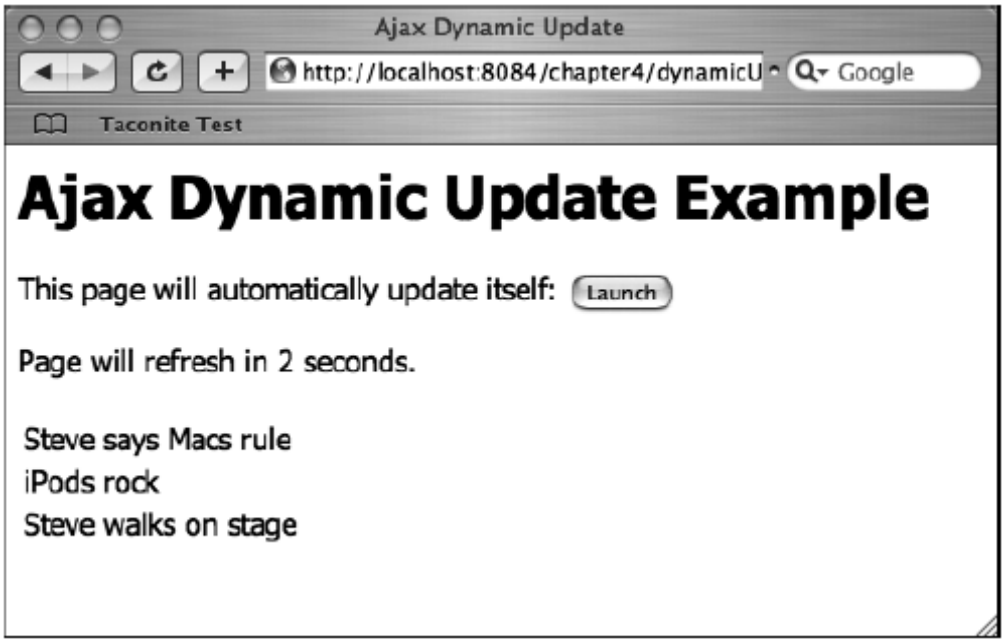
```
function refreshTime(){
      var time_span = document.getElementById("time");
      var time_val = time_span.innerHTML;
      var int_val = parseInt(time_val);
      var new_int_val = int_val - 1;
      if (new_int_val > -1) {
            setTimeout("refreshTime()", 1000);
            time_span.innerHTML = new_int_val;
      } else {
            time_span.innerHTML = 5;
      }
}

function pollCallback() {
      if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
            var message =xmlHttp.responseXML
            .getElementsByTagName("message")[0].firstChild.data;
            if (message != "done") {
                  var new_row = createRow(message);
                  var table =
                  document.getElementById("dynamicUpdateArea");
                  var table_body =
                  table.getElementsByTagName("tbody").item(0);
                  var first_row =
                  table_body.getElementsByTagName("tr").item(1);
                  table_body.insertBefore(new_row, first_row);
                  setTimeout("pollServer()", 5000);
                  refreshTime();
            }
            }
      }
}

function createRow(message) {
      var row = document.createElement("tr");
      var cell = document.createElement("td");
      var cell_data = document.createTextNode(message);
      cell.appendChild(cell_data);
      row.appendChild(cell);
      return row;
}
</script>
</head>
<body>
<h1>Ajax Dynamic Update Example</h1>
This page will automatically update itself:
<input type="button" value="Launch" id="go" onclick="doStart();"/>
<p>
Page will refresh in <span id="time">5</span> seconds.
<p>
<table id="dynamicUpdateArea" align="left">
<tbody>
<tr id="row0"><td></td></tr>
</tbody>
</table>
</body>
</html>
```

کد سمت سرور نیز بسیار ساده می باشد و فقط دسته ای از اطلاعات را براساس یک شمارنده ساده به مرورگر برمی گرداند. در واقع در این قسمت با استفاده از عملیات مربوط به پایگاه داده، کارهای مختلفی انجام داد که در این مثال به منظور سادگی و تمرکز روی بحث از آوردن آنها خودداری شده است.

کد ۸-۴ : DynamicUpdateServlet.java

```
package ajaxbook.chap4;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DynamicUpdateServlet extends HttpServlet {
private int counter = 1;
/** Handles the HTTP <code>GET</code> method.
* @param request servlet request
* @param response servlet response
*/
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      String res = "";
      String task = request.getParameter("task");
      String message = "";
      if (task.equals("reset")) {
            counter = 1;
      } else {
            switch (counter) {
            case 1: message = "Steve walks on stage"; break;
            case 2: message = "iPods rock"; break;
            case 3: message = "Steve says Macs rule"; break;
            case 4: message = "Change is coming"; break;
            case 5: message = "Yes, OS X runs on Intel - has for
            years"; break;
            case 6: message = "Macs will soon have Intel chips";
            break;
            case 7: message = "done"; break;
            }
      counter++;
      }
      res = "<message>" + message + "</message>";
      PrintWriter out = response.getWriter();
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      out.println("<response>");
      out.println(res);
      out.println("</response>");
      out.close();
}
}
```

شکل ۹-۴ نتیجه مثال بارگذاری خودکار را در مرورگر نشان می دهد.




شکل ۹-۴ : به روزرسانی به صورت پویا

## نمایش نوار پیشرفت [۴۲]

در هر برنامه کاربردی ممکن است تراکنشی وجود داشته باشد که زمانی جهت تکمیل آن لازم باشد و این زمان از یک کامپیوتر به کامپیوتر دیگر به خاطر تفاوتهای سخت افزاری، ممکن است متفاوت باشد. در اینگونه مواقع معمولاً با استفاده از نوار پیشرفت مدت زمان طی شده و باقیمانده، از آن عمل به کاربر نشان داده می شود. قبل از ظهور آژاکس این گزینه در مورد وب سایتها به راحتی قابل پیاده سازی نبود. در این مثال می آموزید که چگونه ار آژاکس به منظور ایجاد یک نوار پیشرفت استفاده کنید.

دراین مثال که کد سمت کاربر آن با شماره ۹-۴ ارائه شده است، از تابع setTimeout در متد pollCallback() به منظور فراخوانی ممتدسرور، هر ۲ ثانیه، استفاده شده است. در متد processResult با استفاده از عددی که نشان دهنده درصد تکمیل شده می باشد، مشخص می شود که کدام یک از خانه های نوار پیشرفت باید رنگ شود.

کد ۹-۴ : progressBar.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Ajax Progress Bar</title>
<script type="text/javascript">
	var xmlHttp;
	var key;
	var bar_color = 'gray';
	var span_id = "block";
	var clear = "   "
	function createXMLHttpRequest() {
		if (window.ActiveXObject) {
			xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
		}
		else if (window.XMLHttpRequest) {
			xmlHttp = new XMLHttpRequest();
		}
}

function go() {
	createXMLHttpRequest();
	checkDiv();
	var url = "ProgressBarServlet?task=create";
	var button = document.getElementById("go");
	button.disabled = true;
	xmlHttp.open("GET", url, true);
	xmlHttp.onreadystatechange = goCallback;
	xmlHttp.send(null);
}
```

[42] - progress bar

```
function goCallback() {
      if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
                  setTimeout("pollServer()", 2000);
            }
      }
}

function pollServer() {
      createXMLHttpRequest();
      var url = "ProgressBarServlet?task=poll&key=" + key;
      xmlHttp.open("GET", url, true);
      xmlHttp.onreadystatechange = pollCallback;
      xmlHttp.send(null);
}

function pollCallback() {
      if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
                  var percent_complete =xmlHttp.responseXML
                  .getElementsByTagName("percent")[0].firstChild.data
            ;
            var index = processResult(percent_complete);
            for (var i = 1; i <= index; i++) {
                  var elem = document.getElementById("block" + i);
                  elem.innerHTML = clear;
                  elem.style.backgroundColor = bar_color;
                  var next_cell = i + 1;
                  if (next_cell > index && next_cell <= 9) {
                        document.getElementById("block" + next_cell)
                        .innerHTML =percent_complete + "%";
                  }
            }
            if (index < 9) {
                  setTimeout("pollServer()", 2000);
            } else {
                  document.getElementById("complete").innerHTML =
                  "Complete!";
                  document.getElementById("go").disabled = false;
            }
            }
      }
}

function processResult(percent_complete) {
      var ind;
      if (percent_complete.length == 1) {
      ind = 1;
      } else if (percent_complete.length == 2) {
            ind = percent_complete.substring(0, 1);
            } else {
                  ind = 9;
            }
      return ind;
}

function checkDiv() {
      var progress_bar = document.getElementById("progressBar");
      if (progress_bar.style.visibility == "visible") {
            clearBar();
            document.getElementById("complete").innerHTML = "";
```

```
      } else {
      progress_bar.style.visibility = "visible"
      }
}

function clearBar() {
      for (var i = 1; i < 10; i++) {
            var elem = document.getElementById("block" + i);
            elem.innerHTML = clear;
            elem.style.backgroundColor = "white";
      }
}
</script>
</head>
<body>
<h1>Ajax Progress Bar Example</h1>
Launch long-running process:
<input type="button" value="Launch" id="go" onclick="go();"/>
<p>
<table align="center">
<tbody>
<tr><td>
<div id="progressBar"
style="padding:2px;border:solid black 2px;visibility:hidden">
<span id="block1">   </span>
<span id="block2">   </span>
<span id="block3">   </span>
<span id="block4">   </span>
<span id="block5">   </span>
<span id="block6">   </span>
<span id="block7">   </span>
<span id="block8">   </span>
<span id="block9">   </span>
</div>
</td></tr>
<tr><td align="center" id="complete"></td></tr>
</tbody>
</table>
</body>
</html>
```

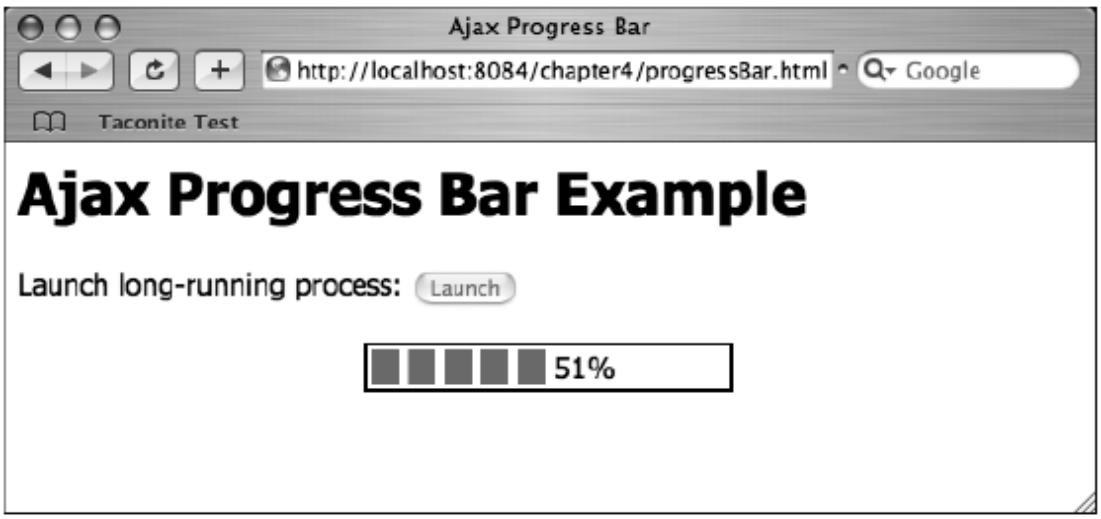
**کد سمت سرور این مثال نقش یک تراکنش طولانی را ایفا می کند.(کد ۱۰-۴ را ببینید.)**

**کد ۱۰-۴ :** progressBarServlet.java

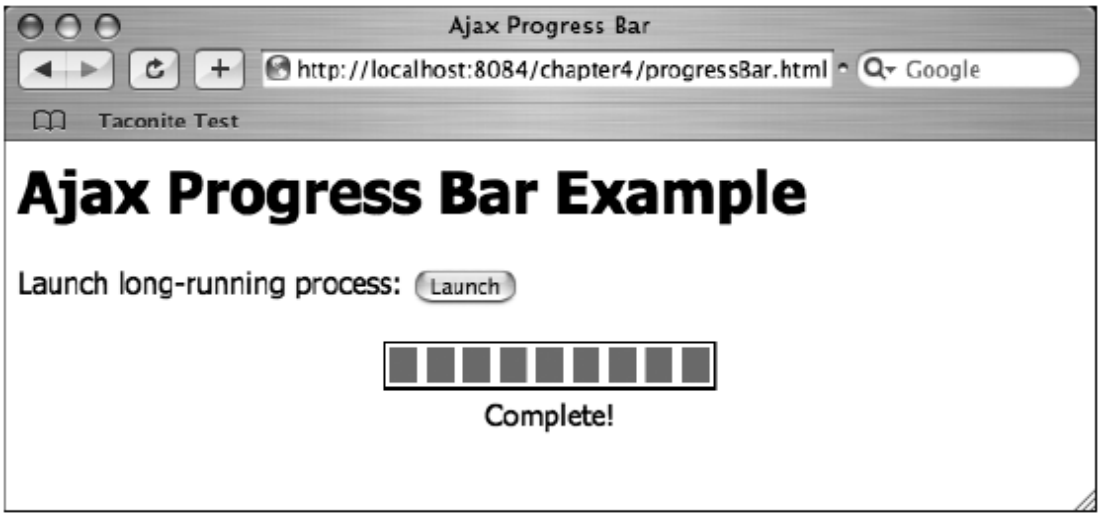
```
package ajaxbook.chap4;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ProgressBarServlet extends HttpServlet {
private int counter = 1;
/** Handles the HTTP <code>GET</code> method.
* @param request servlet request
* @param response servlet response
*/
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
```

```
throws ServletException, IOException {
      String task = request.getParameter("task");
      String res = "";
      if (task.equals("create")) {
            res = "<key>1</key>";
            counter = 1;
      }
      else {
            String percent = "";
            switch (counter) {
                  case 1: percent = "10"; break;
                  case 2: percent = "23"; break;
                  case 3: percent = "35"; break;
                  case 4: percent = "51"; break;
                  case 5: percent = "64"; break;
                  case 6: percent = "73"; break;
                  case 7: percent = "89"; break;
                  case 8: percent = "100"; break;
            }
            counter++;
            res = "<percent>" + percent + "</percent>";
      }
      PrintWriter out = response.getWriter();
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      out.println("<response>");
      out.println(res);
      out.println("</response>");
      out.close();
      }
}
```

**شکل ۱۰-۴ نوار پیشرفت را درحال تکمیل شدن و شکل ۱۱-۴ آن را در وضعیت کامل نـشان می دهد.**




**شکل ۱۰-۴: نوار پیشرفت در حال تکمیل**




**شکل ۱۱-۴ : نوار پیشرفت در حالت کامل**

## ساخت Tooltip

یکی از جالبترین کاربردهای آژاکس، کاربردی است که سایت Netflix به منظور فروش DVD های خود استفاده می کند. وقتی که وارد این وب سایت می شوید، اطلاعاتی در مورد جدیدترین فیلم ها به صورت عکس و متن مشاهده می کنید، اما وقتی که با مکان نمای ماوس روی یکی از شکلهای مربوط به فیلم ها اشاره می کنید، می توانید اطلاعات بیشتری را در مورد آن مشاهده کنید.(شکل ۱۲-۴ را ببینید.) در صورتیکه از تکنیک آژاکس در این زمینه استفاده نشود، پس از بارگذاری مجدد صفحه اطلاعاتی که ممکن است مفید واقع نشوند، نمایش داده شوند. در حقیقت با استفاده از آژاکس اطلاعات مورد نیاز هرگاه که به آنها نیاز شود به مرورگر کاربر ارسال می شود.


شکل ۱۲-۴: نمونه ای از نمایش tooltip

در طی مثال بعد به شما نشان می دهیم که چگونه می توانید از این روش استفاده کنید. کد سمت کلاینت بسیار ساده و روشن است.(کد ۱۱-۴ را ببینید.) نکات جالب این کد در متد calculateOffset قرار دارند. این مثال مقداری اطلاعات را درون یک جدول نمایش می دهد و هرگاه که کاربر با ماوس روی ستونهای جدول اشاره کند، اطلاعات بیشتر به وی نشان داده می شوند. این مثال در نهایت سادگی ارئه شده است تا به اصل تکنیک توجه شود نه به کدهای تولید گرافیک زیبا.

کد ۱۱-۴ : toolTip.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Ajax Tool Tip</title>
<script type="text/javascript">
	var xmlHttp;
	var dataDiv;
	var dataTable;
	var dataTableBody;
	var offsetEl;
function createXMLHttpRequest() {
	if (window.ActiveXObject) {
	xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
	}
	else if (window.XMLHttpRequest) {
		xmlHttp = new XMLHttpRequest();
	}
}

function initVars() {
	dataTableBody = document.getElementById("courseDataBody");
	dataTable = document.getElementById("courseData");
	dataDiv = document.getElementById("popup");
}

function getCourseData(element) {
	initVars();
	createXMLHttpRequest();
	offsetEl = element;
	var url = "ToolTipServlet?key=" + escape(element.id);
	xmlHttp.open("GET", url, true);
	xmlHttp.onreadystatechange = callback;
	xmlHttp.send(null);
}

function callback() {
	if (xmlHttp.readyState == 4) {
		if (xmlHttp.status == 200) {
			setData(xmlHttp.responseXML);
		}
	}
}
```

```
function setData(courseData) {
      clearData();
      setOffsets();
      var length =
      courseData.getElementsByTagName("length")[0].firstChild.data;
      var par =
      courseData.getElementsByTagName("par")[0].firstChild.data;
      var row, row2;
      var parData = "Par: " + par
      var lengthData = "Length: " + length;
      row = createRow(parData);
      row2 = createRow(lengthData);
      dataTableBody.appendChild(row);
      dataTableBody.appendChild(row2);
}

function createRow(data) {
      var row, cell, txtNode;
      row = document.createElement("tr");
      cell = document.createElement("td");
      cell.setAttribute("bgcolor", "#FFFAFA");
      cell.setAttribute("border", "0");
      txtNode = document.createTextNode(data);
      cell.appendChild(txtNode);
      row.appendChild(cell);
      return row;
}

function setOffsets() {
      var end = offsetEl.offsetWidth;
      var top = calculateOffsetTop(offsetEl);
      dataDiv.style.border = "black 1px solid";
      dataDiv.style.left = end + 15 + "px";
      dataDiv.style.top = top + "px";
}

function calculateOffsetTop(field) {
      return calculateOffset(field, "offsetTop");
}

function calculateOffset(field, attr) {
      var offset = 0;
      while(field) {
            offset += field[attr];
            field = field.offsetParent;
      }
      return offset;
}

function clearData() {
      var ind = dataTableBody.childNodes.length;
      for (var i = ind - 1; i >= 0 ; i--) {
            dataTableBody.removeChild(dataTableBody.childNodes[i]);
      }
      dataDiv.style.border = "none";
}
</script>
</head>
<body>
<h1>Ajax Tool Tip Example</h1>
<h3>Golf Courses</h3>
```

```
<table id="courses" bgcolor="#FFFAFA" border="1"
cellspacing="0" cellpadding="2"/>
<tbody>
<tr><td id="1" onmouseover="getCourseData(this);"
onmouseout="clearData();">Augusta National</td></tr>
<tr><td id="2" onmouseover="getCourseData(this);"
onmouseout="clearData();">Pinehurst No. 2</td></tr>
<tr><td id="3" onmouseover="getCourseData(this);"
onmouseout="clearData();">
St. Andrews Links</td></tr>
<tr><td id="4" onmouseover="getCourseData(this);"
onmouseout="clearData();">Baltusrol Golf Club</td></tr>
</tbody>
</table>
<div style="position:absolute;" id="popup">
<table id="courseData" bgcolor="#FFFAFA" border="0"
cellspacing="2" cellpadding="2"/>
<tbody id="courseDataBody"></tbody>
</table>
</div>
</body>
</html>
```

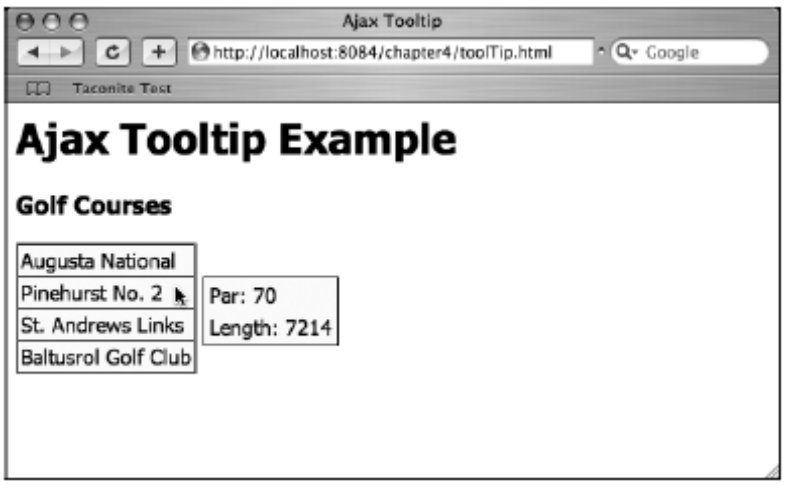
**به یاد داشته باشید که هموار اطلاعات تکمیلی را در سمت سرور از منابعی مثـل پایگـاه داده، بازیابی کنید. کد ۱۲-۴، کد سمت سرور این مثال را نشان می دهد.**

**کد ۱۲-۴ :** toolTipServlet.java

```
package ajaxbook.chap4;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.*;
import javax.servlet.http.*;

public class ToolTipServlet extends HttpServlet {
      private Map courses = new HashMap();
      public void init(ServletConfig config) throws ServletException
      {
            CourseData augusta = new CourseData(72, 7290);
            CourseData pinehurst = new CourseData(70, 7214);
            CourseData standrews = new CourseData(72, 6566);
            CourseData baltusrol = new CourseData(70, 7392);
            courses.put(new Integer(1), augusta);
            courses.put(new Integer(2), pinehurst);
            courses.put(new Integer(3), standrews);
            courses.put(new Integer(4), baltusrol);
      }
/** Handles the HTTP <code>GET</code> method.
* @param request servlet request
* @param response servlet response
*/
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      Integer key = Integer.valueOf(request.getParameter("key"));
      CourseData data = (CourseData) courses.get(key);
```

```
        PrintWriter out = response.getWriter();
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        out.println("<response>");
        out.println("<par>" + data.getPar() + "</par>");
        out.println("<length>" + data.getLength() + "</length>");
        out.println("</response>");
        out.close();
}

private class CourseData {
        private int par;
        private int length;
        public CourseData(int par, int length) {
                this.par = par;
                this.length = length;
        }
        public int getPar() {
                return this.par;
        }
        public int getLength() {
                return this.length;
        }
}
}
```

**شکل ۱۳-۴ نتیجه اجرای این مثال را نشان می دهد.**




**شکل ۱۳-۴ : نتیجه اجرای مثال**

## به روزرسانی صفحه وب به صورت پویا

همانگونه که قبلاً گفته شد، زمانی که تنها قسمتی از صفحه نیاز به عمل به روزرسانی دارد، آژاکس بهترین تکنیک کارساز است. به عبارت دیگر، مواردی که در آنها تمام صفحه به خاطر تغییر در محتوای قسمتی از آن مجدداً بارگذاری می شوند، گزینه های مناسبی جهت استفاده از تکنیک آژاکس هستند تا دیگر نیاز به بارگذاری کل صفحه نباشد.

تصور کنید در یک صفحه وب، کاربر می خواهد اطلاعات جدیدی را به منظور افزودن به یک لیست وارد کند، درحالت عادی پس از هر ورود اطلاعات، نیاز به بارگذاری مجدد کل صفحه می باشد. در مثال آتی، شما صفحه ای را می بینید که لیستی از کارمندان یک سازمان را نشان می دهد. در بالای این صفحه سه مؤلفه ورود اطلاعات ( Input Box ) قرار دارد که مقادیر نام، عنوان و بخش مربوط به کارمند جدید را می پذیرند. فشردن دکمه Add مقادیر وارد شده را به سرور ارسال می کند تا اطلاعات کارمند جدید به پایگاه داده مربوطه افزوده شود.

با استفاده از تکنیکهای معمول وب، کد سمت سرور با استفاده از بارگذاری مجدد کل صفحه که تنها تغییر آن با صفحه قبل افزوده شدن اطلاعات کارمند جدید به لیست می باشد، به مرورگر پاسخ می دهد. در این مثال شما با استفاده از آژاکس اطلاعات را به سرور ارسال می کنید تا به پایگاه داده افزوده شوند. سرور نیز با ارسال یک کد وضعیت که نشان می دهد عملیات سمت سرور موفقیت آمیز انجام شده است یا خیر، به مرورگر پاسخ می دهد. در صورت موفقیت آمیز بودن عملیات، مرورگر با استفاده از متدهای DOM محتوای صفحه را به صورت پویا، به منظور نمایش رکورد مربوط به کارمند تازه افزوده شده، تغییر می دهد. این مثال همچنین یک دکمه حذف نیز به ازای هر رکورد جدید ایجاد می کند تا در صورت لزوم رکورد مربوطه از پایگاه داده و صفحه وب حذف گردد.

کد ۱۳-۴، کد سمت کلاینت این مثال را نشان می دهد. صفحه شامل دو قسمت است؛ قسمت اول شامل سه مؤلفه ورود داده به منظور ورود نام، عنوان و بخش کارمند جدید و یک دکمه که عملیات افزودن رکورد جدید به پایگاه داده را انجام می دهد. قسمت دوم لیستی از همه کارمندان موجود در پایگاه داده را نمایش می دهد که هر رکورد آن دارای یک دکه حذف می باشد تا اطلاعات آن رکورد از پایگاه داده قابل حذف باشد.

کد ۱۳-۴ : employeeList.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Employee List</title>
<script type="text/javascript">
      var xmlHttp;
      var name;
      var title;
      var department;
      var deleteID;
      var EMP_PREFIX = "emp-";

function createXMLHttpRequest() {
      if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      else if (window.XMLHttpRequest) {
            xmlHttp = new XMLHttpRequest();
      }
}

function addEmployee() {
      name = document.getElementById("name").value;
      title = document.getElementById("title").value;
      department = document.getElementById("dept").value;
      action = "add";
      if(name == "" || title == "" || department == "") {
            return;
      }
      var url = "EmployeeList?"
      + createAddQueryString(name, title, department, "add")
      + "&ts=" + new Date().getTime();
      createXMLHttpRequest();
      xmlHttp.onreadystatechange = handleAddStateChange;
      xmlHttp.open("GET", url, true);
      xmlHttp.send(null);
}

function createAddQueryString(name, title, department, action) {
      var queryString = "name=" + name
      + "&title=" + title
      + "&department=" + department
      + "&action=" + action;
      return queryString;
}

function handleAddStateChange() {
      if(xmlHttp.readyState == 4) {
            if(xmlHttp.status == 200) {
                  updateEmployeeList();
                  clearInputBoxes();
            }
            else {
                  alert("Error while adding employee.");
            }
      }
}
```

```
function clearInputBoxes() {
	document.getElementById("name").value = "";
	document.getElementById("title").value = "";
	document.getElementById("dept").value = "";
}

function deleteEmployee(id) {
	deleteID = id;
	var url = "EmployeeList?"
	+ "action=delete"
	+ "&id=" + id
	+ "&ts=" + new Date().getTime();
	createXMLHttpRequest();
	xmlHttp.onreadystatechange = handleDeleteStateChange;
	xmlHttp.open("GET", url, true);
	xmlHttp.send(null);
}

function updateEmployeeList() {
	var responseXML = xmlHttp.responseXML;
	var status = responseXML.getElementsByTagName("status")
		.item(0).firstChild.nodeValue;
	status = parseInt(status);
	if(status != 1) {
		return;
	}
	var row = document.createElement("tr");
	var uniqueID = responseXML.getElementsByTagName("uniqueID")[0]
	.firstChild.nodeValue;
	row.setAttribute("id", EMP_PREFIX + uniqueID);
	row.appendChild(createCellWithText(name));
	row.appendChild(createCellWithText(title));
	row.appendChild(createCellWithText(department));
	var deleteButton = document.createElement("input");
	deleteButton.setAttribute("type", "button");
	deleteButton.setAttribute("value", "Delete");
	deleteButton.onclick = function () { deleteEmployee(uniqueID);
	};
	cell = document.createElement("td");
	cell.appendChild(deleteButton);
	row.appendChild(cell);
	document.getElementById("employeeList").appendChild(row);
	updateEmployeeListVisibility();
}

function createCellWithText(text) {
	var cell = document.createElement("td");
	cell.appendChild(document.createTextNode(text));
	return cell;
}

function handleDeleteStateChange() {
	if(xmlHttp.readyState == 4) {
		if(xmlHttp.status == 200) {
			deleteEmployeeFromList();
		}
		else {
			alert("Error while deleting employee.");
		}
	}
}
```

```
function deleteEmployeeFromList() {
      var status =
      xmlHttp.responseXML.getElementsByTagName("status")
            .item(0).firstChild.nodeValue;
      status = parseInt(status);
      if(status != 1) {
            return;
      }

      var rowToDelete = document.getElementById(EMP_PREFIX +
      deleteID);
      var employeeList = document.getElementById("employeeList");
      employeeList.removeChild(rowToDelete);
      updateEmployeeListVisibility();
}

function updateEmployeeListVisibility() {
      var employeeList = document.getElementById("employeeList");
      if(employeeList.childNodes.length > 0) {
            document.getElementById("employeeListSpan").style.display
            = "";
      }
      else {
            document.getElementById("employeeListSpan").style.display
            = "none";
      }
}
</script>
</head>
<body>
<h1>Employee List</h1>
<form action="#">
<table width="80%" border="0">
<tr>
<td>Name: <input type="text" id="name"/></td>
<td>Title: <input type="text" id="title"/></td>
<td>Department: <input type="text" id="dept"/></td>
</tr>
<tr>
<td colspan="3" align="center">
<input type="button" value="Add" onclick="addEmployee();"/>
</td>
</tr>
</table>
</form>
<span id="employeeListSpan" style="display:none;">
<h2>Employees:</h2>
<table border="1" width="80%">
<tbody id="employeeList"></tbody>
</table>
</span>
</body>
</html>
```

فشردن دکمه Add عملیلا افزودن به پایگاه داده را انجـام مـی دهـد. متـد addEmployee توسط رویداد onClick مربوط به دکمه فراخوانی می شـود. متـد اخیـر بـا اسـتفاده از متـد

createAddQueryString، رشته درخواست را که شامل نام، عنوان و بخش کارمند جدید می باشد، به منظور ارسال به سرور، تولید می نماید. بعد از ساخت و تنظیم مشخصات شئ XMLHttpRequest ، درخواست آماده شده به سرور ارسال می شود.

کد ۱۴-۴ کد سمت سرور را نشان می دهد که درخواست رسیده را پردازش می کند. با دریافت درخواست، متد doGet فراخوانی می شود. این متد پارامترهای موجود در رشته درخواست را بازیابی می کند و سپس متد addEmployee فراخوانی می شود.

کد ۱۴-۴ : EmployeeListServlet.java

```
package ajaxbook.chap4;
import java.io.*;
import java.net.*;
import java.util.Random;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmployeeListServlet extends HttpServlet {
protected void addEmployee(HttpServletRequest request
, HttpServletResponse response)
throws ServletException, IOException {
      //Store the object in the database
      String uniqueID = storeEmployee();
      //Create the response XML
      StringBuffer xml = new StringBuffer("<result><uniqueID>");
      xml.append(uniqueID);
      xml.append("</uniqueID>");
      xml.append("</result>");
      //Send the response back to the browser
      sendResponse(response, xml.toString());
}

protected void deleteEmployee(HttpServletRequest request
, HttpServletResponse response)
throws ServletException, IOException {
      String id = request.getParameter("id");
      /* Assume that a call is made to delete the employee from the
      database */
      //Create the response XML
      StringBuffer xml = new StringBuffer("<result>");
      xml.append("<status>1</status>");
      xml.append("</result>");
      //Send the response back to the browser
      sendResponse(response, xml.toString());
}

protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      String action = request.getParameter("action");
      if(action.equals("add")) {
            addEmployee(request, response);
      }
```

```
      else if(action.equals("delete")) {
            deleteEmployee(request, response);
      }
}

private String storeEmployee() {
      /* Assume that the employee is saved to a database and the
      * database creates a unique ID. Return the unique ID to the
      * calling method. In this case, make up a unique ID.
      */
      String uniqueID = "";
      Random randomizer = new Random(System.currentTimeMillis());
      for(int i = 0; i < 8; i++) {
            uniqueID += randomizer.nextInt(9);
      }
      return uniqueID;
      }
private void sendResponse(HttpServletResponse response, String
responseText)
throws IOException {
      response.setContentType("text/xml");
      response.getWriter().write(responseText);
}
}
```

تابع addEmployee مسئول انجام عملیات افزودن رکورد به پایگاه داده و تولید پاسخ سرور می باشد. این تابع با استفاده از تابع کمکی storeEmployee عمل ثبت اطلاعات در پایگاه داده را انجام می دهد. در پیاده سازی های واقعی این تابع کمکی از سرویسهای پایگاه داده رابطه ای به منظور ثبت اطلاعات استفاده می کند در صورتیکه در این مثال ساده، این تابع فقط نقش افزودن رکورد به پایگاه داده را بازی می کند و همانند کار با پایگاه داده واقعی به صورت تصادفی یک شناسه یکتا[۴۳] برای رکورد جدید تولید می کند و آن را به عنوان خروجی به تابع addEmployee برمی گرداند.

با فرض موفقیت آمیز بودن عملیلات ثبت، متد addEmployee به منظور تولید پاسخ کار خود را ادامه می دهد. پاسخ شامل یک رشته کوتاه XML است که کد وضعیت را به مرورگر بر می گرداند. رشته XML از طریق الحاق رشته های کوچکتر کاراکتری ساخته می شود و سپس به عنوان پاسخ به مرورگر فرستاده می شود.

مرورگر پاسخ سرور را با استفاده از فراخوانی متد handleAddStateChange پردازش می کند. این متد هرگاه که تغییری در حالت شئ XMLHttpRequest رخ دهد، فراخوانی می شود. هنگامی که خاصیت readyState این شئ نشان دهد که عملیات سمت سرور موفقیت آمیز بوده است روال UpdateEmployeeList بعد از روال cleareInputBoxes

[43] - ID

فراخوانی می شود. روال updateEmployeeList مسئول نمایش مشخصات کارمند جدیدی است که با موفقیت به پایگاه داده افزوده شده است و باید در لیست موجود در صفحه نیز نمایش داده شود. روال cleareInputBoxes مؤلفه های ورود اطلاعات بالای صفحه را خالی می کند تا برای ورود اطلاعات بعدی آماده باشند. در ادامه نحوه عملکرد روال UpdateEmployeeList را توضیح می دهیم. این روال یک سطر خالی به انتهای جدولی که نمایش اطلاعات کارمندان را برعهده دارد، می افزاید، این روال کار خود را با استفاده از متد document.createElement به این منظور شروع می کند. صفت id سطر جدید به مقدار شناسه یکتایی که سرور به مرورگر برگردانده است، تنظیم می شود. این صفت کمک می کند تا اگر دکمه حذف فشرده شد، رکورد مربوطه به راحتی حذف شود. این روال از تابع کمکی به نام createCellWithText به منظور ایجاد ستون با متن مشخص، استفاده می کند. تابع اخیر ستونهایی را برای نام، عنوان و بخش براساس اطلاعاتی که کاربر وارد کرده است، ایجاد می نماید. سپس ستون ها به سطری که که قبلاً ساخته شده بود، افزوده می شوند. آخرین موردی که باید ایجاد شود دکمه حذف و ستونی است که آن را در بر می گیرد. برای این منظور از متد document.createElement به منظور ایجاد یک مؤلفه Input که نوع و مقدار آن به ترتیب Button و Delete تنظیم شده اند، استفاده می شود. سپس باید یک ستون به منظور افزودن این دکمه به آن، به سطر جدید افزوده شود و دکمه به عنوان فرزند این ستون قرار گیرد. در انتها سطری را که اکنون حاوی نام، عنوان، بخش کارمند و دکمه حذف می باشد به جدول کارمندان افزوده می شود.

عملیات حذف یک کارمند نیز تقریباً شبیه عملیات درج آن می باشد. رویداد onClick مربوط به دکمه حذف، روال deleteEmployee را فراخوانی می کند که ورودی آن شناسه یکتای آن کارمند می باشد. یک رشته درخواست ساده حاوی عمل مورد نظر(حذف) و شناسه کارمندی است که باید حذف شود، ساخته می شود. بعد از تنظیم خاصیت onReadyStateChange شیئ XMLHttpRequest به نام تابع مربوطه، درخواست به سرور ارسال می شود. در سمت سرور از متد deleteEmployee به منظور حذف کارمند استفاده می شود. در این مثال این کد بسیار ساده شده است، فرض براین است که یک تابع کمکی عملیات مربوط به پایگاه داده را انجام می دهد و براساس نتیجه این تابع یک رشته XML به عنوان پاسخ تولید شده و به مرورگر برگردانده می شود. همانند قسمت افزودن رکورد جدید، این متد نیز یک کد وضعیت به عنوان پاسخ برمی گرداند. مرورگر پاسخ سرور

را با استفاده از تابع handleDeleteStateChange پردازش می کند که در صورت موفقیت آمیز بودن این عمل در سمت سرور، روال deleteEmployeeFromList فراخوانی می شود. تابع اخیر کد وضعیت پاسخ سرور را از درون رشته XML دریافت شده، بازیابی می کند و در صورتیکه این کد نشان دهنده عدم موفقیت سرور در عمل خواسته شده، باشد بدون انجام هیچ عملی پایان می یابد. با فرض موفقیت آمیز بودن عمل سرور، این تابع با استفاده از متد document.getElementById سطری را که باید حذف شود، مشخص می نماید. سپس سطر مربوطه با استفاده از متد removeChild از بدنه جدول حذف می شود. شکل ۱۴-۴ به روزرسانی پویای صفحه این مثال را نشان می دهد.


شکل ۱۴-۴ : به روزرسانی

## قابلیت AutoComplete

یکی از مواردی که برای پیاده سازی با آژاکس، زیاد مورد استقبال است، ساخت مؤلفه ورودی متن با قابلیت کامل شدن خودکار است. این قابلیت سرعت ورود اطلاعات توسط کاربر را سریعتر، ساده تر و کم خطاتر می نماید. البته این قابلیت در برنامه های کاربردی خارج از دنیای وب، از مدتها قبل وجود دارد اما در وب سایتها این مورد وجود نداشت تا اینکه گوگل از این قابلیت در نسخه بتای سرویس جدید خود به نام Google Suggest از آن استفاده کرد.(شکل ۱۶-۴ را ببینید.) با ارائه تعدادی نتیجه مشابه عبارتی که کاربر وارد کرده است، کاربر احساس بهتری در مورد اینکه باید دنبال چه گزینه ای بگردد، پیدا خواهد کرد.


شکل ۱۶-۴ : Google Suggest

مثالی را که در ادامه آورده ایم، تمام قابلیت های سرویس گوگل را ندارد اما به سادگی به شما نشان می دهد که چگونه می توانید این قابلیت را به کارگیرید. توجه داشته باشید که در این مثال در تابع callback برخلاف مثالهای قبل کد وضعیت سرور به صورت ۲۰۴ چک می شود نه حالت معمولی ۲۰۰. کد ۲۰۴ نشان می دهد که اطلاعاتی روی سرور در زمینه آنچه کاربر

وارد کرده است وجود ندارد و با توجه به این مطلب می توان لیست انتخاب را برای کاربر، پر کرد.

کد ۱۷-۴ : autoComplete.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Ajax Auto Complete</title>
<style type="text/css">
.mouseOut {
	background: #708090;
	color: #FFFAFA;
}
.mouseOver {
	background: #FFFAFA;
	color: #000000;
}
</style>
<script type="text/javascript">
	var xmlHttp;
	var completeDiv;
	var inputField;
	var nameTable;
	var nameTableBody;

function createXMLHttpRequest() {
	if (window.ActiveXObject) {
		xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
	}
	else if (window.XMLHttpRequest) {
		xmlHttp = new XMLHttpRequest();
	}
}

function initVars() {
	inputField = document.getElementById("names");
	nameTable = document.getElementById("name_table");
	completeDiv = document.getElementById("popup");
	nameTableBody = document.getElementById("name_table_body");
}

function findNames() {
	initVars();
	if (inputField.value.length > 0) {
		createXMLHttpRequest();
		var url = "AutoCompleteServlet?names=" +
		escape(inputField.value);
		xmlHttp.open("GET", url, true);
		xmlHttp.onreadystatechange = callback;
		xmlHttp.send(null);
	} else {
		clearNames();
	}
}

function callback() {
	if (xmlHttp.readyState == 4) {
	if (xmlHttp.status == 200) {
		var name =xmlHttp.responseXML
```

```
            .getElementsByTagName("name")[0].firstChild.data;
            setNames(xmlHttp.responseXML.getElementsByTagName("name")
            );
      } else if (xmlHttp.status == 204){
                  clearNames();
            }
      }
}

function setNames(the_names) {
      clearNames();
      var size = the_names.length;
      setOffsets();
      var row, cell, txtNode;
      for (var i = 0; i < size; i++) {
            var nextNode = the_names[i].firstChild.data;
            row = document.createElement("tr");
            cell = document.createElement("td");
            cell.onmouseout = function()
            {this.className='mouseOver';};
            cell.onmouseover = function()
            {this.className='mouseOut';};
            cell.setAttribute("bgcolor", "#FFFAFA");
            cell.setAttribute("border", "0");
            cell.onclick = function() { populateName(this); } ;
            txtNode = document.createTextNode(nextNode);
            cell.appendChild(txtNode);
            row.appendChild(cell);
            nameTableBody.appendChild(row);
      }
}

function setOffsets() {
      var end = inputField.offsetWidth;
      var left = calculateOffsetLeft(inputField);
      var top = calculateOffsetTop(inputField) +
      inputField.offsetHeight;
      completeDiv.style.border = "black 1px solid";
      completeDiv.style.left = left + "px";
      completeDiv.style.top = top + "px";
      nameTable.style.width = end + "px";
}

function calculateOffsetLeft(field) {
      return calculateOffset(field, "offsetLeft");
}

function calculateOffsetTop(field) {
      return calculateOffset(field, "offsetTop");
}

function calculateOffset(field, attr) {
      var offset = 0;
      while(field) {
            offset += field[attr];
            field = field.offsetParent;
      }
      return offset;
}

function populateName(cell) {
```

```
      inputField.value = cell.firstChild.nodeValue;
      clearNames();
}

function clearNames() {
      var ind = nameTableBody.childNodes.length;
      for (var i = ind - 1; i >= 0 ; i--) {
            nameTableBody.removeChild(nameTableBody.childNodes[i]);
      }
      completeDiv.style.border = "none";
}
</script>
</head>
<body>
<h1>Ajax Auto Complete Example</h1>
Names: <input type="text" size="20" id="names"
onkeyup="findNames();" style="height:20;"/>
<div style="position:absolute;" id="popup">
<table id="name_table" bgcolor="#FFFAFA" border="0"
cellspacing="0" cellpadding="0"/>
<tbody id="name_table_body"></tbody>
</table>
</div>
</body>
</html>
```

کد سمت سرور نقش جستجوی نام وارد شده توسط کاربر را بازی می کند. در این کد تعدادی نام پیش فرض قرار داده شده است که نام وارد شده توسط کاربر باید از بین این نامها باشد. توجه داشته باشید در این کد اگر نامی که کاربر وارد کرده است، در بین نامهای پیش فرض وجود نداشته باشد، مقداری را که نشان دهنده عدم وجود نام در سمت سرور می باشد، به مرورگر برگردانده می شود. عمل جستجو در واقع توسط کلاسی به نام NameService انجام می شود. کد ۱۸-۴ فایل autoCompleteServlet.java و کد ۱۹-۴ فایل NameService**.java** را نشان می دهند.

کد ۱۸-۴ : autoCompleteServlet.java

```
package ajaxbook.chap4;
import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.servlet.*;
import javax.servlet.http.*;

public class AutoCompleteServlet extends HttpServlet {

private List names = new ArrayList();

public void init(ServletConfig config) throws ServletException {
      names.add("Abe");
```

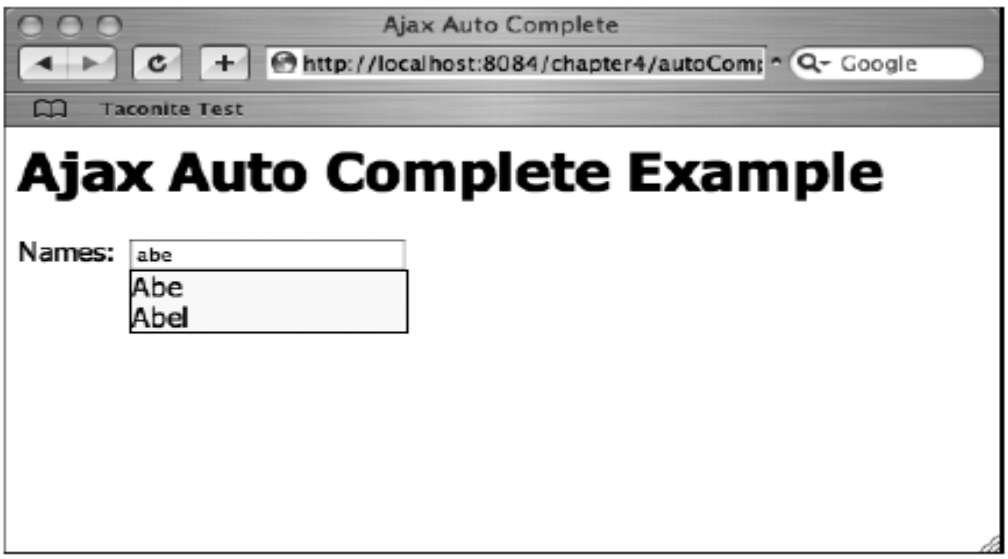
```
      names.add("Abel");
      names.add("Abigail");
      names.add("Abner");
      names.add("Abraham");
      names.add("Marcus");
      names.add("Marcy");
      names.add("Marge");
      names.add("Marie");
}
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
      String prefix = request.getParameter("names");
      NameService service = NameService.getInstance(names);
      List matching = service.findNames(prefix);
      if (matching.size() > 0) {
            PrintWriter out = response.getWriter();
            response.setContentType("text/xml");
            response.setHeader("Cache-Control", "no-cache");
            out.println("<response>");
            Iterator iter = matching.iterator();
            while(iter.hasNext()) {
                  String name = (String) iter.next();
                  out.println("<name>" + name + "</name>");
            }
            out.println("</response>");
            matching = null;
            service = null;
            out.close();
      } else {
            response.setStatus(HttpServletResponse.SC_NO_CONTENT);
            }
}
}
```

کد ۱۹-۴ : NameService.java

```
package ajaxbook.chap4;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class NameService {
private List names;
      /** Creates a new instance of NameService */
private NameService(List list_of_names) {
      this.names = list_of_names;
}

public static NameService getInstance(List list_of_names) {
      return new NameService(list_of_names);
}

public List findNames(String prefix) {
      String prefix_upper = prefix.toUpperCase();
      List matches = new ArrayList();
      Iterator iter = names.iterator();
      while(iter.hasNext()) {
            String name = (String) iter.next();
```

```
            String name_upper_case = name.toUpperCase();
            if(name_upper_case.startsWith(prefix_upper)){
                  boolean result = matches.add(name);
            }
      }
      return matches;
}
}
```

**شکل ۱۷-۴ این مثال را درحالت اجرا نشان می دهد.**




**شکل ۱۷-۴ : حالت اجرا**

# فصل پنجم : آژاکس در ASP.NET

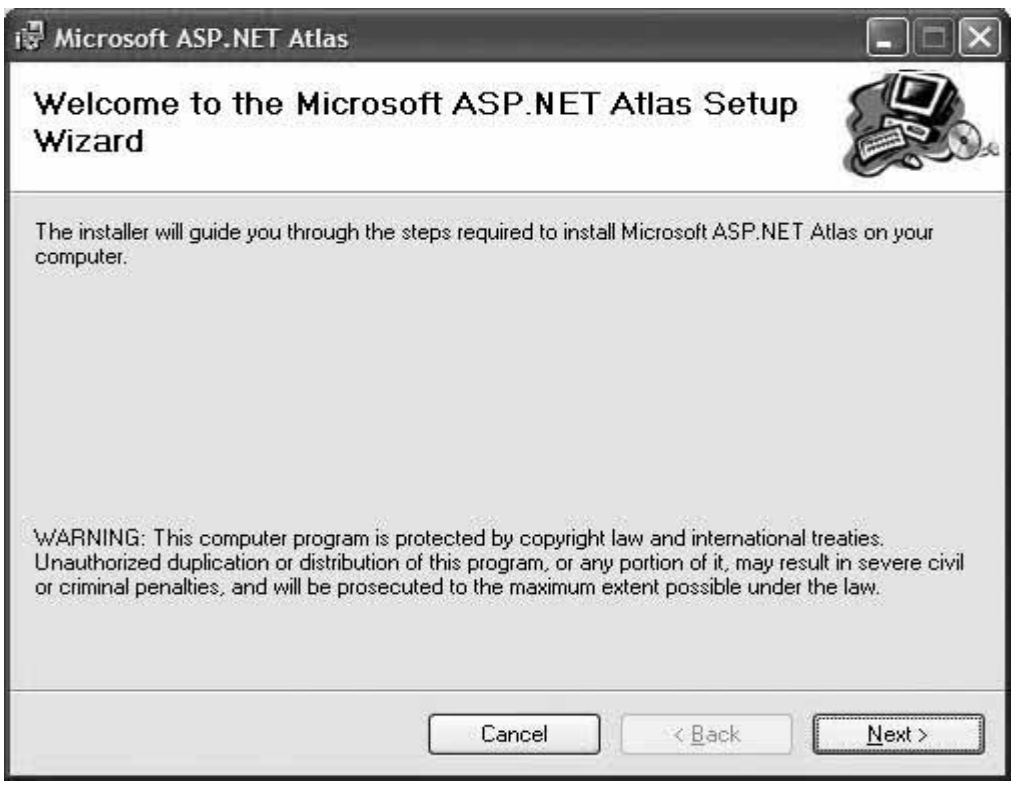
## ابزار Atlass

در فصلهای قبل با اصول اساسی آژاکس بدون وابستگی به هیچ گونه ابزاری آشنا شدید. در این فصل قصد دارم شما را با ابزاری قدرتمند به نام Atlas آشنا کنم. این ابزار از محصولات شرکت مایکروسافت می باشد که به منظور ایجاد قابلیت آژاکس در ASP.NET تولید شده است. البته این تنها ابزار موجود، به منظور استفاده در ASP.NET نیست اما به جرأت می توان گفت قدرتمندترین آنهاست. پشتیبانی تیم توسعه مایکروسافت از این ابزار باعث دلگرمی کاربران آن است. به روزرسانی های نسخه های آن و مطالب آموزشی که به صورت متن و فیلمهای آموزشی در وب سایتهای شرکت مایکروسافت قرار دارد بیانگر این حمایت می باشد.

آخرین نسخه این ابزار را از وب سایت http://www.asp.net می توانید دانلود نمایید. در همین آدرس می توانید مواردی مثل atlas documentation و atlas example را دانلود کنید که استفاده از آنها می تواند به شما کمک نماید.

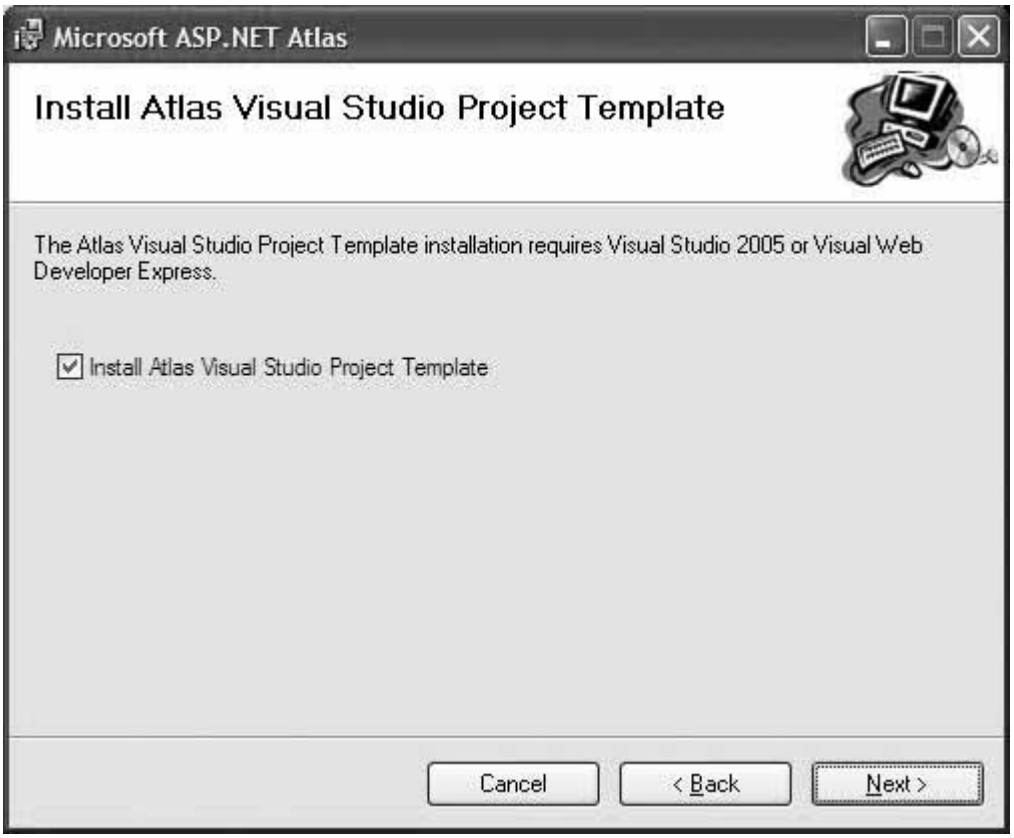
مراحل نصب ابزار Atlas به صورت زیر می باشد:

پس از اجرای فایل نصب این پنجره ظاهر می شود که اطلاعاتی راجع به کپی رایت و مراحل نصب می دهد. به مرحله بعد بروید.



شکل ۱-۵ : اولین مرحله نصب atlas

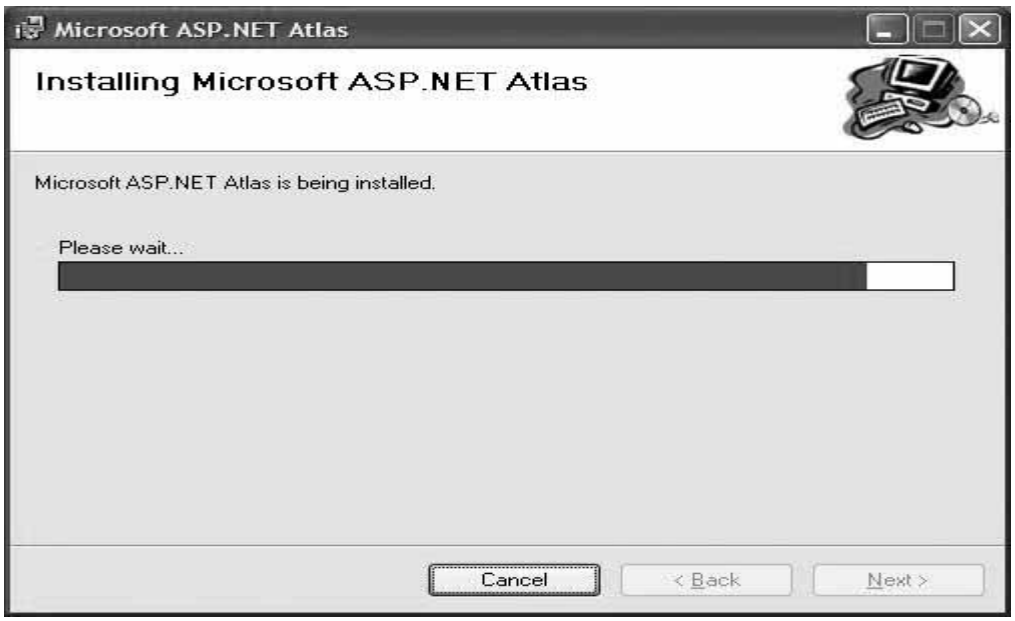
در این مرحله از شما پرسیده می شود که آیا مایل به نصب قالبهای پروژه ای هستید کـه بهتـر است با این مورد موافقت نمایید. در اینصورت یک قالب ایجاد پروژه جدید با قابلیـت Ajax به محیط Visual Studio شما افزوده می شود. به مرحله بعد بروید.




شکل ۲-۵ : دومین مرحله نصب atlas

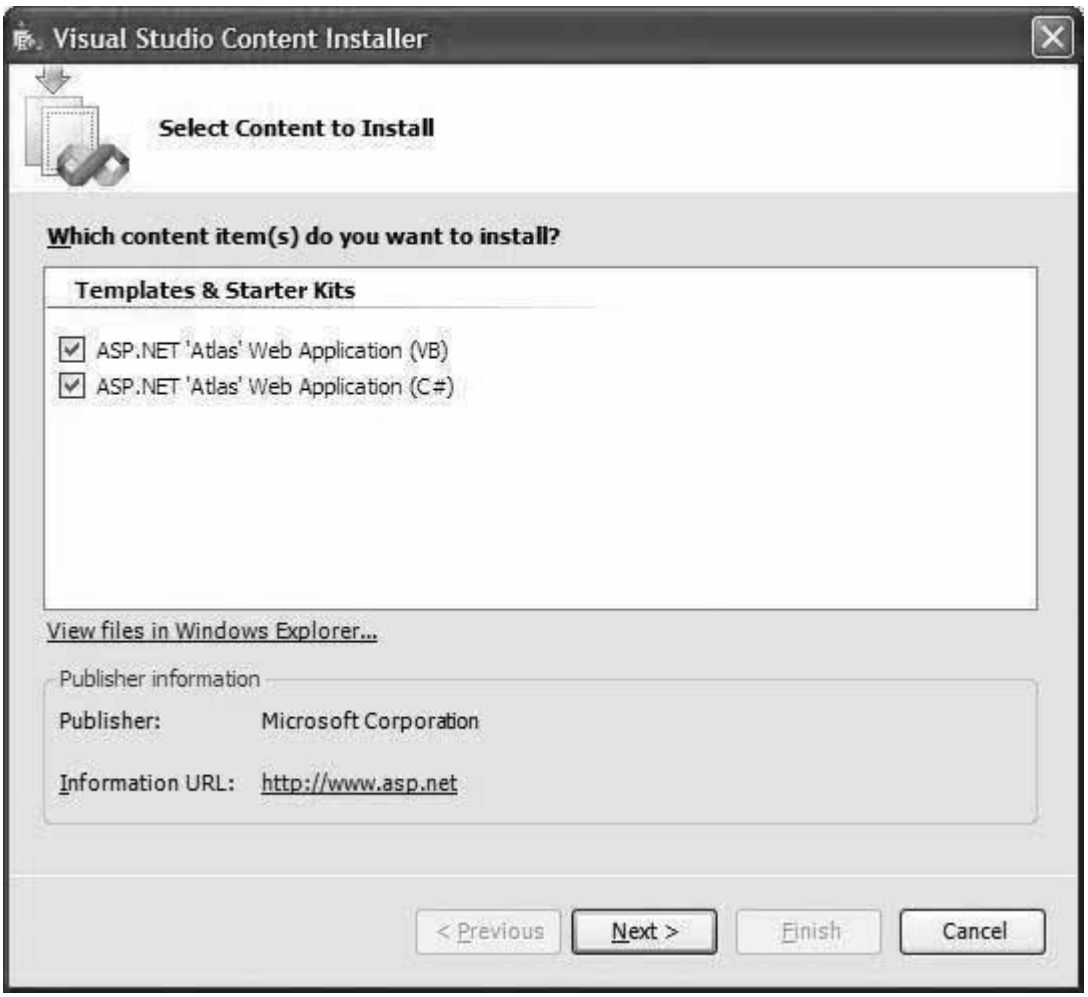
در مرحله سوم از شما پرسیده می شود که ایا مایل به ثبت و معرفی نوع فایـل asbx بـه IIS خود هستید. این نمونه فایلها به منظور دسترسی از وب سایت خود به سرویس های خـارج از دامنه وب سایت استفاده می شوند. در صورتیکه می دانید در پروژه هایتان نیاز به دسترسی بـه سرویسهای خارج از دامنه خود نظیر سرویسهای google دارید این گزینه را فعال نماییـد. بـه مرحله بعد بروید و گزینه next را انتخاب کنید تا نصب ابزار آغاز شود.


شکل ۳-۵ : سومین مرحله نصب atlas




شکل ۴-۵ : چهارمین مرحله نصب atlas

در صورتیکه نصب قالبهای پروژه ای را انتخاب کرده باشید در این مرحله پنجره زیر ظاهر میشود.




شکل ۵-۵ : پنجمین مرحله نصب atlas

بر اساس اینکه کدام قالبهای زبانهای برنامه نویسی را می خواهید، می توانید گزینه های موجود در این فرم را انتخاب نمایید.

پس از این مرحله، نصب ابزار تمام شده است.

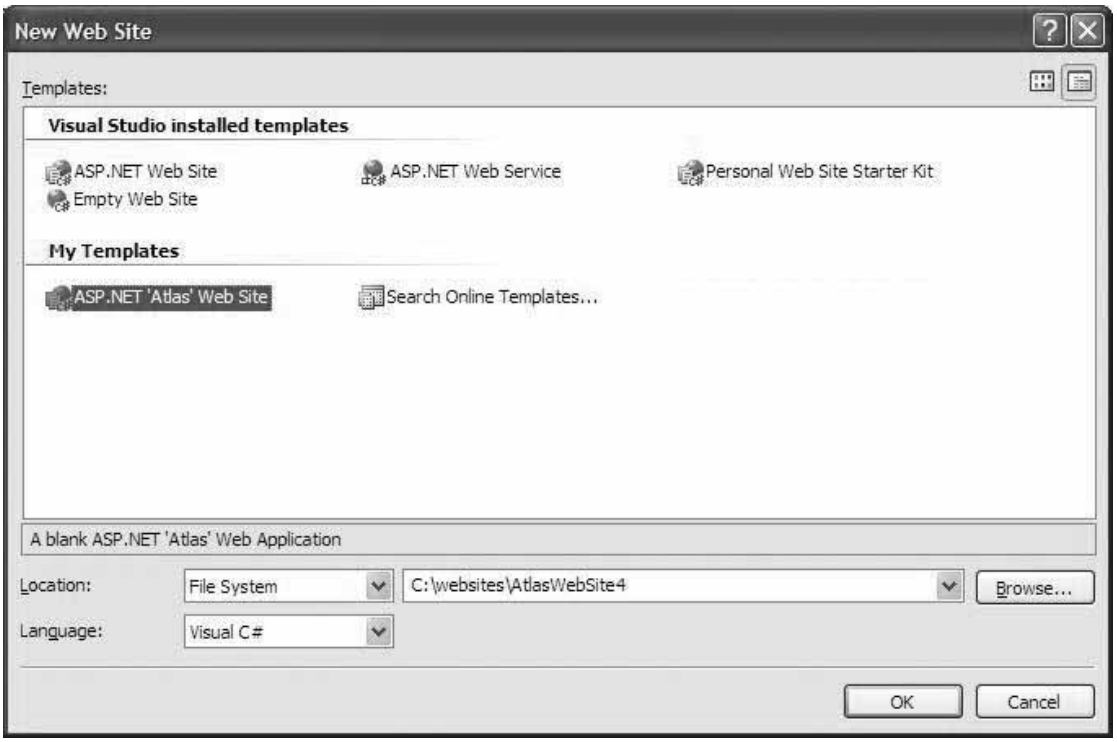
بعد از نصب این ابزار شما می توانید از این به بعد در مرحله اولیه ایجاد پروژه وب، قالب پروژه ای atlas را انتخاب نمایید که قابلیتهای استفاده از ابزار atlas به آن به صورت پیش فرض افزوده شده است.

به منظور ایجاد پروژه وب با Atlas به طریق زیر عمل نمایید :

۱. Visual Studio را اجرا نمایید.

۲. از منوی File --> New گزینه Web Site را انتخاب نمایید.

۳. در پنجره باز شده از قسمت My Template گزینه 'Asp.net 'astlas website را انتخاب نمایید.
۴. مسیر ذخیره پروژه و زبان برنامه نویسی خود را انتخاب نمایید.
۵. دکمه OK را فشار دهید.




شکل ۶-۵ : ایجاد پروژه جدید وب

وقتی از این طریق وب سایت با قابلیت atlas ایجاد می کنید، ویژوال استودیو به صورت اتوماتیک فایلهای اسمبلی مورد نیاز را در پوشه Bin مسیر پروژه قرار می دهد. همچنین تنظیمات مورد نیاز را در فایل web.config پروژه قرار می دهد.

به منظور افزودن قابلیت استفاده از ابزار Atlas به پروژه های قبلی کافی است مراحل زیر را انجام دهید:

۱. پروژه خود را در محیط ویژوال استودیو باز کنید.
۲. فایل اسمبلی ابزار atls (Microsoft.Web.Atlas.dll) را از مسیر نصب این ابزار در پوشه Bin پروژه خود کپی کنید.
۳. فایل web.config را از مسیر نصب ابزار atlas باز کنید.
۴. مولفه های مورد نیاز را از این فایل در فایل web.config پروژه خود کپی کنید.
مولفه های زیر را به عنوان فرزند مولفه <configuration> قرار دهید.

```
<configSections>
  <sectionGroup name="microsoft.web"
     type="Microsoft.Web.Configuration.MicrosoftWebSectionGroup">
     <section name="converters"
              type="Microsoft.Web.Configuration.ConvertersSection"/>
  </sectionGroup>
</configSections>

<microsoft.web>
    <converters>
      <add type="Microsoft.Web.Script.Serialization.Converters.
             DataSetConverter"/>
      <add type="Microsoft.Web.Script.Serialization.Converters.
             DataRowConverter"/>
      <add type="Microsoft.Web.Script.Serialization.Converters.
             DataTableConverter"/>
    </converters>
</microsoft.web>
```

این مولفه ها را نیز به عنوان فرزند به مولفه <System.Web> بیفزایید.

```
<pages>
     <controls>
        <add namespace="Microsoft.Web.UI"
             assembly="Microsoft.Web.Atlas" tagPrefix="atlas"/>
        <add namespace="Microsoft.Web.UI.Controls"
             assembly="Microsoft.Web.Atlas" tagPrefix="atlas"/>
      </controls>
</pages>

<!-- ASMX is mapped to a new handler so that proxy javascripts can
     also be served. -->
<httpHandlers>
      <remove verb="*" path="*.asmx"/>
      <add verb="*" path="*.asmx"
             type="Microsoft.Web.Services.ScriptHandlerFactory"
                     validate="false"/>
</httpHandlers>
<httpModules>
      <add name="ScriptModule"
             type="Microsoft.Web.Services.ScriptModule"/>
</httpModules>
```

۵. همه فایلها را ببندید.

## معرفی کامپوننتهای Atlas

بعد از نصب atlas البته نسخه ۱,۱ آن، کامپوننت های زیر به کنترلهای محیط ویژوال استودیو تحت عنوان ajax Extentions اضافه می شوند.

- Script manager
- Update Panel
- Update Progress
- Script Manager Proxy
- Timer

در ادامه توضیح مختصری در مورد هر یک از این کامپوننتها آورده شده است.

**Update Panel** : این کامپوننت نیاز به بارگذاری مجدد کل صفحه را از بین می برد. این کامپوننت به منظور مشخص کردن ناحیه ای از صفحه به عنوان ناحیه ای که باید بدون بارگذاری مجدد کل صفحه به روزرسانی شود، استفاده می شود. در سمت سرور رویدادها به همان طریق معمول پردازش می شوند اما وقتی پاسخ سرور آماده شد، به جای بارگذاری مجدد کل صفحه، تنها ناحیه مشخص شده توسط این کامپوننت به روزرسانی می شود.
این کامپوننت در صفحه ای که استفاده می شود نیاز به کامپوننت Script Manager دارد که وظیفه تولید اسکریپتهای سمت کلاینت به منظور تولید درخواست و به روزرسانی ناحیه مشخص شده بعد از دریافت پاسخ را به عهده دارد.

**Script Manager** : این کامپوننت کدهای اسکریپتی سمت کلاینت را به منظور کار کامپوننتهایی نظیر Update Panel، ایجاد و مدیریت می کند. البته به طور مستقیم نیز می توان از قابلیتهای این کانپوننت استفاده کرد.

**Timer** : این کامپوننت به شما این امکان را می دهد تا عملیاتی را در زمان مشخصی تکرار کنید. این کامپوننت خاصیتی به نام Interval بر حسب میلی ثانیه دارد که نشان دهنده زمان تکرار عملیات می باشد.

**Update progress** : با توجه به اینکه ارسال درخواست و دریافت پاسخ توسط کامپوننت Script manager بسته به سرعت اینترنت، مدتی طول می کشد و در این مدت کاربر

هیچ گونه تراکنشی توسط مرورگر نمی بیند، با استفاده از این کامپوننت می توانید ترتیبی اتخاذ کنید تا کاربر متوجه شود که عملیات وی در حال اجراست. این کامپوننت ناحیه ای در اختیار شما قرار می دهد که می توانید متن یا تصویری که نشانگر انجام عملیات می باشد به کاربر نمایش دهید. مثلاً در صورتیکه در این ناحیه بنویسید "در حال انجام عملیات..." وقتی که کاربر رویدادی را که در ناحیه Update Panel باشد اجرا نماید، این جمله در مکان مشخص شده تا زمان دریافت پاسخ سرور نمایش داده می شود.

**Script Manager proxy** : در صورتیکه بخواهید به سرویس های وب دسترسی داشته باشید این کامپوننت به شما کمک می کند که این دسترسی شما با قابلیت Ajax انجام گیرد.

در DVD ارائه شده فایلهای آموزشی تصویری که نحوه کامل کار با این کامپوننتها را آموزش می دهند ارائه شده است.

## پروژه مدیریت کاربران وب سایت

به عنوان یک مثال عملی استفاده از Atlas، پروژه ای را که در آن قصد داریم کاربران یک وب سایت و سطوح دسترسی آنها را مدیریت کنیم توضیح می دهیم. هدف این پروژه ایجاد قابلیت افزودن و مدیریت کاربران یک وب سایت با سطح دسترسی مشخص می باشد.

در هر پروژه ای که کاربران متعدد قرار است با سطوح دسترسی مختلف با محصول ارائه شده کار کنند باید ترتیبی اتخاذ شود که سمت هر کاربر در نرم افزار مشخص شود و با توجه به اینکه هر سمت در نرم افزار اجازه دسترسی به گزینه های مختلفی دارد، سطح دسترسی کاربر را چک کرد. به این منظور در ابتدا به بررسی جداول مورد نیاز بانک اطلاعاتی می پردازیم. در این پروژه از نرم افزار SQL Server 2000 به عنوان بانک اطلاعاتی استفاده شده است.

جداول مورد نیاز عبارتند از :

کاربران : این جدول اطلاعات و مشخصات هر کاربر را نگه می دارد.

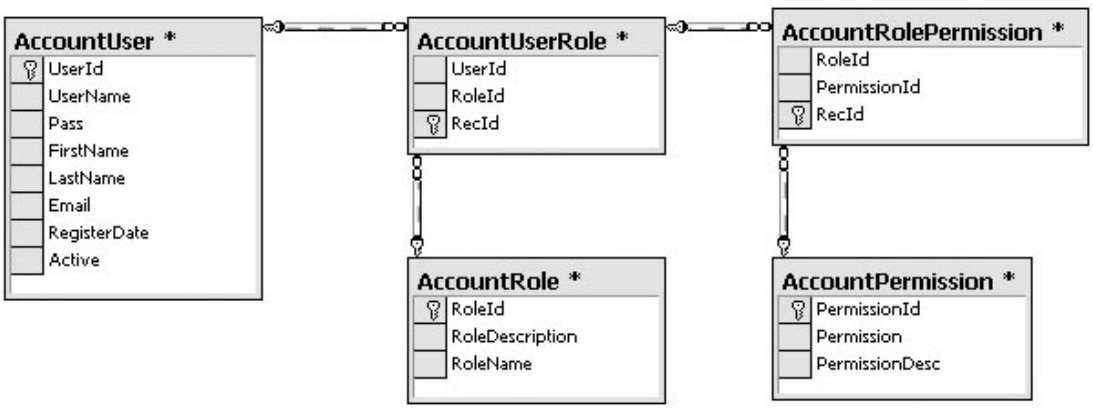
سمتها : سمتهای موجود در این جدول نگه داری می شوند.

مجوزها : مجوزهای دسترسی در این جدول قرار می گیرند.

کاربران- سمتها: در این جدول مشخص می سازیم که سمتهای هر کاربر کدامها هستند.

سمتها- مجوزها : مجوزهای هر سمت را مشخص می سازد.

نمودار جدوال را در زیر می بینید:




شکل ۷-۵ : نمودار جداول

در ادامه کد sql ساخت این جداول ارائه شده است:

کد ۱-۵ : ساخت جدول کاربران

```
CREATE TABLE [dbo].[AccountUser] (
      [UserId] [int] IDENTITY (1, 1) NOT NULL ,
      [UserName] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [Pass] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [FirstName] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [LastName] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [Email] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [RegisterDate] [datetime] NULL ,
      [Active] [bit] NULL
) ON [PRIMARY]
```

کد ۲-۵ : ساخت جدول سمتها

```
CREATE TABLE [dbo].[AccountRole] (
      [RoleId] [int] IDENTITY (1, 1) NOT NULL ,
      [RoleDescription] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [RoleName] [nvarchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
```

کد ۳-۵ : ساخت جدول مجوزها

```
CREATE TABLE [dbo].[AccountPermission] (
      [PermissionId] [int] IDENTITY (1, 1) NOT NULL ,
      [Permission] [nvarchar] (255) COLLATE Arabic_CI_AS NULL ,
      [PermissionDesc] [nvarchar] (255) COLLATE Arabic_CI_AS NULL
) ON [PRIMARY]
```

کد ۴-۵ : ساخت جدول کاربران – سمتها

```
CREATE TABLE [dbo].[AccountUserRole] (
      [UserId] [int] NULL ,
      [RoleId] [int] NULL ,
      [RecId] [int] IDENTITY (1, 1) NOT NULL
) ON [PRIMARY]
```

کد ۵-۵ : ساخت جدول سمتها- مجوزها

```
CREATE TABLE [dbo].[AccountRolePermission] (
      [RoleId] [int] NULL ,
      [PermissionId] [int] NULL ,
      [RecId] [int] IDENTITY (1, 1) NOT NULL
) ON [PRIMARY]
```

در پیاده سازی این پروژه از معماری سه لایه ای استفاده شده است و به همین منظور تمامی دستورات sql مورد نیاز به صورت روالهای ذخیره شده(Stored Procedure) در Sql Server ساخته شده است.

**لیست این روالها در ادامه آورده شده است و هر کدام که نیاز به توضیح داشته، در مورد آن توضیحاتی ارائه شده است:**

- spAddPermission
- spAddRole
- spAddRolePermission : افزودن مجوز به سمت
- spAddUser
- spAddUserRole : **افزودن سمت به کاربر**
- spAllPermission
- spAllRole
- spAllRoleOfUser : **همه سمتهای یک کاربر**
- spAllUsers
- spDeletePermission
- spDeletePermissionOfRole : **حذف مجوز از سمت**
- spDeleteRole
- spDeleteRoleOfUser : **حذف سمت از کاربر**
- spDeleteUser
- spGetPermissionOfUser : **همه مجوزهای دسترسی یک کاربر**
- spLogin : **ورود به سیستم**
- spNotPermissionOfRole : **مجوزهایی که یک سمت فاقد آنهاست**
- spNotRoleOfUser : **سمتهایی که یک کاربر فاقد آنهاست**
- spPermissionOfRole : **همه مجوزهای یک سمت**
- spRemovePermissionOfRole : **حذف مجوز از سمت**
- spRemoveRoleOfUser : **حذف سمت از کاربر**
- spUpdatePermission : **به روزرسانی اطلاعات مجوزها**
- spUpdateRole : **به روزرسانی اطلاعات سمتها**
- spUpdateUser : **به روزرسانی اطلاعات کاربران**

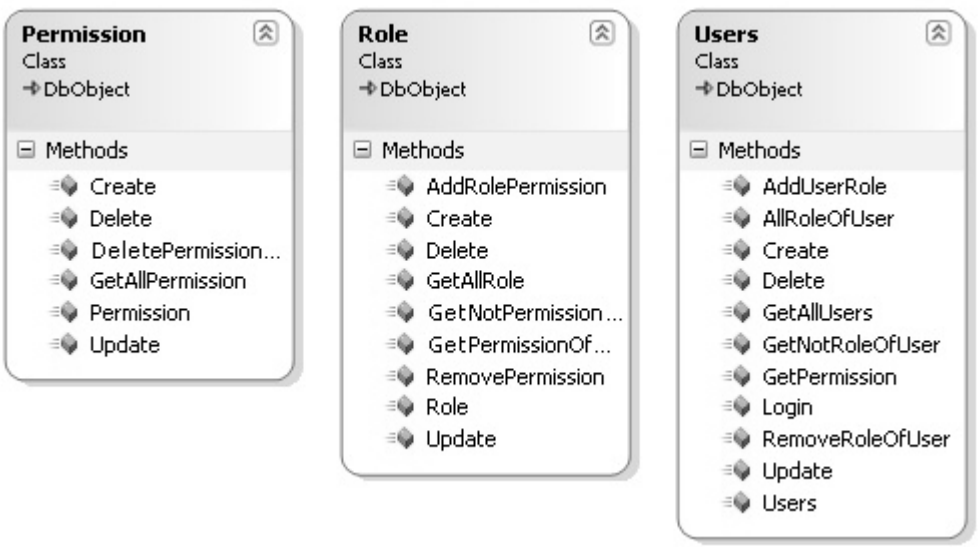
**به دلیل اینکه فایلهای پایگاه داده در DVD همراه موجود می باشد، کدهای sql این روالها در اینجا ارائه نشده است.**

در این پروژه به منظور سهولت در کار نوشتن دستورات sql گاهی نیز از view استفاده شده است. در زیر نام آنها و توضیحاتی در موردشان آورده شده است.

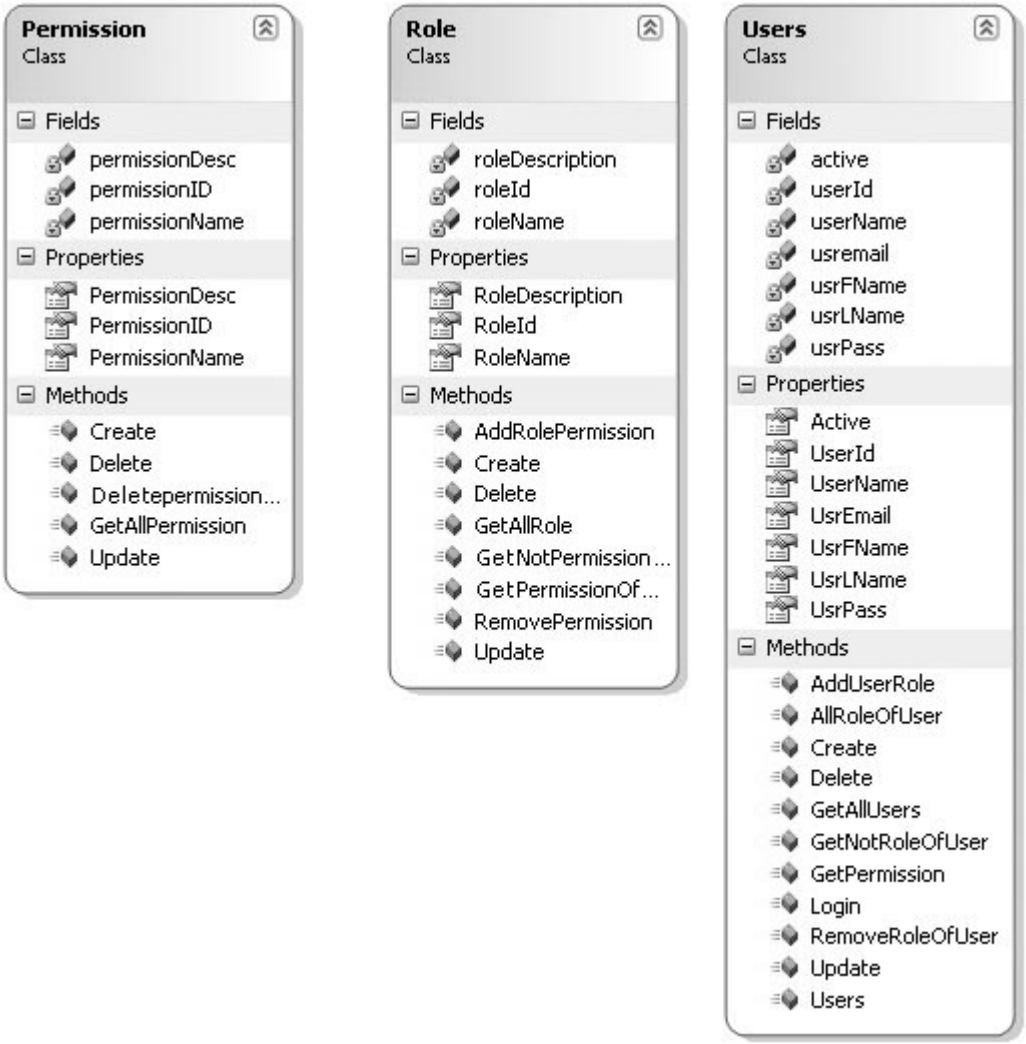
- AllRolePermission : همه سمتها به همراه مجوزهایشان
- VAllRoleOfUser : همه سمتهای کاربران
- VOuserRolePermission : همه مجوزهای کاربران به همراه اطلاعات سمتها

به اندازه کافی در مورد ساختار بانک این پروژه توضیح داده شد. اکنون وقت آن است که به ساختار پروژه بپردازیم. همانطور که گفته شد در پیاده سازی این پروژه از معماری سه لایه ای استفاده شده است. قصد ندارم به توضیح اینگونه معماری ها بپردازم چرا که به وقت و فضایی بیشتر نیاز است، فقط بررسی مختصری در این مورد خواهیم داشت.

در ویژوال استودیو می توان به ازای هر لایه یک project library ایجاد نمود. مثلاً در این پروژه، لایه data در پروژه ای به نام AccountsData و لایه Business در پروژه ای به نام AccountsBusiness پیاده سازی شده اند. به ازای هر یک از سه موجودیت اصلی یعنی کاربر،سمت و مجوز، کلاسهایی در هر کدام از این پروژه ها ساخته شده که حاوی مشخصات و عملکردهای این سه موجودیت می باشند. نمودار کلاس هریک از این پروژها را در ادامه آورده شده است. این نمودارها توسط ویژوال استودیو تهیه شده اند.



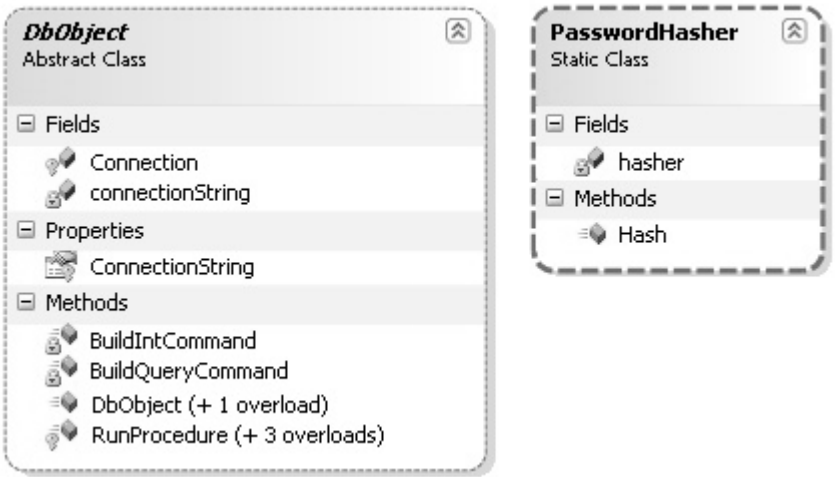

شکل ۸-۵ : نمودار کلاس پروژه لایه data (AccountsData)




**شکل ۹-۵ : نمودار کلاس پروژه لایه Business (AccountsBusiness)**

**همانطور که می بینید، کلاس های لایه data همگی از کلاسی به نام DbObject وراثت یافته اند. کلاس DbObject یک کلاس abstract است که کارهای تکراری در اجرای روالهای ذخیره شده بانک را انجام می دهد. به این ترتیب نیازی به تکرار کدهای تکراری اتصال به بانک و اجرای روالهای ذخیره شده در کلاسهای لایه Data نخواهد بود. کلاس DbObject به همراه کلاسهای کمکی دیگر نظیر PasswordHasher در پروژه کتابخانه ای به نام Core قرار دارد. کلاس passwordHasher یک کلاس static است که عملیات Hash کردن رشته رمز عبور کاربر را انجام می دهد و هنگام ایجاد کاربر جدید یا ورود به سیستم استفاده می شود.**

در زیر نمودار کلاس پروژه Core را می بینید.

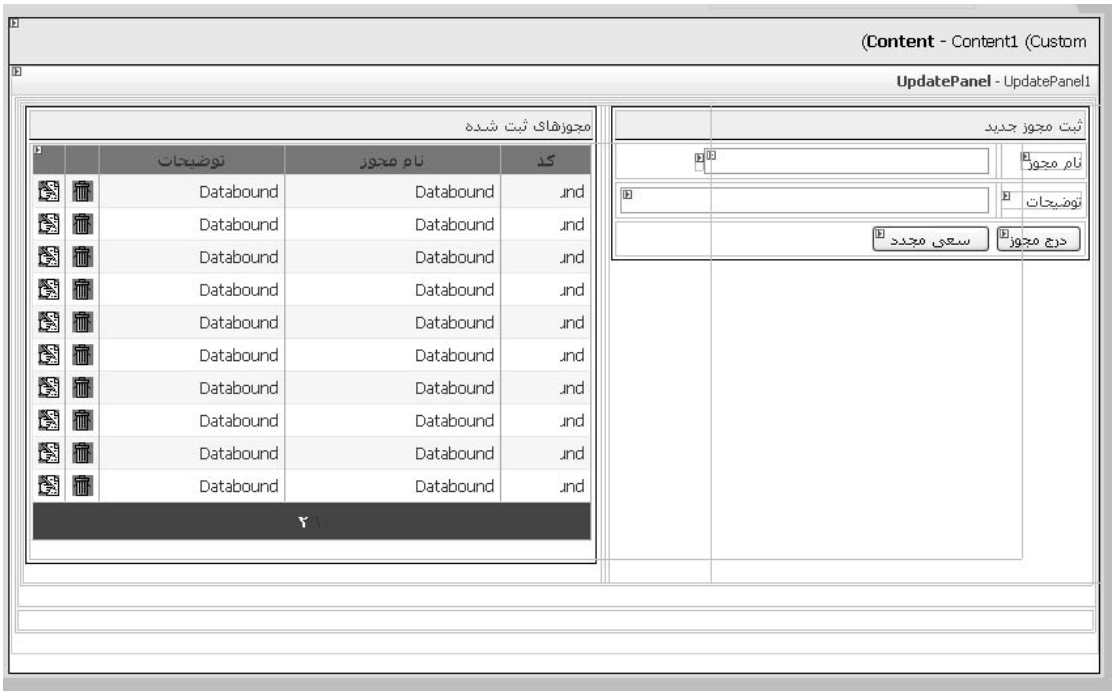



شکل ۱۰-۵ : نمودار کلاس پروژه Core

خوب، با لایه های data و business در این پروژه آشنا شدیم. لایه سوم که لایه رابط کاربر یا Interface می باشد، از قالب پروژه ای Atlas ساخته شده است و شامل صفحات وب و کدهای سمت کاربر می باشد.

کلیه عملیات صفحات مختلف این پروژه با قابلیت Ajax و بدون بارگذاری مجدد صفحات انجام می شود. به لطف ابزار Atlas این امکان بدون نیاز به کدنویسی انجام می گیرد. در آماده سازی هر یک از صفحات این وب سایت، به گونه ای عمل شده که همه مولفه های صفحه درون کامپوننت UpdatePanel که از کامپوننتهای Atlas می باشد، قرار گیرند و کامپوننت ScriptManager نیز بالاتر از دیگر مولفه های صفحه قرار داشته باشد. این تنها و شاید سخت ترین کاری بوده که به منظور افزودن قابلیت آژاکس به صفحات این وب سایت، از جانب من صورت گرفته است.

به عنوان نمونه محتوای بصری صفحه مجوزهای وب سایت را در شکل ۱۱-۵ آورده شده است. همانطور که می بینید، همه مولفه های صفحه درون کامپوننتی به نام UpdatePanel1 قرار گرفته اند.




**شکل ۱۱-۵ : نحوه قرار گرفتن مولفه های صفحه**

**جالب توجه است که با استفاده از ابزار Atlas دیگر نیازی به تغییر رویه برنامه نویسی وب نمی باشد و تنها با رعایت نحوه چیدمان مولفه های صفحه این قابلیت به آن صفحه افزوده می شود.**

**در پایان ذکر یک نکته در استفاده از ابزار Atlass را لازم می بینم. درست است که این ابزار کارها را بسیار آسان می کند اما همواره مسائلی یافت می شود که تنها با علم به اصول، قابل حل می باشند. پس نه تنها در استفاده از تکنیک آژاکس، بلکه در هر زمینه ای سعی به یادگیری اصول داشته باشید.**

# پیوست ۱ : منابع


1. Foundations Of Ajax, Ryan Asleson and Nathaniel T. Schutta, Apress,2006
2. Foundations Of Atlas, Laurence Moroney,Apress,2006
3. اصول مهندسی اینترنت،مهندس احسان ملکیان،نص،۱۳۸۵
4. Programming Atlas,Christian Wenz,O'Reily,2006
5. Beginning JavaScript with DOM Scripting and Ajax,Christian Heilmann,Apress,2006
6. Professional Ajax, Nicholas C. Zakas, Jeremy, McPeak Joe Fawcett,wiley,2006

# پیوست ۲ : محتویات DVD همراه

- **Ajax E-Book** : مجموعه مناسبی از کتابهای الکترونیکی در مورد آژاکس
- **Ajax Video** : مجموعه ای از فیلم های آموزشی در مورد آژاکس
- **Atlas Video** : سری فیلم های آموزشی استفاده از ابزارهای Atlas
- **ASP.NET E-Book** : مجموعه کتابهایی در مورد ASP.NET
- **ASP.NET Video** : شامل فیلم های آموزشی در مورد ASP.NET
- **SQL-Server E-Book** : مجموعه کتابهایی در مورد SQL Server
- **SQL Server 2005 Video** : سری آموزش SQL Server 2005
- **Book Project** : شامل مثالهای بررسی شده در کتاب
- **ASP.NET Project** : پروژه فصل پنجم