Common Configuration Differences Between Development and Production

Introduction

The last two tutorials walked through deploying a simple web application. The *Deploying Your Site Using an FTP Client* tutorial showed how to use a stand-alone FTP client to copy the necessary files from the development environment up to production. The preceding tutorial, *Deploying Your Site Using Visual Studio*, looked at deployment using Visual Studio's Copy Web Site tool and Publish option. In both tutorials every file in the production environment was a copy of a file on the development environment. However, it is not uncommon for configuration files in the production environment to differ from those in the development environment. A web application's configuration is stored in the `Web.config` file and typically includes information about external resources, such as database, web, and e-mail servers. It also spells out the behavior of the application in certain situations, such as the course of action to take when an unhandled exception occurs.

When deploying a web application it is important that the correct configuration information end up in the production environment. In most cases the `Web.config` file in the development environment cannot be copied to the production environment as-is. Instead, a customized version of `Web.config` needs to be uploaded to production. This tutorial briefly reviews some of the more common configuration differences; it also summarizes some techniques for maintaining different configuration information between the environments.

Typical Configuration Differences Between the Development and Production Environments

The `Web.config` file includes an assortment of configuration information for an ASP.NET application. Some of this configuration information is the same regardless of the environment. For instance, the authentication settings and URL authorization rules spelled out in the `Web.config` file's `<authentication>` and `<authorization>` elements are usually the same regardless of the environment. But other configuration information - such as information about external resources - typically differs depending on the environment.

Database connections strings are a prime example of configuration information that differs based on the environment. When a web application communicates with a database server it must first establish a connection, and that is achieved through a connection string. While it is possible to hard-code the database connection string directly in the web pages or the code that connects to the database, it is best to place it `Web.config`'s `<connectionStrings>` element so that the connection string information is in a single, centralized location. Oftentimes a different database is used during development than is

used in production; consequently, the connection string information must be unique for each environment.

> **Note:** Future tutorials explore deploying data-driven applications, at which point we'll dive into the specifics of how database connection strings are stored in the configuration file.

The intended behavior of the development and production environments differs substantially. A web application in the development environment is being created, tested, and debugged by a small group of developers. In the production environment that same application is being visited by many different simultaneous users. ASP.NET includes a number of features that help developers in testing and debugging an application, but these features should be disabled for performance and security reasons when in the production environment. Let's look at a few such configuration settings.

Configuration Settings That Impact Performance

When an ASP.NET page is visited for the first time (or the first time after it has changed), its declarative markup must be converted into a class and this class must be compiled. If the web application uses automatic compilation then the page's code-behind class needs to be compiled, too. You can configure an assortment of compilation options via the `Web.config` file's `<compilation>` element.

The debug attribute is one of the most important attributes in the `<compilation>` element. If the `debug` attribute is set to "true" then the compiled assemblies include debug symbols, which are needed when debugging an application in Visual Studio. But debug symbols increase the size of the assembly and impose additional memory requirements when running the code. Furthermore, when the `debug` attribute is set to "true" any content returned by `WebResource.axd` is not cached, meaning that each time a user visits a page they will need to re-download the static content returned by `WebResource.axd`.

> **Note:** `WebResource.axd` is a built-in HTTP Handler introduced in ASP.NET 2.0 that server controls use to retrieve embedded resources, such as script files, images, CSS files, and other content. For more information on how `WebResource.axd` works and how you can use it to access embedded resources from your custom server controls, see Accessing Embedded Resources Through a URL Using `WebResource.axd`.

The `<compilation>` element's `debug` attribute is usually set to "true" in the development environment. In fact, this attribute must be set to "true" in order to debug a web application; if you try to debug an ASP.NET application from Visual Studio and the `debug` attribute is set to "false", Visual Studio will display a message explaining that the application cannot be debugged until the `debug` attribute is set to "true" and will offer to make this change for you.

You should **never** have the `debug` attribute set to "true" in a production environment because of its impact on performance. For a more thorough discussion on this topic, refer to Scott Guthrie's blog post, Don't Run Production ASP.NET Applications With `debug="true"` Enabled.

Custom Errors and Tracing

When an unhandled exception occurs in an ASP.NET application it bubbles up to the runtime at which point one of three things happens:

- A generic runtime error message is displayed. This page informs the user that there was a runtime error, but does not provide any details about the error.
- An exception details message is displayed, which includes information on the exception that was just thrown.
- A custom error page is displayed, which is an ASP.NET page that you create that displays any message you desire.

What happens in the face of an unhandled exception depends on the `Web.config` file's `<customErrors>` section.

When developing and testing an application it helps to see details of any exception in the browser. However, showing exception details in an application on production is a potential security risk. Moreover, it's unflattering and makes your website look unprofessional. Ideally, in the event of an unhandled exception a web application in the development environment will show the exception's details while the same application in production will show a custom error page.

> **Note:** The default `<customErrors>` section setting shows the exception details message only when the page is being visited through localhost, and shows the generic runtime error page otherwise. This isn't ideal, but it's assuring to know that the default behavior doesn't reveal exception details to non-local visitors. A future tutorial examines the `<customErrors>` section in more detail and shows how to have a custom error page shown when an error occurs in production.

Another ASP.NET feature that is useful during development is tracing. Tracing, if enabled, records information about each incoming request and provides a special web page, `Trace.axd`, for viewing recent request details. You can turn on and configure tracing via the `<trace>` element in `Web.config`.

If you enable tracing make sure that it is disabled in production. Because the trace information includes cookies, session data, and other potentially sensitive information, it is important to disable tracing in production. The good news is that, by default, tracing is disabled and the `Trace.axd` file is only accessible through localhost. If you change these default settings in development make sure that they are turned back off in production.

Techniques for Maintaining Separate Configuration Information

Having different configuration settings in the development and production environments complicates the deployment process. In the previous two tutorials the deployment process involved copying all of the necessary files from development to production, but that approach only works if the configuration information is the same in both environments. There are a variety of techniques for deploying an application with varying configuration information. Let's catalog some of these options for hosted web applications.

Manually Deploying the Production Environment Configuration File

The simplest approach is to maintain two versions of the `Web.config` file: one for the development environment and one for the production environment. Deploying a site to production entails copying all of the files to the production server in the development environment *except* for the `Web.config` file. Instead, the production environment-specific `Web.config` file would be copied to production.

This approach is not very sophisticated, but it is easy to implement because configuration information changes infrequently. It works best for applications with a small development team that are hosted on a single web server and whose configuration information is infrequently changed. It's most tenable when manually deploying the application files using a stand-alone FTP client. When using Visual Studio's Copy Web Site tool or Publish option you'll need to first swap out the deployment-specific `Web.config` file with the production-specific one before deploying, and then swap them back after deployment completes.

Change the Configuration During the Build or Deployment Process

The discussions thus far have assumed an ad-hoc build and deployment process. Many larger software projects have more formalized processes that make use of open-source, home-grown, or third-party tools. For such projects you can likely customize the build or deployment process to appropriately modify the configuration information before it is pushed to production. If you build your web application using MSBuild, NAnt, or some other build tool, you can likely add a build step to modify the `Web.config` file to include the production-specific settings. Or your deployment workflow could programmatically connect to the source control server and retrieve the appropriate `Web.config` file.

The actual approach for getting the appropriate configuration information to production varies widely based on your tools and workflow. As such, we will not delve into this topic further. If you are using a popular build tool like MSBuild or NAnt you can find deployment articles and tutorials specific to these tools through a web search.

Managing Configuration Differences via the Web Deployment Project Add-In

In 2006 Microsoft released the Web Development Project Add-In for Visual Studio 2005. An Add-In for Visual Studio 2008 was released in 2008. This Add-In allows for ASP.NET developers to create a separate Web Deployment Project alongside their web application project that, when built, explicitly compiles the web application and copies the files to deploy to a local output directory. The Web Application Project uses MSBuild behind the scenes.

By default, the development environment's `Web.config` file is copied to the output directory, but you can setup the Web Deployment Project to customize the configuration information that gets copied to this directory in the following ways:

- Via `Web.config` file section replacement, in which you specify the section to replace and an XML file that contains the replacement text.
- By providing a path to an external configuration source file. With this option selected, the Web Deployment Project copies a particular `Web.config` file to the output directory (rather than the `Web.config` file used in the development environment).
- By adding custom rules to the MSBuild file used by the Web Deployment Project.

To deploy the web application build the Web Deployment Project and then copy the files from the project's output folder to the production environment.

To learn more about using the Web Deployment Project check out this Web Deployment Projects article from the April 2007 issue of MSDN Magazine, or consult the links in the Further Reading section at the end of this tutorial.

> **Note:** You cannot use the Web Deployment Project with Visual Web Developer because the Web Deployment Project is implemented as a Visual Studio Add-In and the Visual Studio Express Editions (including Visual Web Developer) do not support Add-Ins.

Summary

The external resources and behavior of a web application in development are typically different than when the same application is in production. For instance, database connection strings, compilation options, and the behavior when an unhandled exception occurs commonly differ between environments. The deployment process must accommodate these differences. As we discussed in this tutorial, the simplest approach is to manually copy an alternate configuration file to the production environment. More elegant solutions are possible when using the Web Deployment Project Add-In or with a more formalized build or deployment process that can accommodate such customizations.

Happy Programming!

Further Reading

For more information on the topics discussed in this tutorial, refer to the following resources:

- Connection Strings Explained
- Database Connection Strings @ ConnectionStrings.com
- Don't Run Production ASP.NET Applications with `debug="true"` Enabled
- Gracefully Responding to Unhandled Exceptions - Displaying User-Friendly Error Pages
- How Do I: Use a Visual Studio 2008 Web Deployment Project?
- Key Configuration Settings When Deploying a Database
- Visual Studio 2008 Web Deployment Projects Download | Visual Studio 2005 Web Deployment Projects Download
- VS 2008 Web Deployment Projects | VS 2008 Web Deployment Project Support Released
- Web Deployment Projects

About the Author

Scott Mitchell, author of multiple ASP/ASP.NET books and founder of 4GuysFromRolla.com, has been working with Microsoft Web technologies since 1998. Scott works as an independent consultant, trainer, and writer. His latest book is *Sams Teach Yourself ASP.NET 3.5 in 24 Hours*. Scott can be reached at mitchell@4guysfromrolla.com or via his blog at http://ScottOnWriting.NET.

Special Thanks To...

This tutorial series was reviewed by many helpful reviewers. Lead reviewer for this tutorial was Stacy Park. Interested in reviewing my upcoming MSDN articles? If so, drop me a line at mitchell@4GuysFromRolla.com.