KAPITOLA 5

Aplikace ASP.NET

V tradičním desktopovém programování se aplikací rozumí spustitelný soubor a podpůrné soubory, které se k němu vztahují. Například typická aplikace Windows se skládá z hlavního spustitelného souboru (EXE), podpůrných komponent (typicky DLL) a dalších zdrojů, jakou jsou databáze a konfigurační soubory. Aplikace ASP.NET používají dost odlišný model.

Na té nejzákladnější úrovni je aplikace ASP.NET kombinací souborů, stránek, ovladačů (handlers), modulů a spustitelného kódu, který se dá vyvolat z nějakého virtuálního adresáře (a z jeho podadresářů) na nějakém webovém serveru. V této kapitole se dozvíte, proč taková odlišnost existuje, a podíváte se poněkud blíže na to, jak se aplikace ASP.NET konfiguruje a rozmisťuje. Také se dozvíte, jak se s aplikací ASP.NET používají komponenty a ovladače HTTP.

ZMĚNY VE WEBOVÉ APLIKACI V .NET 2.0

Model webové aplikace zůstal v ASP.NET 2.0 v podstatě stejný. Nejvýznamnější změnou je zdokonalení konfiguračního modelu, který se nyní honosí programovatelným API a grafickým rozhraním webové stránky. Změny, s nimiž se seznámíte, jsou uvedeny v následujícím výčtu. Uvádíme je v pořadí, v jakém se v kapitole probírají:

- **Struktura adresářů aplikace**. ASP.NET 1.x měl jeden speciální adresář webové aplikace – adresář Bin, ve kterém sídlily zkompilované assembly. ASP.NET 2.0 jich přidává několik navíc: pro zdrojový kód, lokalizovatelné zdroje, definice prohlížeče, motivy a další.
- **Konfigurační API.** Nyní můžete pomocí šikovné sady tříd číst a zapisovat téměř jakékoliv informace konfiguračního souboru.
- **WAT**. Nový nástroj pro administraci webu (Website Administration Tool) prezentuje pomocí nového konfiguračního API snadno zvládnutelné rozhraní v podobě webové stránky, v němž se dá nakonfigurovat webová aplikace.
- **Sekce konfigurace lze zašifrovat**. Konfigurační soubory často obsahují citlivá data. Nyní je můžete ochránit tím, že jakoukoliv sekci je možné zašifrovat.

Jste-li příležitostným vývojářem ASP.NET 1.x, můžete číst tuto kapitolu tak, že se soustředíte hlavně na právě zmíněné novinky.

Anatomie aplikace ASP.NET

Rozdíl mezi aplikací ASP.NET a bohatě vybavenou klientskou aplikací se vám hodně objasní, když se podíváme na model vykonávání aplikace ASP.NET. Na rozdíl od aplikace Windows, konečný uživatel nikdy přímo nespouští aplikaci ASP.NET. Uživatel spustí nějaký prohlížeč, například Internet Explorer, a požaduje přes HTTP nějakou konkrétní URL (jako je `http://www.mujweb.com/mojestranka.aspx`). Požadavek dorazí na nějaký webový server. Když aplikaci ladíte ve Visual Studiu, používáte výhradně místní testovací server. Když aplikaci rozmístíte, používáte webový server IIS, což se popisuje v kapitole 18.

Webový server nezná žádný pojem separátní aplikace – předá prostě požadavek zpracovatelskému procesu (worker process) ASP.NET. Zpracovatelský proces ASP.NET však pečlivě odděluje vykonávání kódu do domén aplikace založených na virtuálním adresáři. Webové stránky a webové služby, jejichž hostitelem je týž virtuální adresář (nebo některý z jeho podadresářů) se vykonávají v téže doméně. Webové stránky a webové služby, které jsou v různých virtuálních adresářích, se vykonávají v separátních doménách.

POZNÁMKA Virtuální adresář je prostě adresář, který je vystavený prostřednictvím nějakého webového serveru. To, jak se vytvářejí virtuální adresáře, se dozvíte v kapitole 18. Používáte-li testovací server Visual Studia, s adresářem webového projektu se zachází jako s virtuálním adresářem. Jedinou výjimkou je, že testovací server výhradně podporuje místní připojení (tedy požadavky iniciované z aktuálního počítače).

Doména aplikace

Doména aplikace je ekvivalent .NET pro proces – je to jistá hranice vynucená CLR, která zajišťuje, že jedna aplikace nemůže ovlivnit jinou aplikaci (nebo vidět její data nacházející se v paměti). Přímé důsledky modelu domény aplikace jsou následující:

- Všechny webové stránky a webové služby v jedné webové aplikaci sdílejí stejné paměťové zdroje, jako jsou globální data aplikace, data relace jednotlivých uživatelů a data uložená do cache. Tyto informace nejsou přímo dostupné jiným ASP.NET nebo ASP aplikacím.
- Všechny webové stránky a webové služby v jedné webové aplikaci sdílejí stejné jádro konfiguračních nastavení. Některá konfigurační nastavení však můžete přizpůsobovat v jednotlivých podadresářích téhož virtuálního adresáře. Například pro webovou aplikaci můžete nastavit pouze jediný mechanismus autentizace (ověřování totožnosti uživatelů), bez ohledu na to, kolik podadresářů má. Můžete ovšem v každém adresáři nastavit jiná pravidla autorizace, abyste mohli lépe specifikovat, kdo má mít přístup těm či oněm skupinám stránek a kdo ne.
- Všechny webové aplikace vyvolávají v různých etapách globální události aplikace (když se úplně poprvé vytvoří doména aplikace, když se zničí atd.), k nimž můžete připojovat ovladače událostí. Ty budou na události aplikace reagovat pomocí kódu uvedeného v globálním souboru .asax, který bude uložen ve virtuálním adresáři aplikace.

Jinak řečeno – virtuální adresář je základní seskupující struktura, která odděluje aplikaci ASP.NET. Legitimní ASP.NET aplikace může mít klidně pouze jedinou webovou stránku (soubor .aspx), nebo webovou službu (soubor .asmx). Obecně však může aplikace ASP.NET obsahovat všechny následující ingredience:

- **Webové stránky (soubory .aspx)**. Základní stavební bloky jakékoliv aplikace ASP.NET.
- **Webové služby (soubory .asmx)**. Umožňují sdílet užitečné funkce s aplikacemi na jiných počítačích a jiných platformách.

- **Soubory kódu v pozadí (code-behind files)**. V závislosti na tom, jaký model kódu používáte, můžete také mít oddělené soubory se zdrojovým kódem. Jsou-li napsány v jazyku C#, budou mít příponu .cs.
- **Konfigurační soubor (web.config)**. Obsahuje spoustu nastavení na úrovně aplikace. Pomocí těchto nastavení se konfiguruje všechno možné – např. věci týkající se bezpečnosti, ladění, či správy stavu (state management).
- **Global.asax**. Obsahuje ovladače událostí, jež reagují na globální události aplikace (jako třeba událost, když aplikace startuje úplně poprvé).
- **Další komponenty**. Jedná se o zkompilované assembly obsahující separátní komponenty, které jste vyvinuli sami, nebo se jedná komponenty s užitečnou funkcionalitou od jiných výrobců. Komponenty umožňují oddělovat obchodní logiku od logiky přístupu k datům a vytvářet vlastní ovládací prvky.

Virtuální adresář může dále obsahovat značné množství dalších zdrojů, které bude webová aplikace ASP.NET používat. Patří mezi ně stylové předpisy (stylesheets), obrázky, soubory XML atd. Kromě toho se dá model ASP.NET rozšiřovat tím, že vyvinete specializované komponenty známé jako ovladače HTTP a moduly HTTP, které se mohou zapojit do vaší aplikace a participovat na zpracování webových požadavků ASP.NET.

POZNÁMKA V jednom virtuálním adresáři je možné mít i typy souborů, které vlastní odlišná rozšíření ISAPI. Může se například stát, že tam budete mít směsici souborů .aspx a .asp. Složitějším příkladem je to, pokud mapujete soubory .aspx webových stránek na verzi 1.1 ASP.NET a soubory .asmx webových služeb na verzi 2.0. V těchto situacích bude jeden virtuální adresář odpovídat více než jedné aplikaci. Aplikace pak jsou přístupné prostřednictvím jediného virtuálního webového adresáře. Každou aplikaci však bude zprostředkovávat jiné rozšíření ISAPI.

Doba života aplikace

ASP.NET používá techniku tzv. lenivé inicializace (lazy initialization) při vytváření domén aplikace. Tím je řečeno to, že doména aplikace pro webovou aplikaci vytvoří až tehdy, když úplně poprvé dorazí požadavek na nějakou stránku nebo službu z této aplikace.

Doména aplikace může být "shozena" z nejrůznějších příčin, včetně té, že byl "shozen" samotný webový server. Běžněji se ale aplikace samy restartují v nových doménách aplikace kvůli tomu, že reagují na nějakou chybovou podmínku nebo na nějakou změnu v konfiguraci. Například podle toho, jaká jsou nastavení v konfiguračním souboru machine.config (tedy na úrovni stroje), může být aplikace ASP.NET pravidelně recyklována, když se narazí na jisté limity. Tento model byl navržen proto, aby byly aplikace stále v dobrém zdravotním stavu, a aby se daly detekovat takové charakteristiky, které mohou indikovat vznik nějakých potíží. V závislosti na tom, jaká máte v souboru machine.config nastavení, se mohou domény aplikace recyklovat podle toho, jak dlouho se provozují, podle počtu požadavků ve frontě, nebo podle množství používané paměti (více to popisujeme v kapitole 18).

Když změníte aplikaci, ASP.NET automaticky recykluje její domény aplikace. Například jste nějak modifikovali soubor web.config. Jiným příkladem může být to, že nahradíte nějaký existující soubor webové stránky nebo DLL soubor assembly nějakým jiným souborem. V obou případech ASP.NET vytvoří novou doménu aplikace pro potřeby zpracování všech budoucích požadavků, přičemž zachová existující doménu aplikace

na tak dlouho, dokud se nedokončí zpracování všech dosud nevyřízených požadavků (včetně požadavků zařazených do fronty).

TIP Doménu webové aplikace je možné programátorským způsobem ukončit pomocí statické metody HttpRuntime.UnloadAppDomain(). (Aplikace se pak automaticky nastartuje při příštím požadavku.) Tato technika se sice používá pouze výjimečně, nicméně se může hodit, pokud na tomtéž serveru hostujete značný počet webových aplikací, ze kterých se některé aplikace používají pouze ojediněle. V takovém případě může být zátěž spočívající v tom, že stále udržujete všechny domény aplikace v provozu, příliš velká, takže se vyplatí, když málo frekventované domény aplikace ukončíte, abyste zvýšili rychlost obsluhy následujících požadavků.

Aktualizace aplikací

Jednou z nejpozoruhodnějších schopností modelu vykonávání ASP.NET je to, že můžete webovou aplikaci aktualizovat (update), aniž byste museli restartovat webový server, nebo se starat o to, zdali tím nějak neuškodíte již existujícím klientům. To znamená, že ve virtuálním adresáři můžete přidávat, nahrazovat nebo odstraňovat soubory kdykoliv, kdy to potřebujete. ASP.NET pak přejde k nové doméně aplikace stejným způsobem, jako když modifikujete její konfigurační soubor web.config.

Mít možnost kdykoliv aktualizovat jakoukoliv část aplikace, aniž by se tím přerušily existující požadavky, je velmi mocná schopnost. Je však důležité, abyste porozuměli architektuře, která to umožňuje. Mnozí vývojáři dělají tu chybu, když předpokládají, že je to schopnost CLR, která umožňuje, aby ASP.NET mohlo hladce přejít k nové doméně aplikace. Realitou je ale, že CLR vždy uzamyká soubory assembly při jejich vykonávání. Aby se ASP.NET s touto situací vypořádalo, ve skutečnosti nepoužívá soubory z virtuálního adresáře. Během procesu kompilace používá jinou techniku, která se nazývá jako stínová kopie (shadow copy). S její pomocí si ASP.NET vytvoří kopie vašich souborů, které jsou umístěny v adresáři c:\[WinDir]\Microsoft.NET\[Verze]\Temporary ASP.NET Files. Zpracovatelský proces ASP.NET pak assembly načítá z tohoto adresáře, což znamená, že uzamčené jsou tyto assembly.

Druhou částí příběhu je to, že ASP.NET je schopno detekovat změnu originálních souborů. Tento detail je dosti přímočarý – spoléhá se při něm na schopnost operačního systému Windows sledovat adresáře a soubory a okamžitě odesílat informace o změnách. ASP.NET si udržuje aktivní seznam všech assembly načtených uvnitř konkrétní domény aplikace, a pomocí monitorovacího kódu dává pozor na změny, na které se následně reaguje.

POZNÁMKA ASP.NET může používat soubory uložené v GAC, což je úschovna na úrovni počítače pro komponenty, které obsahují věci pro běžnou denní potřebu, jako jsou assembly pro celou knihovnu tříd .NET Frameworku. Do GAC můžete dávat i své vlastní assembly, ale webové aplikace se obvykle budou snadněji rozmisťovat a udržovat, když je tam dávat nebudete.

Struktura adresáře aplikace

Každá webová aplikace by měla mít dobře a pečlivě naplánovanou strukturu adresářů. Nezávisle na tom, jako strukturu adresáře si navrhnete vy sami, ASP.NET specifikuje několik adresářů, které mají speciální význam. ASP.NET 1.x používálo pouze jediný takový adresář – adresář Bin. ASP.NET 2.0 kromě něj používá několik dalších, ty jsou popsány v tabulce 5-1.

Tabulka 5-1. Speciální adresáře ASP.NET.

Adresář	Popis
Bin	Obsahuje všechny předem zkomplikované assembly .NET (obvykle DLL), které používá webová aplikace ASP.NET. Do těchto assembly patří předem zkompilované třídy webových stránek a webových služeb, a také jiné assembly, na které se tyto třídy odkazují.
App_Code	Obsahuje soubory zdrojového kódu, které se pro potřeby vaší aplikace dynamicky kompilují. Tyto soubory kódu jsou obvykle separátní komponenty, jako například nějaká přihlašovací komponenta nebo nějaká knihovna pro přístup k datům. Zkompilovaný kód se nikdy neobjeví v adresáři Bin, protože jej ASP.NET umisťuje do dočasných adresářů, které používá pro dynamickou kompilaci.
App_GlobalResources	Do tohoto adresáře se ukládají globální zdroje, k nimž mají přístup všechny stránky webové aplikace. O zdrojích a lokalizacích se více dozvíte v kapitole 17.
App_LocalResources	Slouží stejnému účelu jako App_GlobalResources, s tím rozdílem, že zdroje budou dostupné pouze té stránce, pro kterou jsou určeny. O zdrojích a lokalizacích se více dozvíte v kapitole 17.
App_WebReferences	Do tohoto adresáře se ukládají odkazy na webové služby, které webová aplikace používá. Patří sem i soubory WSDL a dokumenty DISCO. Webové služby se probírají v šesté části knihy.
App_Data	Adresář vyhrazený pro úložiště dat, včetně databázových souborů SQL serveru 2005 Express a souborů XML. Soubory s daty si pochopitelně můžete ukládat i do jiných adresářů.
App_Browsers	Obsahuje definice prohlížečů uložené v XML souborech. Tyto soubory XML definují schopnosti webových prohlížečů pro různé akce. Přestože to ASP.NET dělá globálně (vzhledem k celému počítači), adresář App_Browsers umožňuje nakonfigurovat toto chování pro různé webové aplikace. Další informace o tom, jak ASP.NET rozlišuje jednotlivé prohlížeče, jsou uvedeny v kapitole 27.
App_Themes	Do tohot adresáře se ukládají motivy, které používá webová aplikace. Motivy se probírají v kapitole 15.

Soubor aplikace global.asax

Soubor global.asax umožňuje vytvářet ovladače událostí, kteří reagují na globální události. Uživatelé nikdy nežádají o soubor global.asax přímo. Děje se to jinak. Soubor global.asax vykonává svůj kód automaticky (jako reakci na jisté události aplikace). Soubor global.asax poskytuje podobnou službu, jakou poskytoval soubor global.asa v klasických aplikacích ASP.

Kód se do souboru global.asax píše podobně, jako když píšete kód do nějakého webového formuláře. Rozdíl je v tom, že global.asax neobsahuje žádné značky HTML nebo ASP.NET. Obsahuje metody s konkrétními předdefinovanými názvy. Například následující soubor global.asax reaguje na událost Application.EndRequest, ke které dojde těsně předtím, než se stránka odešle uživateli:

```
<script language="C#" runat="server">
```

```
  protected void Application_OnEndRequest()
  {
    Response.Write("<hr />Tato stránka byla obsloužena v " +
    DateTime.Now.ToString());
  }
</script>
```

Přestože to není v souboru global.asax vyjádřeno, každý soubor global.asax definuje metody pro jedinou třídu – třídu aplikace. Třída Application je odvozená z HttpApplication, což znamená, že kód má přístup ke všem jejím veřejným a chráněným členům. V uvedené ukázce se používá objekt Response, který je poskytován jako zabudovaná vlastnost třídy HttpApplication.

Ovladač události v ukázce zapíše zápatí stránky, v němž uvede datum a čas, kdy byla stránka vytvořena. Protože reaguje na událost Application.EndRequest, vykoná se pokaždé, když se stránka požaduje, poté, co se dokončí zpracování veškerého kódu zpracovávajícího události v této stránce.

Stejně jako u webových formulářů, i zde můžete rozdělit obsah souboru global.asax do dvou souborů. V jednom bude deklarace souboru, druhý bude obsahovat kód. Protože však pro soubory global.asax není k dispozici žádný designérský povrch (design surface), toto rozdělení se nevyžaduje. Visual Studio vám nenabídne volbu vytvořit soubor global.asax s oddělenou třídou kódu v pozadí.

Soubor global.asax file sice není povinný, ale webová aplikace nesmí mít více než jeden soubor global.asax, který musí sídlit v kořenovém adresáři aplikace, nikoliv v nějakém podadresáři. Chcete-li do projektu přidat soubor global.asax, vyberte Website > Add New Item, a zvolte šablonu Global Application Class. Když Visual Studio přidává soubor global.asax, vloží do něj prázdné ovladače nejčastěji používaných událostí aplikace. Vy pak pouze přidáte do patřičných metod kód, který má vykonávat.

Stojí za zmínku, že ovladače událostí aplikace se nepřiřazují stejným způsobem jako běžné ovladače událostí ovládacích prvků. Obvyklý způsob, jakým se rozpoznává ovladač události aplikace, je ten, že se použije dohodnutý název metody. Vytvoříte-li například chráněnou metodu s názvem Application_OnEndRequest(), ASP.NET ji bude volat automaticky, jakmile dojde k události Application.EndRequest.

ASP.NET vytváří fond objektů aplikace v okamžiku, když se poprvé načte doména vaší aplikace, přičemž jeden z objektů aplikace pak použije k obsluze požadavků. Velikost fondu závisí na systému a počtu dostupných vláken, obvykle ale bývá v rozsahu od 1 do 100 instancí. Každý požadavek získá výhradní přístup k jednomu z těchto objektů aplikace. Když daný požadavek skončí, bude objekt opětovně využit. V průběhu různých etap zpracování aplikace ASP.NET volá odpovídající metody, které pak spustí váš kód. Samozřejmě, má-li metoda špatný název, vaše implementace nebude zavolána – kód se bude prostě ignorovat.

POZNÁMKA Globální třída aplikace, kterou používá soubor global.asax, by měla být vždy bezstavová (stateless). To z toho důvodu, protože jakmile se objekty aplikace stanou dostupnými, opětovně se používají pro různé požadavky. Nastavíte-li v jednom požadavku hodnotu nějaké členské proměnné, mohla by se objevit i v jiném požadavku. Neexistuje však žádný způsob, jak specifikovat, jak se to má stát, nebo který z požadavků má získat tu či onu instanci objektu aplikace. Abyste se vyhnuli případným zádrhelům, nepoužívejte členské proměnné, pokud nejsou statické (více to probereme v kapitole 6).

Události aplikace

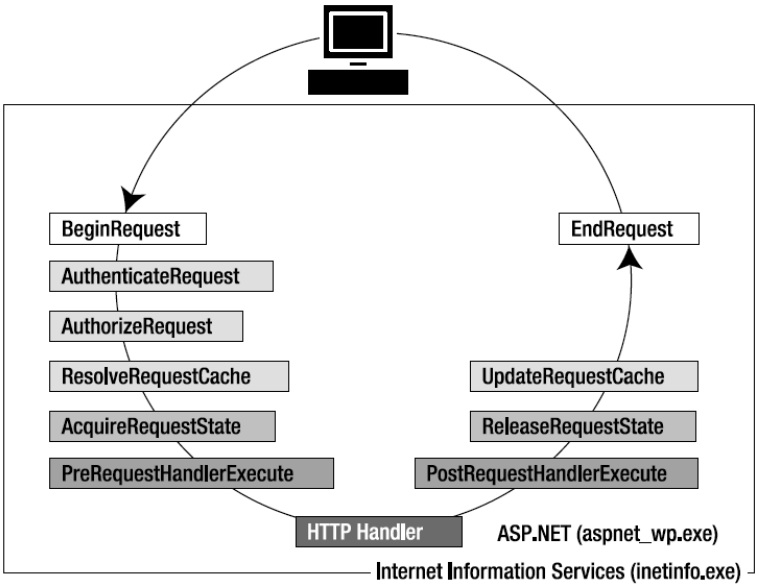
V souboru global.asax se zpracovávají dva druhy událostí:

- Události, k nimž dochází vždy, pro každý požadavek. Vztahují se k požadavku a k odpovědi na něj.
- Události, k nimž dochází pouze za určitých podmínek.

Sled událostí aplikace je následující:

1. **Application_BeginRequest()**. Zavolá se při startu každého požadavku, včetně požadavků na soubory, které nejsou webovými formuláři, jako třeba webové služby.
2. **Application_AuthenticateRequest()**. Zavolá se těsně předtím, než se provede autentizace. Je to přesně ten bod, kde můžete vskočit do zpracování s vaší vlastní autentizační logikou.
3. **Application_AuthorizeRequest()**. Zavolá se poté, co byl uživatel autentizován (tedy identifikován; byla ověřena jeho totožnost), a kdy je třeba určit jeho oprávnění. Pomocí této metody můžete přiřazovat speciální oprávnění.
4. **Application_ResolveRequestCache()**. Běžně se používá v součinnosti s ukládáním výstupu do cache (output caching). Když se výstup ukládá do cache (popisujeme to v kapitole 11), je realizovaný HTML kód webového formuláře opětovně využíván, aniž by bylo nutné vykonávat nějaký váš kód. Přesto se tento ovladač spustí.
5. **V tomto okamžiku se požadavek podá patřičnému ovladači.** Například u požadavku na nějaký webový formulář je to okamžik, kdy se stránka zkompiluje (pokud je to nutné) a vytvoří se její instance.
6. **Application_AcquireRequestState()**. Zavolá se těsně předtím, než se pro klienta získají informace specifické pro relaci a naplní se jimi kolekce Session.
7. **Application_PreRequestHandlerExecute()**. Zavolá se předtím, než patřičný ovladač HTTP vykoná požadavek.
8. **V tomto okamžiku patřičný ovladač vykoná požadavek.** Pokud se například jedná o požadavek na webový formulář, vykoná se kód stránky pro zpracování událostí, a stránka se realizuje do HTML.
9. **Application_PostRequestHandlerExecute()**. Zavolá se ihned poté, co se požadavek zpracoval.
10. **Application_ReleaseRequestState()**. Zavolá se, když je potřeba serializovat informace specifické pro relaci (session) z kolekce Session, aby byly dostupné pro příští požadavek.
11. **Application_UpdateRequestCache()**. Zavolá se těsně předtím, než se informace předají do cache pro výstup. Pokud například pro nějakou webovou stránku zapnete ukládání výstupu do cache (output caching), ASP.NET v tomto okamžiku vloží realizovanou HTML stránku do cache.
12. **Application_EndRequest()**. Zavolá se na konci požadavku – těsně předtím, než se uvolní a znovu vyžádají objekty. Toto je vhodná chvíle pro čistící (cleanup) kód.

Proces zpracování jediného požadavku vidíte na obrázku 5-1.



Obrázek 5-1. *Události aplikace.*

Některé z událostí se neodpalují při každém požadavku:

- **Application_Start().** Zavolá se, když aplikace startuje poprvé a když se vytvořila doména aplikace. Tento ovladač události je šikovným místem pro inicializační kód týkající se celé aplikace. Například byste zde mohli načíst a uložit do cache taková data, která se v průběhu doby života aplikace nebudou měnit, jako třeba stromy navigace, statické katalogy výrobků atd.
- **Session_Start()**. Zavolá se pokaždé, když začne nějaká nová relace. Často se používá pro inicializaci informací specifických pro daného uživatele. Relace a správa stavu se probírají v šesté kapitole.
- **Application_Error()**. Zavolá se, jakmile dojde v aplikaci k nějaké nezpracované výjimce.
- **Session_End()**. Zavolá se pokaždé, když končí relace uživatele. Relace končí, když ji explicitně uvolní váš kód, nebo když vyprší platnost relace, kdy po stanovenou dobu nedorazily žádné další požadavky (obvykle je to 20 minut). Metoda se obvykle používá pro údržbové práce na datech, kterých se to týká.
- **Application_End()**. Zavolá se těsně předtím, než aplikace skončí. Konec aplikace může nastat třeba proto, že se restartovalo IIS, nebo proto, že se aplikace přenáší do nové domény aplikace, jako reakce na aktualizaci souborů, nebo také z toho důvodu, že se jedná o proces recyklace nastavení.
- **Application_Disposed()**. Zavolá se za nějakou chvíli poté, co byla aplikace ukončena a svoz odpadků .NET (garbage collector) se chystá vyžádat si tu část paměti, kterou zabírala aplikace. Zde už je pozdě na důležité údržbové akce, nicméně tuto událost můžete použít jako opatření poslední záchrany při nějakém selhání, pro ověření, zdali byly uvolněny kritické zdroje.

Pomocí událostí aplikace se běžně provádějí inicializace, údržbové akce, přihlašování uživatelů, profilování nebo řešení různých problémů. Nemyslete si ale, že vaše aplikace budou muset používat globální události aplikace. Mnohé aplikace ASP.NET soubor global.asax nepoužívají vůbec.

TIP Soubor global.asax není jediným možným místem, kde se dá reagovat na globální události aplikace. Můžete například vytvořit vlastní moduly, které budou participovat při zpracování webových požadavků, což probereme později v této kapitole, v oddílu popisujícím rozšíření kanálu HTTP.

Ukázka zpracování událostí aplikace

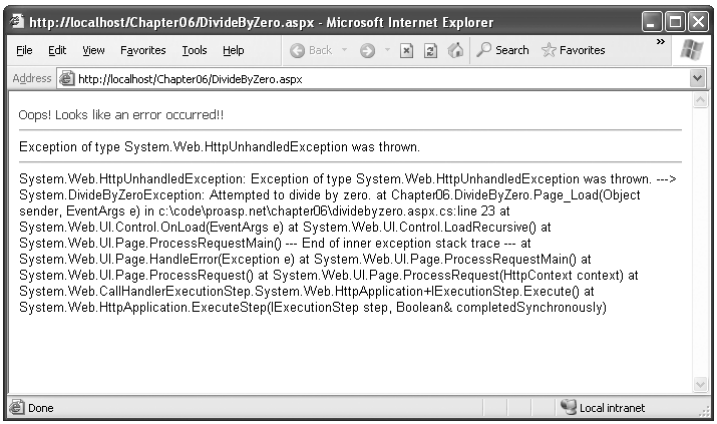
Následující webová aplikace používá soubor global.asax, který reaguje na metodu Application_Error. Chyba je zachycena a informace o ní jsou zobrazeny v předdefinovaném formátu.

```
<script language="C#" runat="server">
  protected void Application_Error(Object sender, EventArgs e)
  {
    Response.Write("<b>");
    Response.Write("Oops! Looks like an error occurred!!</b><hr />");
    Response.Write(Server.GetLastError().Message.ToString());
    Response.Write("<hr />" + Server.GetLastError().ToString());
    Server.ClearError();
  }
</script>
```

Abyste mohli tento ovladač události aplikace otestovat, musíte vytvořit webovou stránku, která způsobí chybu. Podívejte se na ukázku, kde se chyba vygeneruje při pokusu dělit nulou (v okamžiku načítání stránky):

```
protected void Page_Load(object sender, EventArgs e)
{
  int i = 0;
  int j = 1;
  int k = j/i;
}
```

Pokud budete tuto stránku požadovat, uvidíte to, co ukazuje obrázek 5-2.



Obrázek 5-2. *Zachycení nezpracované chyby.*

Není ovšem typické řídit vzhled webových stránek metodou Application_Error(), protože neposkytuje dostatek flexibility, kterou potřebujete při výskytu různých druhů chyb (aniž byste museli psát bolestně pracnou podmínkovou logiku). Patrně to uděláte tak, že si pomocí souboru web.config nakonfigurujete vlastní chybové stránky (popisujeme se to v příštím oddílu). Application_Error() však může být nesmírně prospěšná, chcete-li protokolovat výskyt nějakých chyb pro potřeby budoucích referencí, nebo informace o nich dokonce odesílat e-mailem systémovému administrátorovi. A vskutku – v mnoha situacích budete tuto techniku potřebovat, například tehdy, nebude-li dostupný objekt Response. Dvě z těchto situací jsou metody Application_Start() a Application_End().

Konfigurace ASP.NET

Konfigurace se v ASP.NET řídí pomocí souborů XML. V těchto konfiguračních souborech jsou uloženy nejenom všechny informace, které jsou potřebné pro nastavení klíčových nastavení ASP.NET, ale také i vaše vlastní nastavení, která jsou specifická pro vaši aplikaci.

Konfigurační soubory ASP.NET mají oproti konfiguračním souborům v klasickém ASP několik následujících předností:

- **Nikdy nejsou uzamčeny**. Jak jsme uvedli na začátku kapitoly, konfigurační nastavení můžete aktualizovat kdykoliv, když to potřebujete, přičemž ASP.NET hladce přejde do nové domény aplikace.
- **Snadno se k nim přistupuje a snadno se replikují**. Za předpokladu, že máte přidělena patřičná síťová oprávnění, můžete konfigurační soubor modifikovat ze vzdáleného počítače (nebo jej nahradit tak, že pomocí FTP nahraje jeho novou verzi). Konfigurační soubor můžete také zkopírovat, pokud chcete použít identická nastavení v jiné aplikaci nebo na jiném webovém serveru, který provozuje stejnou aplikaci v rámci webové farmy.
- **Snadno se editují a chápou**. Nastavení v konfiguračním souboru jsou srozumitelná pro lidi, což znamená, že je možné je jak editovat, tak i pochopit, aniž by byl k tomu potřebný nějaký specializovaný konfigurační nástroj.

S ASP.NET se nemusíte starat o konfiguraci metabáze IIS nebo o restarty webového serveru. Přesto však existuje několik věcí, které v souboru web.config dělat nemůžete. Nemůžete například vytvořit nebo odstranit virtuální adresář. Obdobně také nemůžete změnit mapování souborů. Chcete-li, aby služba ASP.NET zpracovávala požadavky i jiných typů souborů (jako třeba HTML nebo vaše vlastní typy souborů, které si nadefinujete), musíte k tomu použít manažera IIS, což je více popsáno v kapitole 18.

Soubor machine.config

Konfigurace začíná souborem, který se jmenuje machine.config, a sídlí v adresáři c:\[WinDir]\Microsoft.NET\Framework\[Verze]\Config. V souboru machine.config se definují podporované sekce konfiguračního souboru, konfiguruje se zpracovatelský proces ASP.NET a registrují se poskytovatelé, kteří se budou používat pro pokročilé schopnosti, jako jsou profily, členství a bezpečnost založená na rolích.

V ASP.NET 2.0 byl soubor machine.config dramaticky zprůhledněn. Aby se proces inicializace optimalizoval, mnohé z výchozích nastavení, které se dříve používaly v souboru machine.config, se nyní inicializují programátorsky. Pořád si však můžete vyhledat všechna relevantní nastavení, otevřete-li nový soubor machine.config.comments, který naleznete ve stejném adresáři. Obsahuje úplný text výchozích nastavení společně s komentáři (je to vlastně obdoba souboru machine.config v ASP.NET 1.x). V souboru machine.config.com-

ments se tedy dozvíte, jaká jsou výchozí nastavení a co znamenají, přičemž hodnoty, které potlačí tato výchozí nastavení, můžete poté přidat do souboru machine.config.

Spolu se souborem machine.config používá ASP.NET 2.0 kořenový soubor web.config (ve stejném adresáři), který obsahuje dodatečná nastavení. Tato nastavení registrují klíčové ovladače a moduly HTTP ASP.NET, připravují pravidla pro podporu prohlížečů, a definují bezpečnostní politiku. V ASP.NET 1.x byla všechna nastavení tohoto druhu umístěna v souboru machine.config.

Všechny webové aplikace na počítači dědí nastavení z obou těchto souborů. Většina z těchto nastavení jsou však v podstatě podpůrnými konstrukcemi, kterých se nemusíte nikdy dotknout. Dvě nejběžnější výjimky si probereme nyní.

<processModel>

Tato sekce umožňuje nakonfigurovat, jak bude zpracovatelský proces ASP.NET recyklovat domény aplikace, a účet Windows, pod kterým se bude proces vykonávat, což určuje jeho privilegia. Používáte-li IIS 6 (což je verze IIS umístěná ve Windows 2003 Server), budou se mnohá z těchto nastavení ignorovat, protože obdobná nastavení je možné nakonfigurovat prostřednictvím manažera IIS.

Další informace o prvku <processModel> najdete v kapitole 18.

<machineKey>

Tato sekce umožňuje nastavit specifický klíč serveru, který se bude používat pro zašifrování dat a vytváření digitálních podpisů. Šifrování se dá používat v součinnosti s několika schopnostmi ASP.NET. ASP.NET je používá automaticky, aby ochránil cookie formulářové autentizace, a vy je můžete rovněž používat, pokud chcete ochránit data stavu zobrazení (view state) (popisuje se v kapitole 6). Klíč se také používá při autentizaci s poskytovateli stavu relace, kteří pracují vně procesu.

Obvykle má prvek <machineKey> následující tvar:

```
<machineKey validationKey="AutoGenerate,IsolateApps"
decryptionKey="AutoGenerate,IsolateApps" validation="SHA1" />
```

Hodnota AutoGenerate,IsolateApps vyjadřuje, že ASP.NET bude vytvářet a ukládat klíče, které jsou specifické vzhledem ke stroji a aplikaci. Jinak řečeno – každá aplikace bude používat odlišný, automaticky vygenerovaný klíč. Tím se zabraňuje potenciálním útokům vedeným z jednoho webu na jiný web.

Nepotřebujete-li klíče specifické pro jednotlivé aplikace, můžete zvolit jediný pro všechny aplikace na aktuálním počítači takto:

```
<machineKey validationKey="AutoGenerate" decryptionKey="AutoGenerate"
validation="SHA1" />
```

Pokud provozuje webovou farmu a jedna aplikace běží na několika počítačích, v obou případech ovšem narazíte na problém. Požadujete např. nějakou stránku, kterou zpracuje jeden server. Stránka se pak odešle zpět na server, nicméně nyní ji zpracuje úplně jiný server, a ten nebude schopen dešifrovat stav zobrazení a cookie formulářové autentizace z prvního serveru. To kvůli tomu, že každý z webových serverů používá jiné klíče.

Abyste tuto záležitost vyřešili, je potřeba, abyste klíč explicitně nadefinovali v souboru machine.config. Podívejte se na ukázku prvku <machineKey>, ve které jsou oba důležité atributy nadefinovány:

```
<machineKey validationKey="61EA54E005915332011232149A2EEB317586824B265326CCDB3AD9A
BDBE9D6F24B0625547769E835539AD3882D3DA88896EA531CC7AFE664866BD5242FC2B05D" decrypt
ionKey="61EA54E005915332011232149A2EEB317586824B265337AF" validation="SHA1" />
```

TIP Takto explicitně lze také nadefinovat klíče specifické pro konkrétní aplikace tím, že přidáte explicitně vytvořený prvek <machineKey> do souboru web.config, který se nachází ve virtuálním adresáři aplikace. Tento přístup budete potřebovat, pokud se ocitnete v situaci, ve které jde o kombinaci obou scénářů popsaných výše. Například to budete potřebovat tehdy, provozujete-li svou aplikaci na několika serverech, a tyto servery současně hostují více webových aplikací, které potřebují individuální klíče.

Hodnota validationKey může být dlouhá v rozsahu od 40 do 128 znaků. Důrazně se doporučuje, abyste používali maximální dostupnou délku. Hodnota decryptionKey může být 16, nebo 48 znaků dlouhá. Bude-li to 16 znaků, bude pro zašifrování použit standard DES (Data Encryption Standard). Bude-li to 48 znaků, použije se Triple DES (neboli 3DES). (Což znamená, že se DES aplikuje třikrát za sebou.) 3DES se luští mnohem obtížněji než DES, takže se doporučuje, abyste pro decryptionKey vždy používali 48 znaků. Je-li délka některého z obou klíčů mimo povolené meze, a pokud budou kladeny na aplikaci nějaké požadavky, bude ASP.NET vracet stránku s chybovou zprávou.

Nemá valnou cenu vytvářet své vlastní ověřovací a dešifrovací klíče. Budete-li to dělat, nebudou pravděpodobně dostatečně náhodné, čímž budou zranitelnější proti některým druhům útoků, takže je lepší generovat silné náhodné klíče pomocí kódu a kryptografických tříd .NET Frameworku (z jmenného prostoru System.Security.Cryptography).

Následující ukázka je obecný kód rutiny s názvem CreateMachineKey(), která vytvoří náhodnou posloupnost bajtů pomocí kryptograficky silného generátoru náhodných čísel. Metoda CreateMachineKey() přebírá jediný parametr, který udává, kolik znaků se má použít. Návratová hodnota je v hexadecimálním formátu, který se požaduje v souboru machine.config.

```
public static string CreateMachineKey(int length)
{
  byte[] random = new byte[length/2];
  // Vytvoří kryptograficky silný generátor náhodných čísel.
  RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
  // Naplní pole bajtů náhodnými bajty.
  rng.GetBytes(random);
  // Vytvoří objekt StringBuilder, ve kterém bude udržovat výsledky,
  // jakmile se převedou do hexadecimálního formátu
  System.Text.StringBuilder machineKey = new
  System.Text.StringBuilder(length);
  // Projde v cyklu pole náhodných bajtů a přidává postupně
  // jednotlivé hodnoty do objektu StringBuilder.
  for (int i = 0; i < random.Length; i++)
  {
    machineKey.Append(String.Format("{0:X2}", random[i]));
  }
  return machineKey.ToString();
}
```

Tuto funkci můžete použít ve svém webovém formuláři pro vytvoření potřebných klíčů. Například následující fragment kódu vytvoří dešifrovací klíč dlouhý 48 znaků a ověřovací klíč dlouhý 128 znaků. Každou z obou hodnot pak zobrazí v oddělených textových polích:

```
txtDecryptionKey.Text = CreateMachineKey(48);
txtValidationKey.Text = CreateMachineKey(128);
```

Vytvořené informace můžete zkopírovat a vložit do souboru machine.config každého počítače ve webové farmě. Je to mnohem pohodlnější a bezpečnější postup, než vytvářet klíče ručně. O kryptografických třídách z jmenného prostoru System.Security.Cryptography se více dozvíte v kapitole 25.

Soubor web.config

Každá webová aplikace dědí nastavení ze souboru machine.config. Kromě toho můžete použít i nastavení specifické pro jednotlivé aplikace. Například tehdy, když chcete nastavit konkrétní metodu pro autentizaci, typ ladění, výchozí jazyk nebo vlastní chybové stránky. Je však důležité, abyste si uvědomili, že v souboru machine.config jsou některá nastavení, která nemůžete potlačit. Některá nastavení, jako jsou nastavení modelu procesu, se nedají měnit na úrovni aplikace. Některá nastavení také mohou být specifická pro konkrétní aplikaci. Používáte-li taková nastavení, musíte umístit soubor web.config do kořenového adresáře své aplikace (nikoliv do libovolného podadresáře).

Celý obsah konfiguračního souboru ASP.NET je vnořen do kořenového prvku <configuration>. Tento prvek obsahuje prvek <system.web>, který se používá pro nastavení ASP.NET. Uvnitř prvku <system.web> jsou pak oddělené prvky určené pro každý aspekt konfigurace.

Základní kostra souboru web.config vypadá takto:

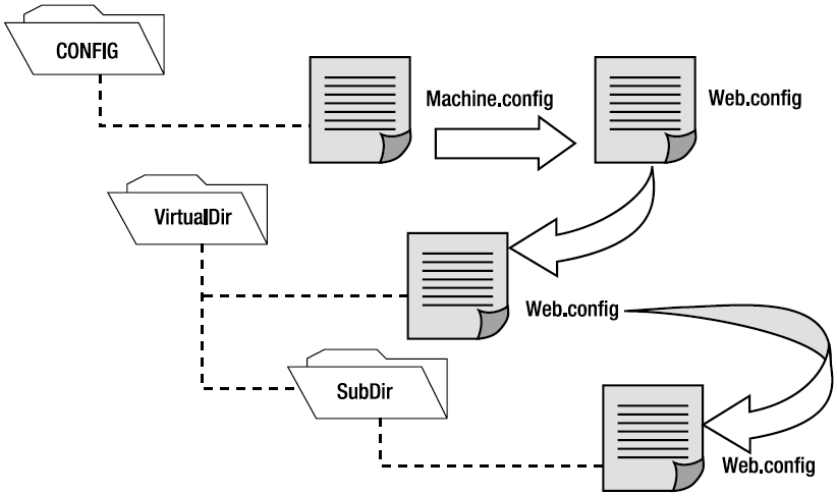
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <!-- sem přijdou konfigurační sekce ASP.NET. -->
  </system.web>
</configuration>
```

ASP.NET používá vícevrstvý konfigurační systém, který umožňuje používat pro různé části aplikace různá nastavení. Chcete-li tento systém používat, musíte přidat do svého virtuálního adresáře dodatečné podadresáře. Ty pak mohou obsahovat své vlastní konfigurační soubory web.config s dodatečnými nastaveními. ASP.NET používá dědění těchto konfigurací, takže každý podadresář obdrží nastavení z rodičovského adresáře.

Dejme tomu, že máme webový požadavek `http://localhost/A/B/C/MojeStranka.aspx`, kde A je kořenový adresář webové aplikace. V tomto případě se do hry zapojují nastavení z několika úrovní:

1. Nejprve se použijí výchozí nastavení z machine.config.
2. Pak se aplikují nastavení web.config z kořenového adresáře počítače. Tento soubor web.config je ve stejné složce CONFIG jako soubor machine.config.
3. Je-li nějaký soubor web.config v kořenu aplikace A, jeho nastavení se uplatní nyní.
4. Je-li nějaký soubor web.config v podadresáři B, jeho nastavení se uplatní nyní.
5. Je-li nějaký soubor web.config v podadresáři C, uplatní se jeho nastavení jako poslední.

K této posloupnosti je důležité připomenout, že ačkoliv můžete mít neomezený počet podadresářů, jsou významná zejména nastavení použitá v krocích 1 a 2. Je to proto, že některá nastavení se dají použít pouze na úrovni machine.config (jako je účet Windows, pod kterým se bude kód vykonávat), a jiná nastavení lze zase použít pouze na úrovni aplikace (jako je druh autentizace, kterou bude používat vaše webová aplikace).



Obrázek 5-3. *Dědění konfigurací.*

V podadresářích je tak možné specifikovat pouze malou podmnožinu nastavení, která se liší od zbytku webové aplikace. Jedním z důvodů, proč je někdy žádoucí použít v aplikaci několik podadresářů, jsou různá nastavení týkající se bezpečnosti. Soubory, které je potřeba zabezpečit, by se umístily do speciálního adresáře s takovým souborem web.config, který byl definoval přísnější bezpečnostní nastavení, než jaká jsou použita v kořenovém virtuálním adresáři.

Pokud jsou nějaká nastavení ve vzájemném rozporu, nastavení ve více vnořeném souboru vždy přepíše nastavení zděděné z rodiče. Až na jednu výjimku – je možné vyznačit konkrétní uzamčené sekce, které nebude možné měnit. Příslušnou techniku popisujeme v následujícím oddílu.

Práce s prvky <location>

Prvek <location> je rozšíření, které umožňuje specifikovat v jednom konfiguračním souboru více než jednu skupinu nastavení. Podadresář nebo soubor, odkud se budou nastavení aplikovat, specifikujete atributem path prvku <location>.

Například následující soubor web.config vytváří pomocí prvku <location> dvě skupiny nastavení – jedno, které je určeno pro aktuální adresář, a druhé, které se bude aplikovat pouze na soubory umístěné z podadresáři Secure:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <!-- Sem přijdou základní konfigurační nastavení. -->
  </system.web>
  <location path="/Secure">
    <system.web>
```

```
      <!-- Sem přijdou konfigurační nastavení pro podadresář Secure. -->
    </system.web>
  </location>
</configuration>
```

Tento soubor web.config v podstatě zastává roli dvou konfiguračních souborů. Funguje stejně, jako kdybyste nastavení rozdělili do dvou oddělených souborů web.config, a jeden z nich dali do podadresáře Secure.

Není stanoven žádný limit na počet prvků <location> v jednom konfiguračním souboru. Prvek <location> se však příliš často nepoužívá, protože konfigurační nastavení se obvykle snadněji spravují a aktualizují, když se udržují v oddělených souborech. Existuje však jedna situace, kdy prvek <location> poskytuje takovou funkcionalitu, které žádným jiným způsobem nedosáhnete. Je to tehdy, když uzamknete nějaké konkrétní nastavení, aby se nedalo potlačit.

Abyste pochopili, jak tato technika funguje, zamyslete se nad následujícím příkladem, ve kterém definuji dvě skupiny nastavení, přičemž jsem v jedné skupině nastavil atribut allowOverride značky <location> na hodnotu false. Je to pěkně vidět v následujícím kódu:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <!-- Sem přijdou nechráněná konfigurační nastavení. -->
    </system.web>
  <location allowOverride="false" >
    <system.web>
      <!-- Sem přijdou uzamčená konfigurační nastavení. -->
    </system.web>
  </location>
</configuration>
```

V tomto případě není možné potlačit žádné nastavení ze sekce <location>. Pokud se o to pokusíte, a budete ve své aplikaci požadovat nějakou stránku, ASP.NET vám vygeneruje nezpracovanou výjimku.

Atribut allowOverride prvku <location> je primárně určen pro potřeby webhostingových firem, které si chtějí být jisté, že určité nastavení nebude možné změnit. V těchto případech administrátor upraví soubor machine.config na webovém serveru tak, že pomocí prvku <location> uzamkne všechny sekce souboru web.config, které chce ochránit.

TIP Když uzamykáte nastavení v souboru machine.config, máte dvě možnosti. Zaprvé, můžete uzamknout nastavení pro všechny aplikace tím, že neuvedete atribut path značky <location>. Zadruhé, můžete uzamknout nastavení pro konkrétní webovou aplikaci, když nastavíte atribut path na její název.

Konfigurační nastavení

Prvek <system.web> obsahuje všechna konfigurační nastavení specifická pro ASP.NET. Specifikuje různé aspekty webové aplikace a zapíná služby, jako je bezpečnost, správa stavu a trasování.

Základní dceřiné prvky, které může prvek <system.web> obsahovat, a jejich účel, jsou uvedeny v tabulce 5-2. Seznam nicméně není úplný. Jeho účelem je vám dát pouze hrubou představu o rozsahu konfigurace ASP.NET.