.net

از دیدگاه برنامه‌نویسی

ترجمه: بابک احترامی

نمی‌توان منکر این شد که مایکروسافت با ابداع فناوری Net. ، فرآیند توسعه نرم‌افزارهای ویندوزی را دگرگون کرده است. البته، اختلاف نظر و بحث‌های زیادی بر سر خوب یا بد بودن Net. مطرح شده است. بعضی‌ها (در واقع منفی‌بافان) می‌گویند تغییراتی که در این محصول نسبت به محصولات قبل پدید آمده‌اند به حدی زیاد و وسیع هستند که پذیرفتن آن را بسیار سخت کرده است. این افراد معتقدند که وسعت این تغییرات چنان زیاد است که کسی قبول نمی‌کند برنامه‌های موجود خود را با استفاده از این فناوری جدید بازنگری و بازنویسی کند. آن‌ها همچنین از تغییراتی که در زبان ویژوال بیسیک صورت گرفته شکایت دارند.

اگر بخواهیم منصف باشیم،باید اعتراف کنیم که این افراد زیاد هم اشتباه نمی‌کنند. انتقال برنامه‌های بزرگ به این فناوری، بدون طراحی دوباره و بازنویسی آن‌ها اگر غیرممکن نباشد، بسیار دشوار است، ولی به هر ترتیب، برای بهره بردن از قابلیت‌ها و مزایای یک سیستم و زبان جدید باید زحمت طراحی مجدد را بپذیریم. مهندسین باید وقت زیادی را صرف یادگیری و حرفه‌ای شدن با این فناوری جدید بکنند؛ چرا که تغییراتی بنیادی در ساختار زبان برنامه‌نویسی صورت گرفته و ساختمان Net. (با بیش از ۶۵۰۰ کلاس) واقعاً بزرگ است. به اعتقاد من این همه زحمت ارزشش را دارد. زبان Net. بسیار قدرتمند و حتی مطمئن‌تر شده است. در عین حال، هنوز هم آسان است؛ فقط «فرق» دارد. قابلیت‌های جدید، خیره‌کننده هستند و انسان را وامی‌دارند به چیزهای جدید فکر کند. حتی محیط کار ویژوال استودیو خیلی پیشرفت کرده؛ مثل این که مایکروسافت طی این سه سال گذشته تمام شکوه و گلایه‌های برنامه‌نویسان را شنیده تا در این محصول تازهٔ خود ایرادها را برطرف کند.

اما من خودم نمونه خوبی هستم برای این که ثابت کنم چرا Net. و تغییرات ویژوال بیسیک بد نیستند. من خودم اوایل خیلی نگران بودم، چرا که سال‌ها بود با VB، COM، DCOM، MST، ASP، ADO و بقیه «فناوری‌های مخفف» (یعنی فناوری‌هایی که با کلمات مخفف بیان می‌شوند) کار می‌کردم و راحت بودم. با خودم فکر می‌کردم چرا باید خودم را به زحمت بیندازم و به این شیوه جدید نرم‌افزارسازی روی بیاورم. ویژگی‌های جالبی دارد، ولی آیا این همه تلاش برای رسیدن به آن ارزش دارد؟ ولی در طول دوره آموزش و به کارگیری Net. و ویژوال بیسیک جدید، فهمیدم که نرم‌افزاری که با این ابزارها تهیه می‌شود به قابلیت‌هایی پیشرفته مجهز است که به آسانی و به سرعت حاصل می‌شوند. بعد از این که به زبان آن عادت کردم و با طرز کار این فناوری جدید آشنا شدم، دیدم که چه توانایی‌ها و امتیازات زیادی در برنامه‌نویسی در اختیار دارم. تازه، من هنوز یک دهم آن ۶۵۰۰ کلاس را هم استفاده نکرده‌ام. برنامه‌نویسانی چون من که از Net. استفاده می‌کنند، برنامه‌هایی را تولید خواهند کرد که تا قبل از این حتی فکرش را هم نمی‌کردند. دنیا عوض شده!

زبان مشترک در زمان اجرا

احتمالاً درباره زبان ویژوال استودیودات‌نت چیزهایی شنیده‌اید و شاید تعجب کرده باشید که امتیاز بزرگ آن چیست، یا به چه کار می‌آید. ویژگی زبان مشترک در زمان اجرا امکان اضافه شدن هر زبانی به محیط ویژوال استودیو و تولید آسان برنامه‌های باینری را فراهم ساخته است. مقصود این است که رسم‌الخط (یا syntax) زبانی که استفاده می‌شود باید کمابیش یک نظر شخصی باشد و تأثیری روی قابلیت‌ها یا عملکرد برنامه نگذارد. وقتی برنامه VB.Net خود را کامپایل می‌کنید، اولین کاری که صورت می‌گیرد ترجمه آن به یک «حالت واسطه» است که «زبان میانی» یا IL (مخفف Intermediate Language) نامیده می‌شود. این زبان، خیلی شبیه به اسمبلی است. تمام زبان‌ها در محیط ویژوال استودیو به IL ختم می‌شوند. و این زبان دست آخر به یک فرمت باینریِ اجرا شدنی تبدیل خواهد شد. فایده این کار چیست؟ اگر بخواهید زبانی بنویسید که در محیط ویژوال استودیو قابل استفاده باشد، فقط باید تا حد تولید IL پیش بروید. این کار را خیلی آسان‌تر می‌کند، چرا که می‌توانید از کامپایلر نهایی که مایکروسافت قبلاً نوشته استفاده کنید، یعنی همان کامپایلر IL. به علاوه، می‌توانید از هر زبانی که فکر می‌کنید برای انجام کار شما مناسب‌تر است استفاده کنید، بدون این که انتخاب زبان تأثیری محسوس روی قابلیت‌های برنامه شما داشته باشد. علاوه بر تمام این‌ها، یک مزیت بزرگ‌تر هم وجود دارد: قابلیت

همکاری بین کامپوننت‌ها. هر زبانی که در ویژوال استودیو با زبان مشترک کار می‌کند «کد مدیریت شده» (managed code) نامیده می‌شود. کامپوننت‌هایی که با کد مدیریت شده نوشته شده باشند می‌توانند بدون هیچ دردسری کامپوننت‌هایی را مورد استفاده قرار بدهند که به زبان‌های دیگر که مدیریت شده نوشته شده باشند. یعنی مثلاً تابعی را در ویژوال بیسیک می‌نویسیم و آن را در C فرا می‌خوانیم. کلاسی را در C++ می‌سازیم و در SmallTalk مورد استفاده قرار می‌دهیم. همه این‌ها می‌توانند با هم کار بکنند، چرا که دست آخر همه IL هستند. به همین دلیل هر کس می‌تواند واقعاً از هر زبانی که دوست دارد استفاده بکند. خود زبان عملاً اهمیتی ندارد و یک انتخاب شخصی و سلیقه‌ای است که عمدتاً به صرف و نحو و(syntax) آن برمی‌گردد. پس از زبانی استفاده کنید که به نظرتان شیرین‌تر است.

کار با دیتا به کمک ADO.NET

دسترسی به بانک اطلاعات همیشه بخش مهمی از فرآیند توسعه نرم‌افزار بود است. این کار سابقاً بسیار دشوار بود و اغلب به APIهای خاص نیاز بود. بعد از یک دوره تاریکی، ODBC به میدان آمد که به تولیدکنندگان امکان می‌داد APIهای خود را به صورت استاندارد شده تولید و عرضه کنند. به‌این ترتیب، مردم می‌توانستند با در نظر گرفتن یک «واسطه» برنامه خود را بنویسند و بعداً بدون تغییر دادن تمام کد، بین بانک‌های اطلاعات مختلف سوئیچ کنند. ولی واسطه ODBC پیچیده بود و یادگیری آن دشوار. راه‌حل مایکروسافت ADO بود، مجموعه‌ای از اشیاء اولیه که دسترسی به داده‌ها را آسان‌تر می‌کرد. درست است که ADO یک پیشرفت محسوب می‌شد، ولی مشکلات خاص خودش را داشت و در ضمن چیز کاملی هم نبود. بالاخره ADO بیرون آمد و واقعاً انقلابی به پا کرد. این فناوری زود رواج پیدا کرد و مورد استفاده اکثریت قرار گرفت. پس چه شد که ADO دیگری مطرح شد؟ با پیدایش اینترنت و برنامه‌های پیچیده مبتنی بر وب که نیاز به کار با بانک اطلاعات داشتند، ADO خوب جواب می‌داد اگر می‌خواستیم یک دسترسی ثابت و مستقیم به بانک اطلاعات داشته باشیم. ADO از قابلیت‌های اساسی «غیراتصالی» (Connectionless) برخوردار بود، ولی ارسال داده‌ها از طریق یک اتصال HTTP با ADO بسیار مشکل بود. مایکروسافت با ساخت ADO.NET این مشکل را نیز حل کرد؛ یک ADO جدید که اینترنت را می‌شناسد و می‌تواند با چارچوب جدید Net. همکاری کند. این دو فناوری تفاوت‌های زیادی با هم دارند، ولی من فقط به دو مورد اشاره می‌کنم که از بقیه مهم‌تر هستند. اولاً، ADO.NET به صورت یک data system غیراتصالی کار می‌کند. این به‌آن معنی است که داده‌ها ابتدا از بانک اطلاعات به داخل Container ریخته شده و سپس به دست برنامه‌ای سپرده می‌شوند که به آن‌ها نیاز دارد. اتصال به بانک اطلاعات فقط در هنگام ضرورت برقرار می‌شود و پس از رفع نیاز بلافاصله قطع خواهد شد تا سایر برنامه‌ها هم بتوانند از منابع بهره‌برداری کنند. در محیطی مثل اینترنت، که درخواست‌های زیادی به طور همزمان در آن صورت می‌گیرند، منابعی چون اتصالات بانک اطلاعات ارزش زیادی دارند. راحت به دست نمی‌آیند. توانایی ADO.NET در استفاده از اتصال بانک اطلاعات برای مدت کوتاه و سپس آزاد کردن آن، امتیازی است که برای برنامه‌های اینترنتی بسیار مفید است. دومین اختلاف مهم این است که داده‌ها در DataSetهای ADO.NET در قالب XML ذخیره می‌شوند، که یک فرمت استاندارد مورد پذیرش صنایع است. این در حالی است که RecordSetهای ADO با فرمت درونی خودشان نگهداری می‌شوند. این فرمت برای برنامه‌ای که اصلاً چیزی از RecordSet نمی‌داند هیچ فایده‌ای ندارد. RecordSetهای مزبور نمی‌توانند مثلاً برای ارسال داده‌ها به یک «جاوااپلت» در صفحه‌ای از وب مورد استفاده قرار بگیرند. ولی ماهیت ایکس‌ام‌ال‌ِ DataSetهای ADO.NET در هر کلاینتی که زبان XML را بفهمد (که این روزها، همه می‌فهمند) قابل استفاده است. ADO.NET اشیاء و کامپوننت‌های بسیار مفیدی دارد که صحبت دربارهٔ آن‌ها را به فرصتی دیگر موکول می‌کنم.

سرویس‌های وب

این روزها، موضوع سرویس‌های وب بحث داغ محافل و مجالس فنی شده است و گاهی آن‌چنان درباره آن‌ها اغراق می‌شود که آدم به یاد تبلیغات تلویزیونی می‌افتد. قرار است این فناوری جدید عرصه‌های تازه‌ای را در فعالیت‌های B2B و مخابرات بگشاید و قابلیت همکاری بین برنامه‌های را برای همگان فراهم سازد. همه قبلاً از این جور چیزها زیاد شنیده‌ایم، به ویژه از طرف شرکت‌هایی که خود را پیشرو در فناوری می‌دانند، از جمله مایکروسافت. ولی واقعاً سرویس‌های وب چه هستند و به چه درد می‌خورند؟ دوست دارم پاسخ این سؤال را با چند مثال ذکر کنم. فرض کنید شما برنامه‌نویسی در یک شرکت IT باشید که از برنامه‌های کاربردی زیادی نگهداری کرده و بعضی از آن‌ها داده‌هایی مشترک دارند. مثلاً، بانک اطلاعات «مشتری‌ها» ممکن است مورد استفاده چند برنامه متفاوت باشد، یا بانک اطلاعات «محصولات» در برنامه‌های بسیاری مورد دسترسی قرار بگیرد. با این اوصاف، می‌توانستید یک وب سرویس عمومی دسترسی به داده‌ها را تولید کنید که داده‌های مشتری‌ها یا محصولات را بر می‌گرداند (یا به روز می‌کند). در این صورت، هر برنامه‌ای می‌توانست از این سرویس استفاده کرده و به بازیابی یا تغییر داده‌های مورد نیاز خود بپردازد. کاربران با استفاده از اتصالات اینترنتی خود می‌توانستند از طریق فیلدهایی به این داده‌ها دسترسی داشته باشند. حتی کاربران‌از راه دور هم می‌توانستند از مثلاً ماشین یا دفتر مرکزی به این داده‌ها دسترسی پیدا کنند. مثالی دیگر: شما برای یک شرکت بازاریابی کار می‌کنید که آدرس پستی و سایر اطلاعات مردم (در واقع مشتریان) را در اختیار دارد. حال می‌خواهید وب سرویسی را ایجاد کنید که به هرکس (که به شما پول داده) امکان می‌دهد به قابلیت «جستجوگری» سرویس شما دسترسی پیدا کرده و فهرستی از مشتریان احتمالی را که ممکن است به محصولات آن‌ها علاقه‌مند باشند برگرداند. باز هم مثالی دیگر: شما برنامه‌نویسی هستید که مجموعه‌ای از امکانات و اسباب را جمع‌آوری کرده‌اید که شرکت شما دوست دارد به سایر شرکت‌های تهیه و توسعه نرم‌افزار بفروشد (مثلاً یک مجموعه بزرگ از تصاویر گرافیکی جالب یا سیستمی که می‌تواند هزینه‌های تولید را تخمین بزند). به جای فروش این بسته، تهیه سی‌دی و خرج تراشی برای توزیع آن، می‌توانید به عنوان یک سرویس وب این مجموعه را عرضه کنید. هرکس به این امکانات احتیاج داشته باشد می‌تواند از طریق اینترنت به آن‌ها دست پیدا کند. فکر می‌کنم با این چند مثال، موضوع را برای شما روشن شده باشد. سرویس‌های وب هرگونه خدماتی را ارائه می‌دهند، بدون این که کاری با مسائل نشر و توزیع داشته باشیم. تهیه این سرویس‌ها در ویژوال بیسیک دات نت بسیار آسان است. در واقع چارچوب دات‌نت همه جور امکانات در اختیار ما گذاشته تا راحت باشیم.

سیستم عمل از راه دور (Remoting)

مایکروسافت اصولاً دو مکانیزم عمده در اختیار ما گذاشته که امکان می‌دهند توابع و سرویس‌ها را از دور صدا کنیم. یکی از این مکانیزم‌ها وب سرویس‌ها بودند که دربارهٔ آن‌ها صحبت کردیم. مکانیزم دیگر Remoting نام دارد. درست است که وب سرویس‌ها می‌توانند از طریق وب و اتصالات HTTP به امکانات دور دست پیدا کنند، ولی این به تنهایی نمی‌تواند تمام مسائل ما را حل کند. وقتی از وب استفاده نمی‌کنیم، باز هم باید

راهی وجود داشته باشد که یک سرویس را از راه دور فرابخوانیم. پیش از این چنین کاری را در قالب DCOM انجام می‌دادیم، که امکان این را می‌داد کامپوننت‌های موجود در ماشینی دیگر یا فضای پردازشی دیگر را صدا بزنیم. این ابزار کار ما را راه می‌انداخت، ولی توزیع بخش پراکسی کامپوننت واقعاً پرزحمت بود. در ضمن، اگر می‌خواستیم از یک دیواره آتش عبور کنیم، با مسائل جدی روبه‌رو می‌شدیم. در Net. این قبیل کارها را از طریق کامپوننت‌های دور انجام می‌دهند. ساخت کامپوننت‌های دور کمی با ساخت کامپوننت‌های DCOM فرق دارد. کامپوننت‌های دور (که به «اشیاء دور شدنی» یا remotable موسومند) انعطاف‌پذیری بیشتری نسبت به کامپوننت‌های DCOM دارند. این اشیاء، روی انواع مختلف اتصالات قابل استفاده بوده و بهتر می‌توانند به اشتراک منابع خود بپردازند. به لحاظ تئوری، اشیاء دورشدنی می‌توانند در شبکه‌های گوناگون و حتی سیستم عامل‌های مختلف کار کنند. سیستم Remoting حتی از این قابلیت حمایت می‌کند که یک کامپوننتِ در حال اجرا در یک سیستم را به پائین بکشانیم، آن را سریال کنیم، به کامپوننتی در ماشینی دیگر بفرستیم، دوباره آن را برپا کنیم و اجرای آن را از سر بگیریم. اگر قرار باشد معماری Remoting را در یک پاراگراف توضیح بدهم، چنین می‌گویم: یک شی دور داخل یک سرور در جایی دیگر قرار می‌گیرد. وقتی کلاینت شما نمونه‌ای از آن شی دور را تعریف و ایجاد می‌کند، در واقع یک «میانجی» (یا proxy) برای آن کامپوننت ساخته می‌شود. این نمونه یک کامپوننت حاضر به خدمت است که از دید برنامه شما دقیقاً مثل همان کامپوننت دور است، ولی این کامپوننت می‌داند که چگونه درخواست شما را بسته‌بندی کرده و به کامپوننت دور واقعی برساند. این کار، با استفاده از یک کانال صورت می‌گیرد، کانالی مخابراتی که خود شما تعیین می‌کنید. کانال مزبور درخواست شما و داده‌های همراه آن را به کامپوننت دور می‌رساند. کامپوننت دور هم بسته‌های ارسالی از طرف شما را گرفته، سر و شکلی تازه به آن‌ها داده و برای اجرا آماده می‌کند. همین فرآیند به صورت معکوس در برگشت داده‌ها انجام می‌شود.

ابزارهای آماده‌سازی نرم‌افزار برای اعزام

اگر تا به حال از ابزارهای داخلی ویژوال بیسیک ۶ برای تهیه یک «برنامه نصب» استفاده کرده باشید، حتماً متوجه شده‌اید که این ابزارها چقدر بی‌کیفیت و محدود هستند. این ابزارها، فقط برای برنامه‌های ساده‌ای که چند فایل بیشتر ندارند خوب بودند. اگر می‌خواستیم کوچک‌ترین تغییری در ظاهر یا عملکرد برنامهٔ نصب بدهیم، مجبور بودیم به سراغ کدی برویم که این برنامه را می‌سازد، و هر کدام از شما که این کد را دیده باشد، تصدیق می‌کند که چیز وحشتناکی است. علاوه بر این‌ها، UI برنامه نصب خیلی بی‌مزه بود و انگار یک بچه آن را ساخته است. انتخاب دیگر، ابزارهای نصب طرف سوم بودند که برای استفاده از آن‌ها به اندازه یک مدرک لیسانس معلومات لازم داشتیم. درست است که این ابزارها قابل بسط هستند، ولی فقط با نوشتن DLL هایی که اینترفیس‌هایی حرفه‌ای دارند. به محض این‌که بخواهیم کاری فراتر از عُرف و قابلیت‌های عادی آن‌ها بکنیم، مثل این است که با یک دیوار مواجه شده باشیم. مایکروسافت با توجه به حجم عظیم سفارشات و انتقاداتی که از ابزارهای نصب تعبیه شده در VB6 داشته، اقدام به اصلاح و بهبود ابزارهای نصب در ویژوال استودیو کرد و سیستم Installer را پدید آورد. مایکروسافت هنگام ساخت این سیستم، بسیاری از مشکلات و ایرادهای قبلی را مورد توجه قرار داد. این برنامه واقعاً یک ابزار نصب حرفه‌ای و تمام عیاری است که بسیار انعطاف‌پذیر و کاملاً قابل برنامه‌نویسی است و با اینترفیس‌های برنامه‌نویسی خاص خود عرضه می‌شود. این برنامه، براساس اصول کاملاً متفاوتی عمل می‌کند که فقط مایکروسافت می‌توانسته از عهده آن برآید. برنامه Installer در واقع موتوری است که در سیستم عامل گنجانده شده است. این موتور، در حال حاضر همراه با ویندوز اِم‌ای، ویندوز ۲۰۰۰ و ویندوز اکس‌پی عرضه می‌شود و روی فایل‌های Installer (یا MSI) عمل می‌کند. وقتی یک بسته نصب برای Installer می‌سازید، فایلی که ایجاد می‌شود (فایلی با پسوند msi.) تقریباً هر چیزی که فکرش را بکنید دارد: تمام فایل‌هایی که می‌خواهید در کامپیوتر کاربر نصب شوند، دستورالعمل‌هایی برای نحوهٔ نصب فایل‌ها و مکان آن‌ها، هر فایلی که به حمایت برنامه نصب می‌آید (مثلاً تصاویری که در طول فرآیند نصب نشان داده می‌شوند و هر کد سفارشی که برای تکمیل نصب نوشته باشید. با تمام این‌ها، این فایل حاوی برنامه‌ای اجرایی «نیست» که فرآیند نصب را به جریان بیندازد. وقتی روی یک فایل MSI کلیک مضاعف می‌کنید، ویندوز آن را به موتور Installer منتسب می‌کند و طوری آن را به اجرا در می‌آورد که گویی Installer بخشی از برنامه نصب بوده است. نکتهٔ جالب‌تر این است که اگر بخواهید برنامهٔ خود را به پلت‌فرمی نصب کنید که مجهز به موتور Installer نیست، می‌توانید این موتور را به بسته نصب خود «اضافه» کنید. در این حالت، کلیک مضاعف روی فایل MSI باعث می‌شود اول موتور Installer در سیستم عامل نصب شود و سپس برنامه نصب به اجرا درآید.

ساخت صفحات وب با ASP.NET

ASP امروزه یکی از گسترده‌ترین روش‌های توسعهٔ برنامه‌های کاربردی وبی شده است. آسان بودن این تکنولوژی دنیای وب سرورها را که زمانی فقط در اختیار گروه کوچکی از برنامه‌نویسان یونیکس بود که بلد بودند با PERL یا C به پیاده‌سازی CGI بپردازند، پیش روی میلیون‌ها برنامه‌نویس ویژوال بیسیک در سراسر دنیا گشوده است. وی‌بی اسکریپت طرف سرور در کنار اشیاء ذاتی ASP (از قبیل Request، Response، Application و Session) به میزان قابل ملاحظه‌ای از سختی‌ها و زمان آموزش افراد کاسته است. البته این آسان بودن بدون جنبه منفی به دست نیامده است. وی‌بی اسکریپت یک زبان interpret شده است، یعنی وب سرور باید در پاسخ به درخواست‌هایی که دریافت می‌کند، به Parse کردن و کامپایل کردن صفحات ASP در سرور بپردازد. اسکریپت ASP به صورت inline در خروجی HTML صفحه درج می‌شود و به سرعت یک کد اسپاگتی گونه را پدید می‌آورد که نگهداری و ارتقاء آن کار راحتی نیست. ادیتورهای WYSIWYG مکرراً از این کد طرف سرور ایراد می‌گرفتند که نمی‌فهمند چه می‌گوید. شی پرمصرف Session خواست که امکان «نگهداری وضعیت» (state maintnenace) را در یک محیط ذاتاً بی‌ثبات (stateless) فراهم کند، ولی این راه حل در یک محیط وب کارساز نشد. هیچ زیرساختار خوبی برای cache کردن خروجی در طرف سرور وجود نداشت. مایکروسافت با آگاهی کامل از تمام این مشکلات قدم پیش گذاشت و چارچوب ASP.NET را از ریشه خلق کرد تا با این مشکلات مقابله کند. در نگاه اول به نظر می‌رسد که ASP.NET شباهت بسیاری به Active Server Page قدیمی دارد. مایکروسافت حتی ادعا می‌کند خیلی راحت پسوند یک صفحه asp را به aspx تغییر دهید و مطمئن باشید که آن صفحه در چارچوب جدید کار خواهد کرد، ولی هرچه بیشتر به عمق این چارچوب وارد شوید، متوجه خواهید شد که این مجموعه یک نسخه کاملاً بازبینی شده و تکمیلی ASP است، و راه حل ASP.NET بیشتر شبیه به راه حل VB6 است تا باتلاق اسکریپتی که داشتیم خودمان را در آن غرق می‌کردیم. سؤال می‌کنید ASP.NET چیست؟ در جواب می‌گویم که ASP.NET یک چارچوب کاملاً پیشرفته است که برای توسعه برنامه‌های قابل بسط وب به کار می‌رود. سیستم

IIS می‌تواند چنان تنظیم شود که پاسخ به درخواست‌هایی را که برای انواع مختلف فایل‌ها انجام می‌شوند، به زیر سیستم‌های مختلف نصب شده در سرور محول کند. با ASP، درخواست‌هایی که برای فایل‌های با پسوند asp. صورت می‌گیرند به ASP.DLL واگذار می‌شوند. ASP.DLL نسخه اجرایی ASP است که کد اسکریپت تعبیه شده در HTML درون فایل را Parse کرده و خروجی HTML آن صفحه را (به صورت پویا) تولید می‌نماید. چارچوب ASP.NET یک نسخه اجرایی کاملاً جدید ASP است که در جهت تولید HTML تدارک دیده شده است. درخواست‌هایی که برای دریافت فایل‌هایی با پسوند aspx. فرستاده می‌شوند، به aspnet-isapi.dll واگذار می‌شوند. این یک کامپوننت مدیریت شده است که در پاسخ به درخواست، به ایجاد نمونه‌هایی از کلاس‌های Net. می‌پردازد. چارچوب ASP.NET زیرمجموعه‌ای از کلاس‌های سیستم Net. است. در این زیرمجموعه، تعداد بی‌شماری کلاس گنجانده شده که به صورت پویا برای شما HTML هایی می‌سازند، مانند انواع و اقسام فهرست، جدول، فرم و منطق اعتبارسنجی (Validation)، کلاس‌هایی برای مدیریت state و caching و برای تنظیم و ایمن کردن برنامه و مجموعه‌ای از کلاس‌هایی برای حفظ سازگاری با ASP قدیمی. با استفاده از ویژوال استودیو دات‌نت، این زیرساختار در اختیار شما قرار می‌گیرد و در نتیجه کدی که شما می‌سازید می‌تواند خیلی شبیه به Active Sever Page قدیمی باشد، ولی هرچه بیشتر با سرویس‌ها و زیرساختار آن آشنا می‌شوید، کد شما بیشتر به یک فرم ویژوال بیسیک در می‌آید، با یک کد برای اجزای UI و فایلی جداگانه برای کد شما.

ASP.NET و ASP چند تفاوت جالب توجه دارند:

- صفحات ASP.NET «کامپایل» می‌شوند نه «تفسیر»: با دریافت اولین درخواست صفحه، یک فایل باینری اجرایی کامپایل می‌شود. این فایل، در یک حافظه cache در وب سرور ذخیره می‌شود، و درخواست‌های بعدی این صفحه، از این نسخه اجرایی برای پاسخ به درخواست استفاده می‌کنند.

- از آن‌جا که دیگر به مفسّر اسکریپت احتیاجی نیست، به ویژگی‌های زبانی ویژوال بیسیک دات‌نت دسترسی کامل داریم. در واقع به جای نوشتن اسکریپتی که HTML تولید کند، برنامه‌ای می‌نویسیم که با اشیاء کار می‌کند.

- چارچوب به روشنی کد را از محتوا تفکیک کرده، با ASP، چون HTML در هنگام interpert شدن صفحه تولید می‌شود، منطق صفحه شما باید در داخل صفحه در جایی گنجانده شود که می‌خواهید HTML ساخته شده با آن منطق در آن‌جا ظاهر شود، ولی با ASP.NET تا وقتی تمام کد صفحه شما به پایان اجرا نرسیده باشد، هیچ اچ‌تی‌ام‌ال‌ی تولید نمی‌شود. کل فرآیند تولید HTML در مرحلهٔ پرداخت (rendering) صفحه صورت می‌گیرد، که از خصوصیات اشیائی استفاده می‌کند که برای تولید HTML ایجاد کرده‌اید.

- چارچوب ASP.NET وضعیت را برای شما حفظ می‌کند: آیا تا به حال پیش آمده که برای اِعمال اعتبارسنجی (Validation) روی داده‌هایی که در یک فرم HTML وارد شده‌اند، مجبور بشوید دستورات را به سرور ارسال کنید؟ با چارچوب ASP.NET این «حفظ وضعیت» خود به خود برای شما انجام خواهد شد و بدون این‌که لازم باشد یک خط کد بنویسید، فیلدها مقادیر خود را حفظ می‌کنند. این امکان، نه تنها برای فیلدهای متنی ساده فراهم شده، بلکه روی لیست‌های SELECT، چک باکس‌ها، دکمه‌های رادیویی و هر نوع ورودی دیگر نیز عمل می‌کند.

- ASP.NET می‌تواند وقایع را در سرور اجرا کند: در ASP، به علت انبوه اسکریپتی که باید با HTML ترکیب شوند، رایج شده که یک ناحیه عملیاتی را بین چند صفحه تقسیم کنند. با ASP.NET، اما می‌توانیم event trap یا «واقعه نگارهایی» را در طرف سرور بنا کنیم. این چیزی شبیه به Remote Scripting است، ولی با HTML استاندارد کار می‌کند. بدین ترتیب، می‌توانیم یک دکمه HTML را روی صفحه بگذاریم و کاری کنیم واقعه Click آن در طرف سرویس تحریک شود.

- مدل وقایع در ASP.NET بسیار منسجم است: با ASP، اسکریپت به یک حالت «بالا به پایین» در صفحه اجرا می‌شود، ولی در ASP.NET، یک مدل رویدادی(Evend Model) وجود دارد و مهم‌تر این‌که، واقعه‌ای وجود دارد که وقتی صفحه شروع به بارشدن می‌کند، تحریک می‌شود. این خیلی شبیه به واقعه From Load در VB است.

در این مقاله قصد داشتم از دید یک برنامه‌نویس به بررسی کلی قابلیت‌ها و امتیازات فناوری دات‌نت بپردازم، اما برای نوشتن مثلاً یک وب‌سرویس با ویژوال بیسیک دات‌نت، واقعاً چه کار باید کرد؟ از کدام کلاس‌ها و اشیاء دات‌نت استفاده می‌شود؟ این‌ها همه سؤالاتی هستند که سعی می‌کنم در مقالات بعدی به طور عمیق و صد در صد کاربردی پاسخی برای آن‌ها ارائه دهم.