2

Návrh webové aplikace

Prvním krokem při vývoji nového webu je návrh jeho vizuální podoby, která sestává z celkového rozvržení jednotlivých grafických prvků. Vizuální architektura webu významnou měrou ovlivňuje pocit, jaký bude mít uživatel při jeho prohlížení. Začneme tedy tak, že si stanovíme, jak by měl na uživatele působit, načež navrhneme mechanismy, s jejichž pomocí tohoto cíle dosáhneme. Zaměříme se především na menu a navigaci, na obrázky a uspořádání prvků na stránce. Menu musí být intuitivní a mělo by být doplněno navigačními pomůckami, jako je mapa stránek nebo *navigační cesta* (*navigace pomocí „sypaných drobků chleba" – breadcrumb navigation*), které uživateli připomínají, kde se právě nachází vzhledem k webu jako celku. Navigační cesta se skládá z posloupnosti krátkých odkazů, které tvoří stezku, po níž se může uživatel vydat, chce-li se po hierarchii stránek vracet vzhůru, zpět ke dříve navštíveným úrovním.

Před zahájením vlastního programování je vhodné vzít v potaz specifické prostředky nabízené technologií ASP.NET 2.0, a tak si ušetřit spoustu práce, kterou za nás již udělali lidé ze společnosti Microsoft. Položením vhodných základů pro technickou architekturu můžeme zvýšit znovupoužitelnost a udržovatelnost našeho kódu. V této kapitole se podíváme na celkové vizuální uspořádání našeho webu a vysvětlíme si, jak s výhodou používat užitečné techniky, jako jsou *vzorové stránky* a *motivy*. Pomocí vzorových stránek lze seskupovat funkční prvky do šablon, které umožňují sdílet společné prvky na více stránkách. Motivy zase dovolují uživatelům přizpůsobení jistých aspektů webu, jehož vzhled si tak přizpůsobí dle vlastních potřeb a chuti (této technice se také říká *skinning*).

Problém

Řada vývojářů začne psát zdrojový kód, aniž by věnovali pozornost hlavnímu cíli webové aplikace: poskytovat uživatelům jednoduchou, ale vysoce funkční grafickou aplikaci. Vývoj uživatelského rozhraní vypadá jako poměrně elementární úkol. Není-li však proveden pořádně, může se stát, že bude nutné se k němu v průběhu vývoje několikrát vracet. Ovšem pokaždé, když je nutné se vrátit a provést na nějakých významných prvcích změny, vyžaduje to nemalé množství práce navíc, a to nepočítáme nové kolo testování jednotek a integračních testů. A co hůř, je-li uživatelské rozhraní bráno příliš na lehkou váhu, může se stát, že nakonec odradí uživatele, kteří se rozhod-

nou pro jinou webovou stránku. Existuje celá řada prvků, které je nutné zvážit při tvorbě návrhu webové stránky. Nejdříve si musíme uvědomit jeden základní fakt: vzhled je opravdu důležitý! Pokud totiž stránka nevypadá dobře, ani lidé se na ní nebudou cítit dobře a pravděpodobně se na ni již nebudou vracet. Pozornost vývojářů snadno strhávají na první pohled obtížné úkoly, jako je organizování zdrojového kódu do tříd nebo programování aplikační logiky, takže význam vzhledu webové aplikace je často neprávem umenšován. Ale uživatelské rozhraní je přeci první věcí, kterou uživatel při otevření webu spatří. Je-li ošklivé, zmatené a vesměs nepoužitelné, pak si uživatel s největší pravděpodobností utvoří podobný obrázek i o společnosti, kterou taková webová stránka v podstatě reprezentuje. A to naneštěstí bez ohledu na ostatní významné vlastnosti webové aplikace, jakými jsou rychlost a škálovatelnost. Dále je třeba vzít v potaz, že ne všichni uživatelé sdílí stejný názor, co se týče zvolené šablony. Pro některé může být text v rámci použitého barevného schématu obtížně čitelný, takže mohou dávat přednost jiným barvám, které zase nemusejí vyhovovat ostatním. Je tedy velice těžké zavděčit se jediným barevným schématem všem uživatelům. Proto některé webové stránky nabízejí několik barevných schémat a někdy také několik různých uspořádání textu, čímž uživatelům umožňují, aby si vzhled stránky přizpůsobili dle svých představ a často i fyziologických překážek, mezi něž se řadí například barvoslepost. Studie ukázaly, že částečnou barvoslepostí trpí překvapivě velký počet lidí, kteří tak nejsou schopni od sebe odlišit určité barvy. Musejí mít tedy možnost zvolit si takové barevné schéma, které jim umožní rozlišit použité barvy a s nímž bude webová stránka vypadat i tak docela příjemně.

Poté, co stanovíme rozvržení a barvy, potřebujeme zajistit, aby stránka vypadala v různých prohlížečích stejně. Před pár lety dominoval mezi uživateli operačního systému Windows Internet Explorer (IE), takže při vývoji odborné stránky zaměřené na vývojáře pro Windows bylo možné předpokládat, že většina této uživatelské základny používá právě IE. Stačilo tedy vyvíjet a testovat pouze pro prohlížeč IE. Dnes však stále více získává na popularitě prohlížeč Firefox, který je k dispozici pro více operačních systémů (např. Linux a Mac OS). Cílová skupina uživatelů našeho webu je poměrně pestřejší (nejedná se jen o vývojáře pro Windows, ale o potenciálně všechny návštěvníky hospůdky našeho klienta) a kvůli tomu, že na platformách odlišných od Windows existují i jiné populární prohlížeče, je bezpodmínečně nutné zajistit, aby i na nich naše webová stránka bez obtíží fungovala. Pokud bychom tak neučinili a zaměřili se pouze na IE, mohlo by se stát, že by uživatelé Firefoxu měli prvky na stránce oproti očekávání uspořádané úplně jinak, špatně zarovnané, v nesprávných velikostech a barvách, s překrývajícími se panely a textovými rámečky – jinými slovy, obdrželi by naprostý guláš. Není těžké uhádnout, že uživatel, kterému bychom naservírovali takto nepěknou stránku, by jistě odešel, což by ale znamenalo, že bychom přišli i o potenciálního klienta či zákazníka internetového obchůdku. Když už nic jiného, tak ztráta návštěvnosti snižuje počet zobrazení reklamních proužků. A poněvadž návštěvníky ztratit nechceme, budeme se věnovat oběma hlavním prohlížečům, tedy Internet Exploreru i Firefoxu.

Plánování vrstvy uživatelského rozhraní nesestává jen z psaní HTML-kódu nějaké stránky. Zahrnuje také navigační systém a možnost pro webmastera či správce stránky (nebo i pro koncového uživatele) snadno změnit vzhled stránky, aniž by do ní musel jakkoliv zasahovat. Je tedy velmi užitečné vytvořit systém, s jehož pomocí lze jednoduše měnit jednotlivá menu a upravovat vzhled (fonty, barvy a velikosti nejrůznějších částí stránky), protože tím minimalizujeme práci správců, kteří nám za to jistě poděkují. Jakmile dokončíme úvodní stránku, zabere nám vývoj ostatních stránek mnohem méně času, neboť úvodní stránka již obsahuje kompletní uspořádání a navigační prvky, které se uplatní v rámci celé webové aplikaci. Pokud jsme vytvořili společné uživatelské rozhraní sdílené mnoha dalšími stránkami, tak jakákoliv pozdější úprava uspořádání prvků na

stránce (třeba přidání rámečku s hlasováním na pravou stranu všech stránek) bude velice jednoduchá. To je také důvod, proč skutečně stojí za to strávit nějaký ten čas navíc přemýšlením o kvalitním návrhu základní vrstvy uživatelského rozhraní namísto okamžitého spuštění aplikace Visual Studio .NET následované bezmyšlenkovitým programováním. Jde vskutku o strategické rozhodnutí, které nám může ušetřit hodiny nebo i dny práce navíc. Pamatujme si tedy, že začlenění zásadních změn aplikovaných v pozdější, vývojové fázi, vyžaduje mnohem více času a úsilí.

Návrh

V této části se podíváme na problémy popsané výše a promyslíme způsob jejich řešení, přičemž vytvoříme technický návrh systému. V praxi to znamená navrhnout a implementovat následující prvky:

- dobře vypadající grafická šablona (rozvržení grafických prvků), která se bude zobrazovat stejně v Internet Exploreru i Firefoxu, včetně mechanismu pro dynamickou aplikaci různých barevných schémat a ostatních vizuálních atributů na tuto šablonu;
- jednoduchý způsob sdílení vytvořené šablony všemi stránkami webové aplikace, aniž bychom do každé z nich museli kopírovat a vkládat celý kód;
- navigační systém, jenž umožní snadnou úpravu odkazů v menu, z něhož bude uživatelům zřejmé, kde se v rámci mapy webu právě nacházejí a kudy se mohou vydat zpět;
- mechanismus, který umožní nejen společný vizuální návrh všech stránek webové aplikace, ale také jejich společné chování, jako je například počítání zobrazení stránky či aplikace oblíbeného stylu uživatele;

Při implementaci požadavků týkajících se uživatelského přizpůsobení parametrů, znovupoužitelnosti, menu a navigace si popíšeme, jak s výhodou využít některé z nových prostředků technologie ASP.NET 2.0. Později, v části „Řešení“, si tyto užitečné prvky oživíme!

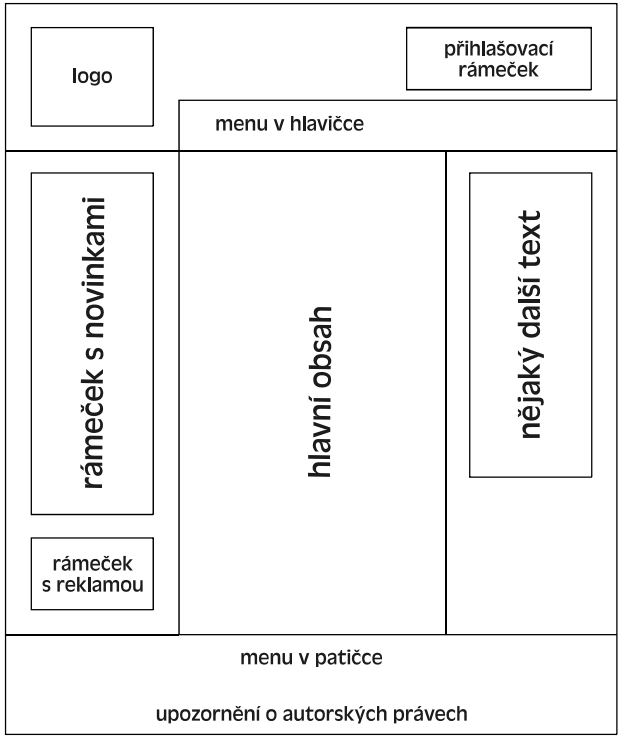
Návrh uspořádání grafických prvků

Při vývoji návrhu webové stránky se obvykle pomocí grafické aplikace (např. Adobe Photoshop či Jasc Paint Shop Pro) nejdříve vytvoří maketa, z níž by měla být patrná finální podoba stránky, a to ještě před tím, než začneme s jakýmkoliv programováním. Jakmile máme maketu připravenou, můžeme ji ukázat nejrůznějším uživatelům, testerům a manažerům, kteří poté mohou rozhodnout o zahájení vlastního programování. Stačí i obyčejný obrázek podobný tomu na obrázku 2.1, který zachycuje rozdělení textu do jednotlivých oblastí stránky.

Jedná se o typické třísloupkové uspořádání s hlavičkou a patičkou. Po schválení jej musíme sestavit znovu, tentokráte však s opravdovou grafikou a HTML-kódem. Pro maketu se používá grafický program, protože s tímto typem programu zabere návrháři webových stránek její vytvoření mnohem méně času. Poté, co klient finální podobu makety odsouhlasí, může ji návrhář webových stránek rozřezat na malé kousky a použít je pro tvorbu HTML-stránky.

Tvorba makety není vždy jednoduchá, obzvláště pro ty, kteří nejsou od přírody umělecky založeni. Pro střední či větší firmu to většinou nebývá problém, protože často je tu někdo jiný, konkrétně profesionální návrhář webových stránek, který vytvoří grafický návrh, z něhož pak vývojáři při práci na webové aplikaci vycházejí. Někdy může být vhodné najmout externí firmu, která se o grafický návrh postará sama, nebo lze tvorbu šablony svěřit ve formě sub-kontraktu

někomu talentovanějšímu. Pro účely našeho projektu jsme sáhli po jedné z šablon serveru TemplateMonster (`www.templatemonster.com`), z níž jsme vytvořili poměrně slušný grafický návrh webové stránky, na němž budeme dále stavět. Server TemplateMonster nabízí šablony ve formě souborů typu PSD (ty lze otevřít v aplikaci Photoshop či Paint Shop Pro), JPEG a HTML, přičemž obrázky jsou již rozřezané na části a uspořádané pomocí HTML-tabulek. HTML-stránky však nepoužijeme bez předchozích úprav, chceme totiž vytvořit alespoň do jisté míry náš vlastní styl. Nicméně je velmi užitečné, máme-li nějakou vizuální předlohu, od které se můžeme odrazit. Náš web tak navíc od počátku získá profesionální nádech, což může výrazně pomoci při dalším prodeji návrhu.



Obrázek 2.1: Maketa naší nové webové stránky

Technologie použité pro implementaci návrhu

Technologie ASP.NET 2.0 zaujímá vzhledem k fungování naší webové aplikace prvotní místo. Běží na webovém serveru a s výhodou využívá funkční prvky poskytované rámcem .NET Framework. Nicméně ASP.NET neběží na počítači uživatele, ale dynamicky generuje prvky, které používá prohlížeč pro vykreslování webových stránek. Patří mezi ně HTML-značky, obrázky a definice kaskádových stylů (CSS – *Cascading Style Sheets*), které definují barvy, velikosti a polohy nejrůznějších objektů na HTML-stránce. ASP.NET dále generuje procedurální kód jazyka JavaScript, který posílá prohlížeči pro kontrolu uživatelem zadaných údajů a také pro ustavení způsobu interakce s webovým serverem.

HTML-kód lze definovat několika různými způsoby. Můžeme sáhnout po vizuálním návrháři formulářů v prostředí Visual Studio, který jej automaticky vygeneruje na základě vkládaných řídicích prvků z palety na formulář. Nebo můžeme ručně upravit či vytvořit vlastní HTML-kód v souboru typu `.aspx`, čímž získáme přístup také k těm prvkům, se kterými bychom pomocí návrháře formuláře pracovat nemohli. Poslední možností je nechat HTML-kód vygenerovat dynamicky z našeho kódu v jazyku C# nebo ze tříd rámce .NET Framework.

Prostředí ASP.NET 1.x používá model s tzv. *kódem v pozadí* (*code-behind*): HTML-kód (a některý prezentačně orientovaný C#-kód) byl uložen v souboru typu `.aspx` a implementační C#-kód se nacházel ve zvláštním souboru, od kterého byl soubor typu `.aspx` odvozen. Soubor typu `.aspx` se nazývá „stránka“, poněvadž definuje vizuální podobu webové stránky. Tento model poskytuje určitou míru separace mezi prezentačním a s ním spojeným implementačním kódem. Ovšem jeden s problémů s tímto modelem spočívá v tom, že kód automaticky generovaný návrhářem formulářů je umístěn ve stejných souborech, které používá vývojář pro svůj vlastní kód.

ASP.NET 2.0 tento model upravuje a přidává k němu prvky z nové verze rámce .NET Framework 2.0, mezi které patří i *částečné třídy*. Princip je jednoduchý: umožnit definici jediné třídy překlenout několik souborů. Prostředí aplikace Visual Studio tak může za běhu generovat kód pro deklaraci řídicích prvků a registraci událostí a kombinovat jej s kódem, který píše vývojář. Výsledkem je pak jediná třída, od které je odvozena stránka v souboru typu `.aspx`. V něm je deklarována direktiva `@Page`, která pomocí atributu `CodeFile` odkazuje na soubor typu `.cs` obsahující kód v pozadí psaný uživatelem.

Další změna v ASP.NET 2.0 spočívá v eliminaci souborů projektu. Prvky projektů se nyní rozpoznávají na základě definované struktury adresářů. Kód všech stránek projektu byl navíc v ASP.NET 1.x generován do jediného souboru typu `.dll`. ASP.NET 2.0 však generuje kód pro každou stránku zvlášť. Z jakého důvodu? Při změně jediné stránky tak není třeba opětovně nasazovat větší množství kódu. Stačí totiž opětovně nasadit jen kód příslušný dané stránce, takže vývojáři mají jemnější kontrolu nad správou změn zdrojového kódu.

Použití jazyka CSS pro definici stylů

Tato kniha nemá ambice poskytnout vyčerpávající výklad o jazyku CSS, nicméně některé obecné pojmy si alespoň stručně vysvětlíme. Pro podrobnější informace o jazyku CSS je třeba se obrátit na některé další zdroje. Cílem jazyka CSS je specifikovat způsob vykreslení HTML-značek pomocí nejrůznějších stylistických prvků, jako je velikost fontu, barva, zarovnání a tak podobně. Tyto styly lze ve formě atributů vložit do HTML-tagů přímo nebo je lze uložit zvlášť a odvolat se na ně přes jejich jméno či ID.

Někdy mají HTML-soubory styly napevno vloženy do HTML-značek, jak je tomu v následujícím příkladu:

```
<div style="align: justify; color: red; background-color: yellow; font-size:
12px;">nějaký text</div>
```

To však není dobré, neboť je obtížné tyto stylistické prvky upravovat, aniž by bylo nutné procházet všechny HTML-soubory a hledat příslušné CSS-atributy. Místo toho by definice stylů měly být vždy umístěny ve zvláštním souboru s příponou `.css` nebo alespoň v horní části HTML-souboru v rámci značky `<style>`.

K seskupování CSS-stylů dohromady lze vytvářet malé třídy, které svou syntaxí připomínají třídy či funkce v C#. Lze jim přiřadit jméno, neboli ID, takže se na ně můžeme v HTML-značkách odkazovat pomocí atributu class.

Při použití tříd definujících styl můžeme změnit velikost fontu všech HTML-značek, které se ni odkazují, pouhou úpravou hodnoty v rámci její deklarace, čímž dojde ke změně celé řady vizuálních HTML-prvků daného typu. Pokud styl definujeme v odděleném souboru, pak stačí upravit jediný soubor, načež svůj vzhled může změnit hned několik stránek.

Hlavní výhoda při použití jazyka CSS spočívá v minimalizaci úsilí při správě stylů a k prosazení jednotného vzhledu většího počtu stránek. Ovšem kromě toho, jazyk CSS svým způsobem také zajišťuje bezpečnost HTML-kódu a potažmo i celého webu. Předpokládejme, že klient chce změnit některé styly na stránkách, které jsou již v ostrém provozu. Pokud bychom styly umístili do HTML-prvků stránky napevno, museli bychom při provádění jakýchkoliv změn procházet množství souborů, což může vést k chybám zapříčiněným překlepy či přehlédnutím hledaného stylu. Máme-li však třídy stylů uložené ve zvláštních souborech typu CSS, pak je mnohem snazší vyhledat třídy, které je třeba změnit, přičemž náš HTML-kód zůstane v bezpečí a nedotčen.

Kromě toho mohou soubory typu CSS celé webové stránky zefektivnit. Prohlížeč je totiž stáhne pouze jednou a umístí si je do vyrovnávací paměti. Jednotlivé HTML-stránky pak neobsahují opakující se definice stylů, ale mohou se odkazovat na tyto sobory ve vyrovnávací paměti, čímž dojde ke snížení jejich velikosti, což zase vede ke zrychlení jejich stahování. V některých případech může tento způsob práce se styly způsobit dramatické zvýšení rychlosti načítání webové stránky v prohlížeči uživatele.

Následuje ukázka redefinice stylu výše uvedeného objektu typu DIV. Styl jsme uložili do samostatného souboru jménem `styles.css`:

```
.mystyle
{
   align: justify;
   color: red;
   background-color: yellow;
   font-size: 12px;
}
```

> Všimněte si, že jsme při deklaraci stylu uvedli před jménem třídy tečku. U všech vlastních tříd stylů je nutné tento prefix uvádět.

S HTML-kódem v souboru typu `.aspx` nebo `.htm` jej propojíme takto:

```
<head>
   <link href="/styles.css" text="text/css" rel="stylesheet" />
   <!-- další meta-značky ... -->
</head>
```

Nakonec zapíšeme HTML-značku `<div>` a určíme, kterou CSS-třídu má použít:

```
<div class="mystyle"> nějaký text </div>
```

Chceme-li námi definovaný styl aplikovat na všechny HTML-objekty určitého typu (například na všechny značky <p> nebo <body>), s nimiž není explicitně spojena nějaká jiná třída, pak můžeme do souboru se styly zapsat následující specifikace:

```
body
{
   margin: 0px;
   font-family: Verdana;
   font-size: 12px;
}

p
{
   align: justify;
   text-size: 10px;
}
```

Tímto způsobem nastavíme z jednoho místa výchozí styl pro všechny značky <body> (tělo stránky) a <p> (odstavec). Nicméně stále máme možnost pro některé odstavce definovat odlišný styl explicitním uvedením jména třídy stylu v příslušné značce <p>.

Další způsob propojení třídy definující styl s HTML-objektem spočívá ve využití jeho ID. Jméno třídy definujeme v tomto případě s prefixem #:

```
#header
{
   padding: 0px;
   margin: 0px;
   width: 100%;
   height: 184px;
   background-image: url(images/HeaderSlice.gif);
}
```

Nyní můžeme propojit styl s HTML-objektem pomocí jeho atributu id. Tímto způsobem lze například definovat HTML-značku , která používá styl s názvem #header:

```
<div id="header"> nějaký text </div>
```

Výše uvedený postup se obvykle používá pro neopakující se objekty, jako je kupříkladu hlavička, patička, kontejner pro levý, pravý či prostřední sloupek a podobně.

Oba přístupy lze bez problémů kombinovat. Předpokládejme, že chceme dát jistý styl všem odkazům v kontejneru se stylem sectiontitle a jiný styl odkazům v kontejneru se stylem sectionbody. Můžeme to provést například takto:

V souboru typu .css

```
.sectiontitle a
{
   color: yellow;
```

```
}

.sectionbody a
{
   color: red;
}
```

V souboru typu .aspx / .htm

```
<div class="sectiontitle">
   nějaký text
   <a href="http://www.cpress.cz">Computer Press</a>
   nějaký text
</div>

<div class="sectionbody">
   nějaký jiný text
   <a href="http://knihy.cpress.cz">Knihy</a>
   nějaký jiný text
</div>
```

HTML-tabulky se pro uspořádání prvků na stránce nehodí

Vývojáři občas používají HTML-tabulky pro uspořádání objektů na webové stránce. Před vytvořením jazyka CSS šlo o standardní postup, nicméně řada vývojářů používá tuto techniku dodnes. Ačkoliv jde o velmi rozšířenou praxi, organizace W3C od ní oficiálně odrazuje tvrzením (viz `http://www.w3.org/TR/WAI-WEBCONTENT/`): „Pomocí tabulek by se měly označovat pouze informace skutečně tabulkového charakteru (tzv. datové tabulky). Neměly by se používat pro uspořádání prvků na stránkách s textovým obsahem (tzv. rozvrhovací tabulky). Tabulky používané k v podstatě jakýmkoliv účelům představují zvláštní problémy pro uživatele se čtečkami obrazovek.“ Jinými slovy, HTML-tabulky by se měly používat pro zobrazování tabulkových dat, a ne pro celkové rozvržení stránky. K tomuto účelu bychom měli vždy sáhnout po řídicích prvcích kontejneru (jako je např. značka `<div>`) s příslušným nastavením jejich stylu, nejlépe prostřednictvím zvláštní sekce `<style>` nebo samostatného souboru. Jedná se o ideální řešení, a to hned z několika důvodů:

- Pokud definujeme vzhled a rozmístění pomocí značek `<div>` a samostatného souboru se styly, nemusíme na každé stránce opakovat tyto definice stále znovu, což vede k webové stránce, která je zároveň rychleji vytvořena a snadněji se udržuje.
- Stránka se koncovému uživateli načte mnohem rychleji! Pamatujme, že klient si soubor se styly stáhne pouze jednou a poté jej bude mít pro následující požadavky k dispozici ve vyrovnávací paměti, dokud se jejich obsah na straně serveru nezmění. Pokud bychom definovali rozvržení uvnitř HTML-souboru pomocí tabulek, musel by klient stahovat tabulkové rozvržení pro každou stránku zvlášť, čímž by přenášel větší množství bajtů, což by zabralo více času. Rozvržení řízené styly typicky sníží počet stahovaných bajtů až na polovinu, takže výhoda tohoto přístupu je ihned viditelná. Tyto úspory jsou navíc mnohem více patrné na těžce zatěžovaných webových serverech, poněvadž snížení počtu zasílaných bajtů každému z uživatelů se násobí počtem souběžně připojených klientů.

- Čtečky obrazovek, což je software, který dokáže číst text a ostatní informace na stránce slepým a zrakově handicapovaným uživatelům, mají při použití tabulek pro rozvržení prvků na stránce mnohem těžší práci. Z toho vyplývá, že při rozvrhování bez použití tabulek můžeme zvýšit přístupnost webové stránky, což je naprosto nezbytná nutnost pro určité skupiny serverů, jako je veřejná správa a vládní organizace. Některé společnosti jsou místo přijetí takto jednoduchých pravidel ochotné odepsat celé skupiny uživatelů.
- CSS-styly a značky `<div>` jsou na rozdíl od tabulek mnohem všestrannější. Můžeme mít například různé soubory se styly definující odlišné vzhledy a umístění pro nejrůznější objekty na stránce. Přepínáním připojeného souboru se styly tak můžeme kompletně měnit vzhled stránky, aniž bychom cokoliv upravovali na samotných stránkách s obsahem. S dynamickými stránkami využívajícími technologii ASP.NET lze navíc měnit soubor se styly za běhu a snadno tak implementovat mechanismus, díky němuž si mohou koncoví uživatelé vybrat styl, který jim nejvíce vyhovuje. Přitom nejde jen o barvy a fonty – v souborech typu CSS totiž můžeme stanovit polohy jednotlivých objektů a vytvořit si tak soubor, který umístí rámeček s menu do levého horního rohu stránky, a k tomu jiný, který jej zasadí do spodního pravého rohu. To je velice důležité, neboť chceme, aby si uživatelé zvolili svůj oblíbený styl ze seznamu dostupných motivů.
- Jazyk CSS umožňuje vyvíjet pro různé typy zařízení, v některých případech dokonce bez nutnosti nového HTML-kódu. Patří mezi ně například mobilní zařízení, jako jsou PDA či chytré telefony. Kvůli menší obrazovce je nutné přizpůsobit rozvržení obsahu na stránce, aby se vešel na tak malou obrazovku a byl stále dobře čitelný. Lze to provést pomocí zvláštního souboru se styly, který změní velikost a polohu některých kontejnerů (např. svislé sloupky umístí pod sebe) nebo je úplně skryje. Můžeme například skrýt kontejner pro reklamní proužky, hlasování a hlavičku s velkým logem. Pokud bychom se o něco takového pokoušeli s tabulkami, bylo by to mnohem obtížnější. Bylo by nutné pořádně promyslet mechanismus uživatelského přizpůsobení skinů, přičemž by bylo zapotřebí napsat samostatné stránky pro každé rozvržení. Oproti prostému vytvoření nového souboru typu CSS se jedná o neúměrně větší množství práce.

  Všimněte si, že výše uvedená diskuse se týká použití tabulek pro celkové rozvržení prvků na webové stránce. Tvorba vstupních formulářů s tabulkovou strukturou pomocí tabulek je však přípustná. V opačném případě by totiž k jejich snadnému zápisu a údržbě bylo zapotřebí příliš mnoho CSS-kódu. Navíc není pravděpodobné, že bude nutné dynamicky měnit rozvržení vstupního formuláře, čímž se celá flexibilita jazyka technologie CSS stává zbytečnou a použití HTML-tabulek tak zůstává mnohem přímočařejší.

Sdílení společného vzhledu mezi více stránkami

Jakmile dokončíme grafický návrh naší webové stránky, musíme najít způsob, jak jej jednoduše aplikovat na *n* dalších stránek, kde *n* může znamenat desítky, či dokonce stovky stránek. V předchozí edici této knihy týkající se ASP.NET 1.x jsme se drželi klasického postupu izolace společných částí návrhu do souborů s uživatelskými řídicími prvky, které se poté importovaly do všech stránek, které je potřebovaly. Speciálně jsme museli vytvořit jeden řídicí prvek pro hlavičku a druhý pro patičku. Ačkoliv šlo o výrazně lepší postup, než je kopírování kódu do všech stránek, a o něco lepší techniku, než je vkládání souborů s klasickým ASP-kódem (kvůli jejich objektově orientované povaze), stále to nebylo ono. Problém tkvěl především v tom, že pro každou stránku bylo stále nutné k importování řídicích prvků napsat několik řádků do souboru typu `.aspx` a k tomu přidat pár dalších řádků pro jejich umístění na vybraná místa na stránce. Pokud bychom je tedy umístili

do určité pozice na první stránce a do jiné na druhé stránce, vypadaly by tyto dvě stránky při spuštění odlišně. Není však možné, abychom se při každé tvorbě nové stránky neustále věnovali těmto detailům. Místo toho bychom se spíše měli zaměřit na vlastní obsah přidávané stránky a společné rozvržení prvků si nechat aplikovat na všechny stránky zcela automaticky. Ve skutečnosti potřebujeme nějaký druh vizuální dědičnosti, s jejíž pomocí bychom mohli definovat „základní" stránku a ostatní stránky od ní odvodit. V ASP.NET 1.x je však možné aplikovat dědičnost pouze na úrovni programového kódu v pozadí, který má vliv pouze na chování stránky (např. co se stane, když se stránka načte či vykreslí), a ne na její vzhled. I tento problém se dal několika způsoby obcházet, žádný z nich však nesplňuje naše požadavky co se funkčnosti a podpory ve fázi návrhu týče. Řešení tak nakonec přinesla sama technologie ASP.NET 2.0.

Model vzorové stránky

Technologie ASP.NET 2.0 zavádí nový prvek, tzv. „vzorové stránky", které umožňují definovat společné oblasti (např. hlavičky, patičky, menu a podobně) sdílené všemi stránkami. Vzorová stránka dovoluje umístit kód s uspořádáním stránky do jediného souboru, od kterého budou vizuálně dědit všechny ostatní stránky. Vzorová stránka obsahuje celkové rozvržení grafických prvků webové aplikace. Stránky s obsahem mohou vzhled vzorové stránky zdědit a vlastní obsah umístit do řídících prvků typu `ContentPlaceHolder`. I když výsledkem je určitá forma vizuální dědičnosti, samotný mechanismus není ve skutečnosti realizován jako dědičnost ve smyslu objektově orientovaného programování. Jeho implementace je totiž založena na modelu šablon.

Příklad řekne víc než tisíc slov, pojďme se tedy podívat, jak tento princip vypadá v praxi. Vzorová stránka má příponu `.master` a „pod povrchem" je podobná uživatelskému řídicímu prvku. Následuje ukázka kódu vzorové stránky obsahující nějaký text, hlavičku, patičku a mezi nimi řídicí prvek `ContentPlaceHolder`:

```
<%@ Master Language="C#" AutoEventWireup="true"
   CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<html>
<head id="Head1" runat="server">
   <title>Pivnice</title>
</head>

<body>
<form id="Main" runat="server">
   <div id="header">Pivnice</div>
   <asp:ContentPlaceHolder ID="MainContent" runat="server" />
   <div id="footer">Copyright 2005 Marco Bellinaso</div>
</form>
</body>
</html>
```

Jak je patrné, vzorová stránka je velmi podobná stránce standardní, až na to, že má v horní části místo direktivy `@Page` direktivu `@Master` a deklaruje jeden nebo i více řídicích prvků typu `ContentPlaceHolder`, do nichž mohou stránky typu `.aspx` umisťovat svůj obsah. Vzorová a obsahová stránka za běhu splynou v jednu, a protože vzorová stránka definuje značky `<html>`, `<head>`,

<body> a <form>, není překvapivé, že stránky s obsahem již tyto značky definovat nesmějí. Měli by definovat pouze obsah pro řídicí prvky typu ContentPlaceHolder. Následující úryvek zachycuje ukázku obsahové stránky:

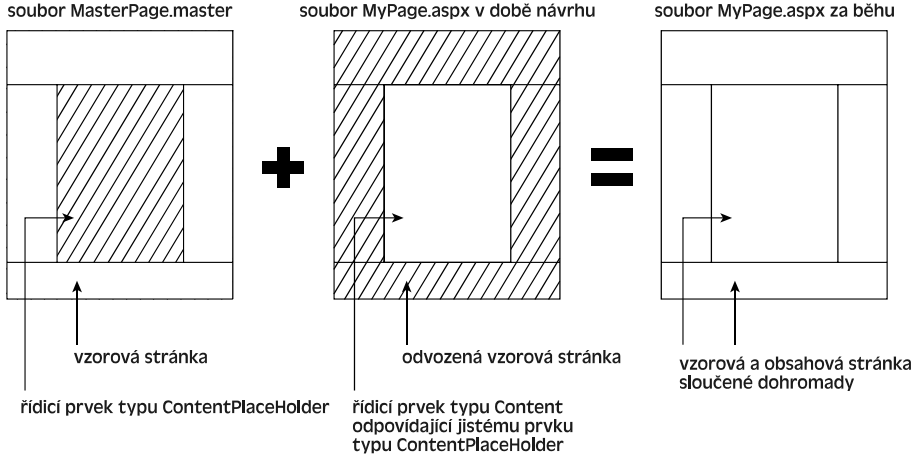
```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
   AutoEventWireup="true" CodeFile="MyPage.aspx.cs" Inherits="MyPage"
   Title="Pivnice - Moje stránka" %>

<asp:Content ID="MainContent" ContentPlaceHolderID="MainContent"
   Runat="Server">
   Obsah stránky bude tady ...
</asp:Content>
```

Prvním pravidlem je, že direktiva @Page nastavuje atribut MasterPageFile na virtuální cestu ke vzorové stránce, která se má použít. Vlastní obsah jsme umístili do řídicího prvku typu Content, jehož atribut ContentPlaceHolderID musí být nastaven na ID jednoho z řídicích prvků typu ContentPlaceHolder ve vzorové stránce. Do stránky s obsahem nelze umístit nic jiného než řídicí prvky typu Content, přičemž všechny ostatní řídicí prvky jazyka ASP definující vizuální prvky musejí být seskupeny v nejzevnějším prvku typu Content. Další novinkou, která stojí za pozornost, je nový atribut Title direktivy @Page, jehož prostřednictvím je možné přepsat hodnotu specifikovanou v meta-značce <title> vzorové stránky. Pokud atribut Title pro danou stránku s obsahem nedefinujeme, použije se nadpis zadaný ve vzorové stránce.

Při úpravě obsahové stránky v prostředí Visual Studio, vykreslí návrhář formulářů obě stránky (vzorovou i obsahovou) správně, přičemž obsah vzorové stránky bude „zašedlý“. Jedná se o připomenutí, že při úpravě obsahové stránky není možné modifikovat obsah vzorové stránky.

Nyní je třeba připomenout, že i vzorová stránka má svůj soubor s doprovodným kódem (*codebeside*), který lze využít pro zápis vlastností a funkcí jazyka C#, k nimž můžeme přistupovat jak ze souborů typu .aspx, tak i z doprovodného kódu obsahových stránek.



Obrázek 2.2: Grafická reprezentace vzorové stránky

Při definici prvků typu `ContentPlaceHolder` ve vzorové stránce můžeme také specifikovat jejich výchozí obsah, který se použije v případě, že konkrétní obsahová stránka neobsahuje odpovídající prvek typu `Content`. Následující úryvek demonstruje specifikaci výchozí obsahu:

```
<asp:ContentPlaceHolder ID="MainContent" runat="server">
   Zde může být výchozí obsah ...
</asp:ContentPlaceHolder>
```

Výchozí obsah je velice užitečný při řešení takových situací, kdy potřebujeme přidat novou část do většího počtu obsahových stránek, aniž bychom chtěli či mohli upravit každou z nich. Ve vzorové stránce můžeme připravit nový prvek typu `ContentPlaceHolder`, přiřadit mu nějaký výchozí obsah a pak beze spěchu přidávat nové informace na stránky s obsahem, přičemž na těch, ke kterým jsme se dosud nedostali, se bez problémů zobrazí výchozí obsah ze vzorové stránky.

Atribut `MasterPageFile` na úrovni stránky je užitečný v případě, kdy pro odlišné sady obsahových stránek potřebujeme použít různé vzorové stránky. Pokud by však pro všechny stránky naší webové aplikace používaly tutéž vzorovou stránku, pak by bylo jednodušší, specifikovat ji prostřednictvím elementu `<pages>` v souboru `web.config`:

```
<pages masterPageFile="~/Template.master" />
```

Pokud i v této situaci uvedeme na úrovni stránky atribut `MasterPageFile`, přepíšeme pro příslušnou stránku hodnotu v souboru `web.config`.

Vnořené vzorové stránky

Půjdeme-li o krůček dále, pak obsahem vzorové stránky může být opět jiná vzorová stránka. Jinými slovy, vzorové stránky můžeme vnořovat, takže jedna vnořená vzorová stránka zdědí vzhled nadřazené vzorové stránky, přičemž obsahové stránky typu `.aspx` dědí od vnořené vzorové stránky. Vzorová stránka druhé úrovně může vypadat například takto:

```
<%@ Master Language="C#" MasterPageFile="~/MasterPage.master"
   AutoEventWireup="true" CodeFile="MasterPage2.master.cs"
   Inherits="MasterPage2" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
   Nějaký další obsah ...
   <hr style="width: 100%;" />
   <asp:ContentPlaceHolder ID="MainContent" runat="server" />
</asp:Content>
```

K tomu, aby obsahová stránka začala používat vzorovou stránku druhé úrovně, stačí upravit pouze hodnotu jejího atributu `MasterPageFile`, což je dáno tím, že prvky typu `ContentPlaceHolder` v rodičovské i odvozené vzorové stránce mohou mít totožné ID.

Díky tomu můžeme mít vnější vzorovou stránku, která definuje velmi hrubé rozvržení (často jde o rozvržení typu *companywide*), a pak ostatní vzorové stránky, jež stanoví uspořádání pro specifickou oblast webu, jako je například Internetový obchod či administrační část. Jediný problém s vnořenými vzorovými stránkami spočívá v tom, že pro ně (narozdíl od vzorových stránek první úrovně) neexistuje v prostředí Visual Studia podpora ze strany vizuálního návrháře. Při úpravě ob-

sahových stránek je nutné vše programovat přímo ve zdrojových souborech, přičemž výsledek lze zhodnotit až při otevření stránky v prohlížeči. Pro vývojáře, kteří si raději většinu kódu píší sami v okně SOURCE VIEW, to zase tak velký problém není, a také ti ostatní by měli možnost použití vnořených vzorových stránek zvážit, protože se skutečně jedná skvělou techniku!

Přístup z obsahové do vzorové stránky

Z obsahové stránky můžeme přes vlastnost `Master` třídy `Page` přistupovat k její vzorové stránce. Obdržíme objekt typu `MasterPage`, který je přímo odvozen od třídy `UserControl` (říkali jsme si, že vzorové stránky jsou podobné uživatelským řídicím prvkům), k níž přidává několik vlastností. Nabízí kolekci `Controls`, jejímž prostřednictvím lze z obsahové stránky přistupovat k řídicím prvkům stránky vzorové. To se může hodit, zvláště pak chceme-li na určité stránce skrýt některé řídicí prvky vzorové stránky (např. rámeček s přihlašovacími údaji či reklamou). S kolekcí `Controls` bychom sice mohli pracovat přímo, získané objekty bychom však museli manuálně přetypovat z obecného typu `Control` na správný typ, což by znamenalo používat slabě typovaný přístup. Mnohem lepší postup, který je navíc v souladu s objektovým myšlením, spočívá v přidání nových vlastností do třídy v doprovodném kódu vzorové stránky. V našem případě to znamená zabalit vlastnost `Visible` nějakého řídicího prvku, což můžeme provést například takto:

```
public bool LoginBoxIsVisible
{
   get { return LoginBox.Visible; }
   set { LoginBox.Visible = value; }
}
```

Nyní můžeme k obsahové stránce přidat za direktivu `@Page` následující řadek:

```
<%@ MasterType VirtualPath="~/MasterPage.master" %>
```

Tím jsme stanovili cestu ke vzorové stránce, kterou použije prostředí ASP.NET k dynamickému vytvoření silně typované třídy `MasterPage`, jež odhaluje vlastnosti nově přidané ke třídě v jejím doprovodném kódu. Vypadá to sice jako kopírování atributu `MasterPageFile` direktivy `@Page`, jde však o způsob, jak obsahové stránce zpřístupnit vlastnosti stránky vzorové. Vzorový typ lze kromě virtuální cesty (jako ve výše uvedeném příkladě) specifikovat také pomocí jména příslušné třídy, k čemuž slouží atribut `TypeName`. Jakmile tuto direktivu přidáme do souboru s doprovodným kódem obsahové stránky (nebo do sekce `<script runat="server">` samotného souboru typu `.aspx`), můžeme se snadno dostat k vlastnosti vzorové stránky `LoginBoxIsVisible`, přičemž používáme silně typovaný přístup:

```
protected void Test_OnClick(object sender, EventArgs e)
{
   this.Master.LoginBoxIsVisible = false;
}
```

Z použití silně typovaného přístupu mimo jiné plyne, že v prostředí Visual Studia máme k dispozici pomocný nástroj `Intellisense`, který nám po napsání „`this.Master.`“ bez problému nabídne také naši nově přidanou vlastnost!

Metodologie přístupu ke vzorovým objektům z obsahových stránek je užitečná zvláště v případě, kdy potřebujeme do vzorové stránky umístit metody, běžně používané na odvozených stránkách. Pokud bychom neměli přístup k silně typovanému objektu `MasterPage` vytvořenému za běhu v prostředí ASP.NET, museli bychom pro volání těchto metod využít reflexi, která je pomalejší a rozhodně ne tak jednoduchá (v našem případě by bylo snazší umístit sdílené metody do samostatné třídy, ke které mají přístup všechny stránky).

Co se první edice této knihy týče, je třeba upozornit na rozdíly při používání bázové stránky ve smyslu OOP a vzorové stránky (MasterPage). V první edici jsme si definovali bázovou třídu `ThePhile`, od níž jsme odvodili všechny „obsahové" stránky. Šlo o skutečnou dědičnost dle OOP, ale s omezenými možnostmi, neboť jsme od bázové třídy nemohli odvodit žádný z prvků definujících vzhled stránky. K realizaci společných vizuálních prvků jsme museli vytvořit vlastní řídicí prvky. Ovšem s nástupem ASP.NET 2.0 jsme při definici vzorové stránky schopni zajistit úplnou vizuální dědičnost (ne tak dědičnost kódu ve smyslu OOP). Neexistence dědičnosti kódu není ve skutečnosti nijak vážným omezením, protože ke kódu vzorové stránky se můžeme dostat přes odkaz `MasterType`, jak jsme si vysvětlili výše.

Přepínání vzorových stránek za běhu

Poslední věcí, které se budeme v tomto úvodu ke vzorovým stránkám věnovat, je možnost dynamické změny vzorové stránky za běhu webové aplikace! Vskutku, můžeme mít několik vzorových stránek a po spuštění si vybrat, kterou chceme použít. Stačí v obslužné metodě události `PreInit` nastavit hodnotu vlastnosti stránky `MasterPageFile`:

```
protected void Page_PreInit(object sender, EventArgs e)
{
   this.MasterPageFile = "~/OtherMasterPage.master";
}
```

Událost `PreInit` je v prostředí ASP.NET 2.0 nová, přičemž vlastnost `MasterPageFile` lze nastavit pouze v ní, protože sloučení vzorové a obsahové stránky musí proběhnout v rané fázi životního cyklu webové stránky (v obslužné metodě události `Load` či `Init` by již bylo pozdě).

Při dynamické změně vzorové stránky se vždy musíme ujistit, že řídicí prvky typu `ContentPlaceHolder` mají ve všech vzorových stránkách totéž ID, jinak by se řídicí prvky typu `Content` definované v obsahových stránkách s nimi nepropojily. Díky této úžasné možnosti můžeme vytvořit několik vzorových stránek definujících naprosto odlišné uspořádání prvků na stránce, takže uživatelé si mohou zvolit přesně takové, které jim nejvíce vyhovuje. Nevýhodou tohoto postupu však je, že pokud k doprovodnému kódu jedné vzorové stránky připíšeme vlastní kód, pak jej musíme zkopírovat do souborů s doprovodnými kódy všech ostatních vzorových stránek. V opačném případě by jej obsahová stránka nemusela vždy najít. Navíc nemůžeme použít silně typovanou vlastnost Master, poněvadž typ vzorové stránky za běhu změnit nemůžeme. Lze jej totiž nastavit pouze direktivou `@MasterType`. Z uvedených důvodů nebudeme pro mechanismus změny rozvržení prvků používat odlišné vzorové stránky. Místo toho budeme mít pouze jednu, na níž budeme aplikovat různé soubory se styly. Přitom díky tomu, že jsme se rozhodli nepoužívat pro uspořádání tabulky, můžeme pomocí stylů zcela změnit vzhled (fonty, barvy, obrázky a pozice) našeho webu.

Tvorba sady volitelných motivů

Motivy jsou v prostředí ASP.NET 2.0 novým prvkem, který uživatelům nabízí větší kontrolu nad vzhledem webové stránky. Pomocí motivu lze definovat barevná schémata, názvy, velikosti a styly fontů, a dokonce obrázky (mají-li mít hranaté či zakulacené okraje, nebo prostě obrázky v různých barvách a s různým stínováním). Podpora „skinů" v ASP.NET 2.0 vznikla jako rozšíření principu kaskádových stylů. Každý uživatel si může zvolit motiv z dostupné nabídky, přičemž vybraný motiv určuje „skin", jenž přesně stanoví, jaká vizuálně-stylistická nastavení se mají v rámci daného sezení použít. Skiny jsou tedy jakýmisi příbuznými kaskádových stylů, k jejichž zpracování dochází na straně serveru. Soubor skinu je podobný CSS-souboru, ale narozdíl od něj může přepsat nejrůznější vizuální vlastnosti, jež byly pro řídicí prvky na straně serveru explicitně nastaveny v rámci stránky (globální CSS-specifikace nemůže nikdy přepsat lokální styl konkrétního řídicí prvku). Motivy mohou uchovávat speciální verze obrázků, což se může hodit, máme-li několik sad obrázku, které používají odlišná barevná schémata založená na aktuálním skinu. Nicméně motivy kaskádové styly nijak nenahrazují, spíše je vhodně doplňují. Při použití obou technologií lze dosáhnout skutečně vysoké míry flexibility a přitom mít vše pěkně pod kontrolou. Co se souborů se styly týče, v ASP.NET 2.0 nedošlo k žádným velkým změnám, snad až na pár řídicích prvků, kterým přibyla vlastnost CssClass, a několik řídicích prvků, pro něž lze ve vizuálním návrháři zvolit předpřipravený CSS-styl.

Motiv tvoří skupina vzájemně souvisejících souborů uložených v nějakém podadresáři adresáře `/App_Themes`, které mohou obsahovat následující položky:

- Soubory se styly (typu `.css`), které definují vzhled HTML-objektů.
- Soubory skinu, které definují vzhled řídicích prvků prostředí ASP.NET na straně serveru. Můžeme se na ně dívat jako na soubory se styly na straně serveru.
- Ostatní zdroje (např. obrázky).

Jedna ze skvělých věcí týkajících se způsobu, jakým ASP.NET 2.0 implementuje motivy, spočívá v tom, že při aplikaci motivu na stránku (popíšeme si později), vytvoří ASP.NET za běhu a zcela automaticky na všech stránkách meta-značku `<link>` pro každý soubor typu `.css`, který se nachází v adresáři motivu! To je výborné, protože stačí jen přejmenovat existující CSS-soubor nebo přidat nový, a rázem se s ním automaticky propojí všechny stránky. To je nesmírně důležité, protože, jak uvidíme později, při dynamické změně motivu (podobně jako při změně vzorové stránky) připojí ASP.NET soubory z adresáře nového motivu, čímž změní vzhled webu dle preferencí jednotlivých uživatelů. Bez tohoto mechanismu bychom museli za běhu ručně vytvářet meta-značky `<link>` podle toho, jaký motiv by uživatel zvolil, což by bylo velice obtížné.

Nejlepší novinkou v rámci motivů jsou styly na straně serveru, kterým se říká *skiny*. Jedná se o soubory s příponou `.skin`, které obsahují deklaraci řídicích prvků prostředí ASP.NET, která může vypadat například takto:

```
<asp:TextBox runat="server" BorderStyle="Dashed" BorderWidth="1px" />
```

Vše je stejné jako při normální deklaraci v souboru typu `.aspx`, jedinou výjimkou je, že řídicím prvkům nepřiřazujeme ID. Po aplikaci motivu na stránku dostanou její řídicí prvky vzhled dle definic v souboru skinu. Co se řídicího prvku typu `TextBox` týče, nemusí technika skinů zatím vypadat tak úžasně, poněvadž stejného výsledku bychom dosáhli vytvořením stylu pro HTML-element `<input>`. Nicméně ihned poté, co si uvědomíme, že to samé lze provádět i s mnohem složitějšími

řídicími prvky, jako je například `Calendar` či `DataGrid` (nebo nový `GridView`), začne celý mechanismus dávat větší smysl, neboť tyto prvky se nevztahují pouze k jednomu HTML-elementu, takže jejich styl nelze definovat tak snadno jen jedinou třídou v klasickém souboru se styly.

> Můžeme mít jediný soubor typu `.skin`, do něhož umístíme definice pro řídicí prvky jakéhokoliv typu, nebo si můžeme vytvořit samostatné skiny pro každý typ zvlášť (např. `TextBox.skin`, `DataGrid.skin`, `Calendar.skin` a podobně). K jejich sloučení dojde v paměti za běhu webové aplikace, takže jde jen o to, aby si každý uspořádal definice skinů podle vlastní chuti.

Pro aplikaci motivu na jedinou stránku stačí použít atribut `Theme` direktivy `@Page`:

```
<%@ Page Language="C#" Theme="NiceTheme" MasterPageFile="~/MasterPage.master"
... %>
```

Při aplikaci na všechny stránky můžeme využít atribut theme elementu `<pages>` v souboru `web.config`:

```
<pages theme="NiceTheme" masterPageFile="~/MasterPage.master" />
```

Stejně jako v případě vzorových stránek i motivy lze měnit z programového kódu, konkrétně z metody obsluhující událost `PreInit` ve třídě `Page`. Motiv, jehož jméno je uloženo v proměnné `Session`, lze aplikovat například takto:

```
protected void Page_PreInit(object sender, EventArgs e)
{
   if (this.Session["CurrentTheme"] != null)
      this.Theme = this.Session["CurrentTheme"];
}
```

> V kapitole 4 tento mechanismus vylepšíme nahrazením proměnné `Session` novou vlastností `Profile`.

> Při použití atributu `Theme` direktivy `@Page` (nebo atributu `theme` v souboru `web.config`) potlačí atributy vzhledu specifikované v souborech skinu své protějšky specifikované v souborech typu `.aspx`. Mají-li motivy fungovat podobně jako kaskádové styly (styly definované v souborech typu `.skin` půjdou v souborech typu `.aspx` přepsat), musíme motiv připojit přes atribut `StylesheetTheme` direktivy `@Page` nebo atribut `styleSheetTheme` elementu `<pages>` v souboru `web.config`. Atributy `Theme` a `StylesheetTheme` bychom si tedy v žádném případě neměli plést!

Dosud jsme probírali *anonymní* skiny – tedy skiny, které definují vzhled všech řídicích prvků daného typu. V některých případech však budeme potřebovat řídicí prvky s odlišným vzhledem, než jak jej definuje soubor skinu. Toho dosáhneme třemi různými způsoby:

1. Jak jsme si popsali výše, motiv můžeme aplikovat pomocí vlastnosti `StylesheetTheme` (namísto vlastnosti `Theme`), takže vizuální vlastnosti zapsané do souborů typu `.aspx` přepíší vzhled definovaný v souboru skinu. Nicméně výchozí chování mechanismu motivů zajišťuje, aby všechny řídicí prvky téhož typu měly stejný vzhled, což se hodí zejména v situacích, kdy na stránkách pracuje několik vývojářů, přičemž nemůžeme zajistit, že všichni budou používat atributy ve stránkách typu `.aspx`, pouze pokud to bude striktně vyžadováno.

2. Můžeme vypnout aplikaci motivu jen pro daný řídicí prvek a aplikovat na něj atributy vzhledu standardním způsobem:

```
<asp:TextBox runat="server" ID="btnSubmit" EnableTheming="False"
   BorderStyle="Dotted" BorderWidth="2px" />
```

3. Můžeme použít pojmenovaný skin, který oproti anonymnímu navíc obsahuje atribut `SkinID`:

```
<asp:Label runat="server" SkinID="FeedbackOK" ForeColor="green" />
<asp:Label runat="server" SkinID="FeedbackKO" ForeColor="red" />
```

   Při deklaraci řídicího prvku je třeba uvést odpovídající hodnotu vlastnosti `SkinID`:

```
<asp:Label runat="server" ID="lblFeedbackOK"
   Text="Zpráva byla úspěšně odeslána."
   SkinID="FeedbackOK" Visible="false" />

<asp:Label runat="server" ID="lblFeedbackKO"
   Text="Omlouváme se, ale při odesílání zprávy se vyskytl problém."
   SkinID="FeedbackKO" Visible="false" />
```

   Jedná se pravděpodobně o nejlepší variantu, neboť se nám tak otevírá cesta k definici několika vzhledů v jednom souboru pro tentýž typ řídicího prvku, které pak můžeme aplikovat na jakékoliv stránce. Ponecháme-li navíc definice všech stylů v souborech typu `.skin` místo toho, abychom je vkládali do samotných stránek, budeme schopni kompletně změnit vzhled webové stránky pouhým přepnutím aktuálního jména motivu (což bylo také důvodem zavedení motivů).

V části „Řešení“ této kapitoly použijeme motivy pro vytvoření několika různých vizuálních reprezentací téže vzorové stánky.

Tvorba navigačního systému

Jak jsme si již řekli v části „Problém“, potřebujeme vytvořit nějaký systém pro menu, který by se jednoduše udržoval a pro uživatele by byl snadno pochopitelný. Mohlo by se zdát, že stačí menu zapsat napevno ve formě HTML-kód, ale to není vhodná volba, protože v případě, že bychom chtěli mít menu na více stránkách nebo bychom jej potřebovali změnit, museli bychom jeho kód pokaždé kopírovat (tedy hlavičku a patičku). V první edici této knihy jsme vytvořili svůj vlastní řídicí prvek, který vzal XML-soubor obsahující mapu stránek (tj. jméno a URL-adresu odkazů pro zobrazení v menu) a pomocí XSL-souboru sestavil výsledný HTML-kód. Prostředí ASP.NET 2.0 zavádí některé nové řídicí prvky a vlastnosti, které dovedou v podstatě to samé, ale způsobem, který je pro vývojáře podstatně snazší.

Definice souboru s mapou webu

Položky menu specifikujeme v XML-souboru s mapou webu. Soubor s hlavní mapou webu, určený pro web jako celek, nese název `web.sitemap`. Tvoří jej hierarchická struktura uzlů `<siteMapPath>` s atributy `title` a `url`, která může vypadat například takto:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
```

```
   <siteMapNode title="Úvod" url="~/Default.aspx" />
   <siteMapNode title="O nás" url="~/About.aspx" />
   <siteMapNode title="Kontakt" url="~/Contact.aspx" />
</siteMap>
```

V případě našeho projektu budeme mít jak uzly první (např. „Úvod“, „Kontakt“ a „O nás“), tak I uzly druhé a možná i třetí úrovně (např. „Obchod / Košík“). Následuje o něco složitější příklad se záznamy na druhé úrovni, který se navíc odkazuje na soubor s o úroveň nižší mapou webu obsahující položky pro menu Internetového obchodu:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode title="Knihy" url="~/Books/Books.aspx">
      <siteMapNode title="Autoři" url="~/Books/Authors.aspx" />
      <siteMapNode title="Vydavatelství" url="~/Books/Publishers.aspx" />
   </siteMapNode>
   <siteMapNode title="DVD" url="~/DVDs/DVDs.aspx">
      <siteMapNode title="Filmy" url="~/DVDs/Movies.aspx" />
      <siteMapNode title="Hudba" url="~/DVDs/Songs.aspx" />
   </siteMapNode>
   <siteMapNode siteMapFile="~/StoreMenu.sitemap" />
</siteMap>
```

Webová mapa na nižší úrovni (`StoreMenu.sitemap`) má stejný formát jako soubor `web.sitemap` a obsahuje vnější uzel typu `siteMap`.

Vazba mapy webu na řídicí prvky menu

Poté, co jsme definovali soubor typu sitemap, můžeme jej použít jako datový zdroj pro nové řídicí prvky prostředí ASP.NET 2.0, mezi něž patří například `Menu` a `TreeView`. K dispozici máme též řídicí prvky nevizuálního charakteru jménem `DataSource`, které se dovedou připojit k databázi, XML-souboru nebo ke třídě komponenty. Využívají je grafické řídicí prvky pro získávání dat určených k zobrazení na obrazovce. Ve skutečnosti slouží jako most mezi vizuálním prvkem a právě používaným datovým úložištěm. Jedním z těchto řídicích prvků typu `DataSource` je i prvek typu `SiteMapDataSource`, který byl navržen zejména pro soubory s mapou webu. Lze jej definovat například takto:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

Všimněte si, že jsme nedefinovali cestu k souboru `web.sitemap`, což si můžeme dovolit díky tomu, že pro celý web existuje pouze jeden soubor se jménem `~/web.sitemap`. Při tvorbě map webu na nižší úrovni pro podadresáře umístěné v rámci adresářového stromu naší webové aplikace dochází k transparentnímu zpracování vazeb, neboť všechny začínají v souboru `web.sitemap`.

> V případě, že by se nám nelíbil způsob, jakým řídicí prvek typu `SiteMapDataSource` funguje (třeba proto, že chceme mít několik souborů typu sitemap nebo potřebujeme uložit mapu webu do databáze místo do XML-souborů), musíme si buď napsat vlastní řídicí prvek typu `DataSource`, nebo vytvořit novou třídu, která by transparentně poskytovala obsah pro prvek typu `SiteMapDataSource`.

Řídicí prvek typu `Menu` vytváří pomocí jazyka DHTML oblíbené vysouvací menu orientované buď vertikálně, nebo horizontálně. Ve verzi ASP.NET 1.1 neexistovaly žádné slušné a přitom standardní řídicí prvky pro menu a pravděpodobně každá společnost vyvíjející webové komponenty nabízela své vlastní řešení. Nicméně s příchodem ASP.NET 2.0 již máme zdarma k dispozici standardní prvky pro menu, které lze navíc integrovat s nejrůznějšími datovými zdroji. K vytvoření řídicího prvku typu `Menu` svázaného s prvkem `SiteMapDataSource` definovaného výše stačí jen jediný řádek:

```
<asp:Menu ID="mnuHeader" runat="server" DataSourceID="SiteMapDataSource1" />
```

Řídicí prvek typu `Menu` samozřejmě nabízí velké množství vlastností, s jejichž pomocí můžeme stanovit jeho orientaci (vlastnost `Orientation`), CSS-třídu (`CssClass`) či vzhled jednotlivých částí, počet vnitřních úrovní mapy, které se zobrazí při rozbalení menu (`StaticDisplayLevels`), a celou řadu dalších. Jejich kompletní popis však přesahuje rámec této knihy; zájemce o podrobnější informace tedy odkazujeme na oficiální MSDN-dokumentaci.

Zobrazení mapy webu ve formě stromu namísto vysouvacího menu je otázkou záměny deklarace řídicího prvku `Menu` za prvek `TreeView`:

```
<asp:TreeView runat="server" ID="tvwMenu" DataSourceID="SiteMapDataSource1" />
```

Navigační cesta

Kromě menu potřebujeme svým uživatelům také nabídnout nějaké vizuální vodítko poskytující informace o tom, kde se právě nacházejí a kudy vede cesta zpět na úvodní stránku. Tento problém se obvykle řeší pomocí *navigační cesty*, což je navigační lišta s odkazy na všechny stránky nebo sekce v hierarchii od úvodní až po aktuální stránku. Uživateli, který si právě prohlíží svůj nákupní košík, se může zobrazit například takto:

```
Úvod > Obchod > Košík
```

Nyní se uživatel může přesunout o dvě stránky zpět, aniž by ve svém prohlížeči musel stisknout tlačítko „Zpět“ (které nemusí být z mnoha důvodů viditelné) nebo se vracet na úvodní stránku a pokoušet se rozpomenout na stránky, které předtím procházel. S ASP.NET 2.0 můžeme na náš web přidat navigační cestu pomocí jediného řádku kódu, v němž deklarujeme instanci nového řídicího prvku typu `SiteMapPath`:

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server" />
```

A jako vždy má i tento řídicí prvek řadu vlastností, díky nimž si můžeme zcela přizpůsobit jeho vzhled, o čemž se blíže přesvědčíme v části „Řešení.“

Tvorba stránky přístupné všem skupinám uživatelů

Všechny standardně vestavěné řídicí prvky prostředí ASP.NET 2.0 generují ve výchozím stavu validní kód dle standardu XHTML 1.0 Transitional. XHTML je v podstatě HTML dodržující pravidla jazyka XML, takže pro jeho syntaxi platí mnohem přísnější pravidla. Například všechny hodnoty atributů musí být uzavřeny do uvozovek, všechny značky musí mít explicitní zakončení nebo k nim musí existovat odpovídající ukončovací značka (např. místo `<br>` a `<img src="...">` musí být `<br />` a `<img src="..." />`) a vnořené značky musí být uzavírány ve správném pořadí (např.

místo ahoj Pepo musí být ahoj Pepo). K tomu je celá řada HTML-značek, které sloužily pro formátování textu (např. <font>, nebo <s>), považována za zastaralé a měly by být nahrazeny CSS-styly (např. font-family: Verdana; text-align: center). Totéž platí pro některé atributy ostatních značek, jako je například width či align. Důvodem pro zavedení tohoto nového standardu je trend směřující k většímu oddělení vzhledu od vlastního obsahu (vysvětlili jsme si již dříve) a k tvorbě čistšího kódu – tedy kódu, který dovedou číst programy pracující s XML-daty. Skutečnost, že prostředí ASP.NET 2.0 automaticky generuje XHTML-kód, alespoň tedy dokud používáme jeho řídicí prvky, je pro vývojáře velkou úsporou času a činí tak proces adopce jazyka XHTML plynulejší. Oficiální dokumentace jazyka XHTML 1.0 z rukou organizace W3C je k dispozici na adrese http://www.w3.org/TR/xhtml1/.

Co se přístupnosti všem skupinám uživatelů týče, definuje organizace W3C sadu pravidel, která by měla usnadnit používání webu také handicapovaným uživatelům. Oficiální stránky nadepsané „Web Content Accessibility Guidelines 1.0“ (běžně odkazované jako WCAG) se nacházejí na adrese http://www.w3.org/TR/WCAG10/. Směrnice s názvem „Section 508“ vzešly z WCAG a musí být dodržovány webovými stránky vládních organizací v USA (více viz http://www.section508.gov/). Například všechny značky <img> musejí mít atribut alt poskytující alternativní text pro zrakově postižené uživatele, aby tak čtečky obrazovek dovedly popsat obrázek, a se vstupním polem je nutné vždy propojovat značku <label>. Implementace dalších směrnic je obtížnější, a navíc se přímo netýkají prostředí ASP.NET; pro více informací je vždy k dispozici oficiální dokumentace. Prostředí ASP.NET 2.0 usnadňuje dodržování některých jednodušších pravidel, tedy například takových, která jsme si uvedli výše. Tak například řídicí prvek Image má novou vlastnost GenerateEmptyAlternateTex, která při nastavení na hodnotu true generuje alt="" (naproti tomu nastavením AlternateText="" by se nevygeneroval nic). Řídicí prvek Label má novou vlastnost AssociatedControlID, která může obsahovat jméno vstupního řídicího prvku, pro nějž pak vygeneruje značku <label> (měla by se používat společně s vlastností AccessKey, čímž se ke vstupním polím vytvoří klávesové zkratky).

Zájemce o více informací o jazyku XHTML, přístupnosti pro všechny skupiny uživatelů a nových vlastnostech prostředí ASP.NET 2.0 týkajících se těchto oblastí odkazujeme na článek na Internetu: od Alexe Homera „Accessibility Improvements in ASP.NET 2.0 – Part 1“ (http:// www.15seconds.com/issue/040727.htm) a „Accessibility Improvements in ASP.NET 2.0 – Part 2“ (http://www.15seconds.com/issue/040804.htm) nebo od Stephena Walthera „Building ASP.NET 2.0 Web Sites Using Web Standards“ (http://msdn2.microsoft.com/en-us/library/aa479043.aspx).

Sdílení společného chování všemi stránkami

Vzorové stránky a motivy fungují skvěle pro sdílení téhož návrhu a vzhledu všemi stránkami webu. Nicméně někdy potřebujeme sdílet také společné chování, tedy kód, který má běžet v určité fázi životního cyklu stránky. Pokud například chceme zaznamenávat přístupy na všechny stránky, abychom mohli sestavovat a zobrazovat statistiky svého webu, musí se při načítání stránky spustit jistý kód. Další situace, kdy potřebujeme spustit nějaký kód na každé stránce, nastává například tehdy, chceme-li nastavit vlastnost stránky Theme v obsluze události PreInit. Faktem je, že společný kód bychom mohli umístit do externí funkce a do každé stránky prostě přidat řádek, který ji zavolá. Tento postup má však dva zásadní nedostatky:

- Při vytváření nové stránky bychom nikdy nesměli zapomenout na řádek s kódem spouštějícím onu externí funkci. Pokud by na stránkách pracovalo více vývojářů (což bývá velmi často), museli bychom zajistit, aby na to žádný z nich nezapomínal.
- Může se stát, že budeme chtít část inicializace spouštět při události PreInit a další část až při události Load. V takovém případě by bylo nutné napsat dvě samostatné metody (např. ve tvaru `xxxInitialize`) a do každé stránky přidávat ne jeden, ale více řádků, abychom tak zajistili volání patřičné metody ze správné obsluhy události. Neměli bychom se tedy spoléhat na to, že přidání jediného řádku na každou stránku je jednoduché, poněvadž se může stát, že později bude těchto řádků stále více. Máme-li stovky stránek, pak není sporu o tom, že přidávání nových řádků začne být v podstatě neproveditelné.

Tyto nevýhody jsou dostatečným důvodem k odmítnutí uvedeného řešení. Další možnost spočívá v přidání společného kódu ke kódu v pozadí vzorové stránky. V řadě situací se skutečně může jednat o vhodné řešení, což ovšem neplatí pro náš případ, protože v našem projektu musíme zpracovat událost `PreInit`, kterou však třída `MasterPage` (a její bázové třídy) nezná. Můžeme v ní zpracovat například událost `Init` nebo `Load`, událost `PreInit` však nikoli, takže musíme vymyslet něco jiného.

V předchozí edici této knihy jsme měli třídu `BasePage` a všechny obsahové stránky jsme odvozovali od ní, místo toho, aby dědily přímo od standardní třídy `System.Web.UI.Page`. Pravděpodobně se stále jedná o nejlepší řešení, poněvadž v rámci této třídy máme možnost reagovat na jakoukoliv událost jen tím, že přetížíme příslušnou metodu tvaru `OnXXX`, kde `XXX` je jméno požadované události.

Následující úryvek obsahuje základní kostru naší bázové třídy, kterou jsme odvodili od třídy `Page`, a která přetěžuje metody `OnPreInit` a `OnLoad`:

```
public class BasePage : System.Web.UI.Page
{
   protected override void OnPreInit(EventArgs e)
   {
      // zde můžeme přidat náš vlastní kód ...

      base.OnPreInit(e);
   }

   protected override void OnLoad(EventArgs e)
   {
      // zde můžeme přidat náš vlastní kód ...

      base.OnLoad(e);
   }
}
```

Třídy v souborech s doporovodnými kódy stránek pak namísto třídy `Page` odvodíme od naší třídy `BasePage`:

```
public partial class Contact : BasePage
{
   // kód standardní stránky ...
}
```

Nutnost úpravy tříd v doprovodných kódech každé stránky sice stále zůstává, avšak poté již stačí pracovat jen se třídou `BasePage`. Její metody můžeme upravovat nebo k nim přidávat nové a přitom již nemusíme sáhnout do doprovodného kódu žádné jiné stránky. Pokud tímto způsobem budeme postupovat od samého počátku vývoje webové aplikace, pak třídy v doprovodných kódech upravíme pouze jednou při jejich vytvoření, což je velmi jednoduché, a navíc tak položíme základ pro dobře udržovatelné a flexibilní řešení.

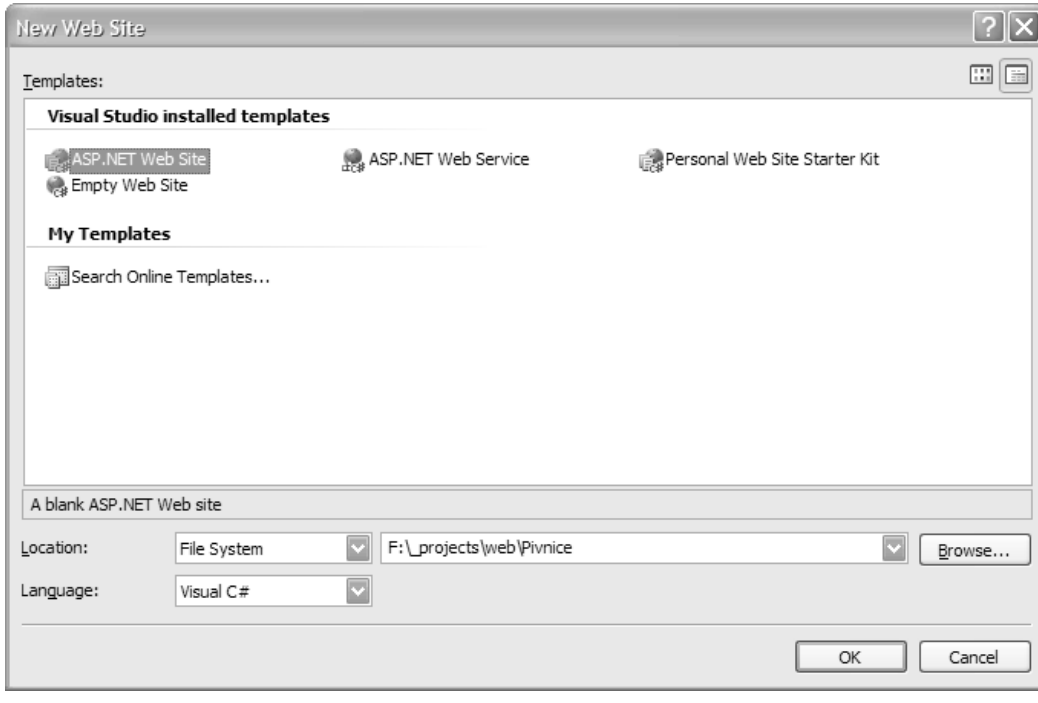
Řešení

V tuto chvíli bychom již měli mít jasnou představu o tom, na čem budeme pracovat a jak budeme postupovat. Pusťme se tedy do toho! V dřívějších částech této kapitoly jsme si vysvětlili, jak pomocí grafických aplikací, jako je Photoshop či Paint Shop Pro, vytvoříme maketu našeho webu, kterou můžeme uložit třeba ve formátu PSD. Poté, co je maketa schválena a můžeme začít programovat, ji nejdříve musíme rozřezat na obrázky typu `.gif` a `.jpg`, které již lze přímo použít na webové stránce. Bez ohledu na způsob jejich vytvoření je nyní můžeme vzít a použít při budování webové stránky. První krok spočívá v založení nového projektu webové aplikace, následuje vytvoření vzorové a úvodní stránky a výchozího motivu. Později můžeme vyvinout druhý motiv a implementovat mechanismus pro jejich přepínání za běhu webové aplikace.

Nejdříve tedy v prostředí aplikace Visual Studio .Net 2005 založíme nový projekt webové aplikace (FILE > NEW > WEB SITE > ASP.NET WEB SITE). Zde se setkáváme s další novou vlastností: můžeme vytvořit projekt stanovením adresáře v souborovém systém (místo specifikace lokace na webu), což provedeme zvolením položky FILE SYSTEM v roletovém menu LOCATION (viz obrázek 2.3).

Tímto způsobem můžeme v prostředí ASP.NET založit projekt, aniž bychom museli v IIS-metabázi vytvářet příslušnou virtuální aplikaci nebo virtuální adresář (metabáze je místo, kam si IIS ukládá konfigurační data). Přitom se projekt načítá ze skutečného adresáře na disku, a spouští se pomocí integrovaného odlehčeného webového serveru (jménem ASP.NET Development Server), který zpracovává požadavky na jiném TCP/IP portu, než který pro tento účel používá IIS (IIS používá port číslo 80). Aktuální číslo portu se určuje náhodně při každém stisku klávesy F5, která slouží pro spouštění webové aplikace v ladicím režimu. Požadavky, které dokáže zpracovávat, mohou mít například tvar `http://localhost:1168/ProjName/Default.aspx`. Díky této novince lze projekty mnohem snadněji přesouvat a zálohovat, stačí totiž jen zkopírovat obsah příslušného adresáře (není již nutné cokoliv nastavovat z konzoly pro správu IIS). Dokud nepřejdeme k nasazení aplikace na webový server systému IIS nebo při vytváření projektu místo lokální cesty nezadáme URL-adresu, tak Visual Studio 2005 ve skutečnosti systém IIS ani nepotřebuje.

Každý, kdo pracoval s jakoukoli dřívější verzí prostředí ASP.NET či VS2005, jistě tuto novou možnost přivítá. Zůstává však jen na vývojáři, kterou cestu si zvolí, stále má totiž možnost vytvořit projekt s cestou ve formě URL-adresy (tedy vytvořit a provozovat web přímo pod IIS). Stačí, když v roletovém menu LOCATION zvolí položku HTTP. Nejlepší možností je vyvíjet webovou aplikaci v lokálním adresáři a používat integrovaný webový server a až v testovací fázi přejít na plně vybavený webový server IIS. VS2005 obsahuje nového pomocníka pro nasazení webové aplikace, který usnadňuje nasazení kompletního řešení na místní či vzdálený webový server IIS. Prozatím tedy postačí, když novou webovou aplikaci umístíme do adresáře s názvem `TheBeerHouse`.



Obrázek 2.3: Založení nového projektu webové aplikace v prostředí aplikace Visual Studio .NET 2005

> Integrovaný webový server byl vyvinut pro usnadnění vývoje a rychlého testování. Neměli bychom však s jeho pomocí nikdy provádět finální testování v rámci fáze zajištění kvality a integrace, k čemuž bychom měli výhradně používat webový server IIS. Obsahuje totiž mnohem více prvků, jako je vyrovnávací paměť a HTTP-komprese, a řadu bezpečnostních nastavení, což může způsobit, že webová aplikace poběží oproti integrovanému serveru poněkud odlišně.

Po vytvoření nové webové aplikace můžeme soubor `Default.aspx` klidně smazat. Výchozí stránku si totiž za chvíli navrhneme sami.

Implementace návrhu stránky

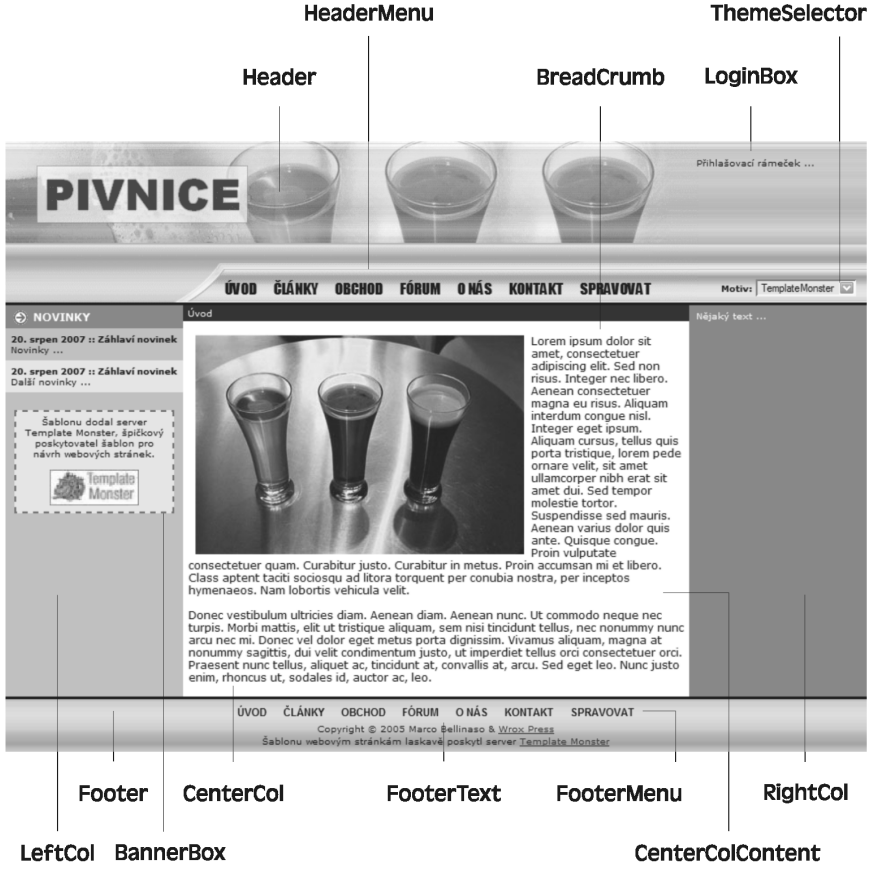
Tvorba vzorové stránky se sdílenou grafickou podobou není v případě, že máme obrázek (nebo sadu obrázků) makety, zase až tak obtížná. V podstatě stačí vyříznout logo a další grafické prvky a vložit je na HTML-stránku. Ostatní části uspořádání stránky (např. lišta s menu, sloupky a patička) můžeme snadno realizovat pomocí HTML-elementů `<div>`. Šablona, kterou nám poskytl server TemplateMonster (a kterou jsme malinko upravili a rozšířili), je na obrázku 2.4.

Obrázek 2.4: Mírně upravená a rozšířená šablona ze serveru TemplateMonster

Z tohoto obrázku můžeme vyříznou celou lištu s hlavičkou a umístit přes ni několik DIV-kontejnerů. Jeden pro odkazy menu, jeden pro přihlašovací rámeček a další pro přepínač motivů (roletové menu obsahující názvy dostupných motivů). Všechny umístíme do absolutních pozic, takže budou přesně tam, kde je chceme mít. Správné souřadnice jejich levého či pravého horního rohu určíme velice snadno: stačí nad ně v grafickém editoru umístit ukazatel myši a přímo opsat zobrazené x-ové a y-ové hodnoty.

Patičku tvoří element typu DIV s pozadím ve formě plátků obrázků o šířce 1 pixel opakovaně umístěných horizontálně vedle sebe. Dále obsahuje pár pod-elementů typu DIV: jeden pro odkazy menu (stejné jako menu v hlavičce) a druhý pro upozornění o ochraně autorských práv.

Nakonec nám zůstala oblast stránky s obsahem rozděleným na tři sloupky. Prostřední má pravé a levé okraje stanovené na 200 pixelů a vyplněné dvěma dalšími elementy typu DIV, které jsou pomocí absolutních pozic přilepené na okrajích stránky. Obrázek 2.5 zachycuje výše uvedené elementy aplikované na předchozí obrázek.



Obrázek 2.5: Uspořádání DIV-kontejnerů na stránce

Tvorba vzorové stránky

U čtenářů této knihy se předpokládá jistá obeznámenost s prostředím ASP.NET a Visual Studio .NET. Konkrétně se má za to, že čtenář umí na základní úrovni pracovat s kteroukoli předchozí verzí aplikace Visual Studio .NET. Z toho plyne, že při krocích, které si zde budeme vysvětlovat, se zaměříme pouze na nové prvky zavedené od verze 2.0, aniž bychom se přitom zabývali každou z dílčích podrobností. V případě, že uvedené vysvětlení nestačí k plnému pochopení prováděných činností, doporučujeme nejdříve prostudovat nějakou knihu o ASP.NET pro začátečníky (např. „*Naučte se ASP.NET za 21 dní*", kterou si můžete objednat na adrese `http://knihy.cpress.cz/knihy/pocitacova-literatura/programovani/naucte-se-asp-net-za-21-dni/`).

Po vytvoření webové stránky dle výše uvedeného popisu, nyní vytvoříme soubor s novou vzorovou stránkou (zvolíme položku MASTER PAGE v menu WEBSITE > ADD NEW ITEM a nový soubor pojmenujeme `Template.master`), načež lze na její povrch pomocí vizuálního návrháře přidat řídicí prvky prostředí ASP.NET na straně serveru a statické HTML-elementy. Nicméně při práci s DIV-kontejnery

a samostatnými soubory se styly není vizuální návrhář schopen poskytnout potřebnou míru flexibility. Bude tak pro nás snazší pracovat přímo v okně SOURCE VIEW a psát zdrojový kód ručně. Jak bylo již dříve řečeno, tvorba vzorové stránky se od tvorby normální stránky příliš neliší. Nejnápadnějším rozdílem je direktiva `@Master` v horní části souboru a přítomnost řídicích prvků typu `ContentPlaceHolder` na místech, kam `.aspx`-stránky umístí svůj obsah. Následuje kód definující standardní meta-značky jazyka HTML a hlavičku webové stránky pro soubor `Template.master`:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Template.master.cs"
   Inherits="TemplateMaster" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

<head id="Head1" runat="server">
   <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
   <title>Pivnice</title>
</head>

<body>
<form id="Main" runat="server">
   <div id="header">
      <div id="header2">
         <div id="headermenu">
            <asp:SiteMapDataSource ID="SiteMapDataSource1"
               runat="server" StartingNodeOffset="0" />
            <asp:Menu ID="mnuHeader" runat="server"
               CssClass="headermenulink"
               DataSourceID="SiteMapDataSource1"
               Orientation="Horizontal"
               MaximumDynamicDisplayLevels="0"
               SkipLinkText=""
               StaticDisplayLevels="2" />
         </div>
      </div>
      <div id="loginbox">Přihlašovací rámeček ...</div>
      <div id="themeselector">Přepínač motivů ...</div>
   </div>
```

Všimněte si, že v první ukázce kódu není nic, co by souviselo se skutečným vzhledem hlavičky. To je dáno tím, že vzhled kontejnerů, textu a dalších objektů určíme až v souborech se styly a skinem stránky. Kontejner LOGINBOX prozatím ponecháme prázdný (naplníme jej v kapitole 4, v níž budeme probírat bezpečnost a registraci členů). Rámeček THEMESELECTOR naplníme v pozdější části této kapitoly, ihned poté, co vyvineme řídicí prvek, který zobrazuje dostupné styly, z nichž si uživatel může vybírat. DIV-element s názvem headermenu obsahuje řídicí prvek typu `SiteMapPathDataSource`, který načítá obsah souboru `Web.sitemap`, který zanedlouho také vytvoříme. Kromě jiného obsahuje řídicí prvek typu `Menu`, jehož datovým zdrojem pro tvorbu položek je `SiteMapPathDataSource`.

Pokračujeme dalšími elementy typu DIV pro prostřední část stránky se třemi sloupky:

```
<div id="container">
   <div id="container2">
      <div id="rightcol">
         <div class="text">Nějaký text ...</div>
            <asp:ContentPlaceHolder ID="RightContent" runat="server" />
      </div>
      <div id="centercol">
         <div id="breadcrumb">
            <asp:SiteMapPath ID="SiteMapPath1" runat="server" />
         </div>
         <div id="centercolcontent">
            <asp:ContentPlaceHolder ID="MainContent" runat="server">
               <p> </p><p> </p><p> </p><p> </p>
               <p> </p><p> </p><p> </p><p> </p>
            </asp:ContentPlaceHolder>
         </div>
      </div>
   </div>
   <div id="leftcol">
      <div class="sectiontitle">
         <asp:Image ID="imgArrow1" runat="server"
            ImageUrl="~/images/arrowr.gif"
            ImageAlign="left" hspace="6" />Novinky
      </div>
      <div class="text">
         <b>20. srpen 2007 :: Záhlaví novinek</b><br />
         Novinky ...
      </div>
      <div class="alternatetext">
         <b>20. srpen 2007 :: Záhlaví novinek</b><br />
         Další novinky ...
      </div>
      <asp:ContentPlaceHolder ID="LeftContent" runat="server" />
      <div id="bannerbox">
         <a href="http://www.templatemonster.com" target="_blank">
            Šablonu dodal server Template Monster,
            špičkový poskytovatel šablon pro návrh webových stránek.
            <br /><br />
            <asp:Image runat="server" ID="TemplateMonsterBanner"
               ImageUrl="~/images/templatemonster.jpg" Width="100px" />
         </a>
      </div>
   </div>
</div>
```

V předchozím kódu jsme definovali tři řídicí prvky typu `ContentPlaceHolder`, pro každý sloupek jeden. Díky tomu může obsahová stránka umístit text do tří různých pozic. Pamatujme si, že jejich vyplnění je nepovinné a v některých případech budeme mít stránky, které přidají obsah pouze do prostředního sloupku s tím, že ostatní dva sloupky si ponechají svůj výchozí obsah definovaný ve vzorové stránce. Prostřední sloupek navíc obsahuje několik pod-elementů typu DIV s řídicím prvkem typu `SiteMapPath` pro navigační cestu.

Ve zbývající části vzorové stránky definujeme kontejner pro patičku s pod-kontejnery pro menu patičky (až na aplikovaný styl přesně kopírující menu v hlavičce) a upozornění o ochraně autorských práv:

```
   <div id="footer">
      <div id="footermenu">
         <asp:Menu ID="mnuFooter" runat="server"
            style="margin-left:auto; margin-right:auto;"
            CssClass="footermenulink"
            DataSourceID="SiteMapDataSource1"
            Orientation="Horizontal"
            MaximumDynamicDisplayLevels="0"
            SkipLinkText=""
            StaticDisplayLevels="2" />
      </div>
      <div id="footertext">
         <small>Copyright &copy; 2007 Marco Bellinaso &amp
         <a href="http://www.wrox.com" target="_blank">Wrox Press</a><br />
         Šablonu webovým stránkám laskavě poskytl server
         <a href="http://www.templatemonster.com" target="_blank">
            Template Monster</a></small>
      </div>
   </div>
</form>
</body>
</html>
```

Poznámka ohledně kompatibility prohlížečů: DIV-element `footer` deklarovaný ve výše uvedeném kódu bude mít text zarovnaný na střed. Ovšem v něm deklarovaný řídicí prvek typu `Menu` se za běhu vygeneruje jako HTML-tabulka, přičemž tabulky nejsou v prohlížečích, jako je Firefox, považovány za text, takže nebudou zarovnány jako text. Protože vyvíjíme pro oba prohlížeče, musíme zajistit, aby v obou byla tabulka zarovnána na střed. Z toho důvodu jsme do deklarace řídicího prvku typu `Menu` přidali atribut `style`, který na obě strany tabulky přidá shodné, maximálně široké okraje. Výsledkem tak bude tabulka zarovnaná horizontálně. Atribut `style` se nemapuje na vlastnost na straně serveru nabízenou řídicím prvkem. Vypadá to, že neexistuje žádná vlastnost nebo něco podobného, co se za běhu transformuje na „style“, takže jsme tímto způsobem vlastně použili HTML-atribut. Díky tomu, že se nemapuje na žádnou vlastnost řídicího prvku, bude k vygenerované HTML-tabulce přidán beze změn.

Tvorba souboru s mapou webu

Díky jednoduchosti přidávání, odebírání a úpravy odkazů v menu při využití souboru s mapou webu a řídicího prvku typu `SiteMapPath` se v současné chvíli nemusíme zabývat tím, jaké odkazy budeme vlastně potřebovat. Tyto údaje totiž bez problémů doplníme později. Nyní můžeme přidat několik zkušebních odkazů a posléze soubor upravit. K projektu tedy připojíme soubor `Web.sitemap` (položka SITE MAP v menu WEBSITE > ADD NEW ITEM ...), do něhož umístíme následující XML-uzly:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode title="Úvod" url="~/Default.aspx">
      <siteMapNode title="Obchod" url="~/Store/Default.aspx">
         <siteMapNode title="Košík" url="~/Store/ShoppingCart.aspx" />
      </siteMapNode>
      <siteMapNode title="Fórum" url="~/Forum/Default.aspx" />
      <siteMapNode title="O nás" url="~/About.aspx" />
      <siteMapNode title="Kontakt" url="~/Contact.aspx" />
      <siteMapNode title="Správa" url="~/Admin/Default.aspx" />
   </siteMapNode>
</siteMap>
```

Nyní by mohla zaznít otázka: Proč je uzel Úvod nadřazen ostatním uzlům a není na stejné úrovni jako ony? Tato možnost tu samozřejmě také je, ale my potřebujeme, aby řídicí prvek typu `SiteMapPath` vždy před zbývající částí cesty zobrazoval odkaz Úvod. Kvůli tomu musí být kořenovým uzlem. Ve skutečnosti řídicí prvek typu `SiteMapPath` nefunguje tak, že by si pamatoval předchozí stránky, ale jen se podívá do souboru s mapou webu na XML-uzel popisující aktuální stránku a zobrazí odkazy všech jemu nadřazených uzlů.

> Pokud nasadíme projekt do kořenového adresáře webového prostoru prostředí IIS, můžeme se na kořenový adresář našeho projektu odkazovat pomocí znaku lomítko („/“). Nasadíme-li však náš web do nějaké virtuální podsložky, musíme se na něj odkazovat jinak. V mapě webu uvedené výše můžete vidět, že pro tento účel používáme dvojici znaků „~/“. Tyto části URL-adresy se rozvinou až za běhu aplikace dle jejího umístění, takže vše funguje, jak má, bez ohledu na to, zdali se stránky nacházejí v kořenovém nebo v nějakém virtuálním adresáři.
>
> Integrovaný webový server spouští daný web vždy, jako by byl nasazen do virtuálního adresáře, přičemž jeho URL-adresa obsahuje jméno projektu. To nutí vývojáře používat v URL-adrese znaky „~/“, čímž se minimalizují problémy s odlišným umístěním při nasazení.

Tvorba prvního motivu

Nyní je vhodný čas pro vytvoření motivu pro vzorovou stránku, který nazveme `TemplatehMonster`. Můžeme to provést dvěma způsoby, z nichž oba jsou shodné, co se výsledné funkčnosti týče. Mohli bychom k projektu přidat nový adresář jménem `App_Themes` a do něj nový podadresář nazvaný `TemplateMonster`. Nebo si můžeme nechat asistovat od VS2005: zvolíme položku THEME FOLDER v menu WEBSITE > ADD FOLDER a nový adresář nazveme `TemplateMonster` (adresář `App_Themes` bude vytvořen automaticky). Adresář `App_Themes` hraje zvláštní roli, neboť má vyhra-

zené jméno a v okně SOLUTION EXPLORER je zašedlý. Označíme tedy tento adresář a přidáme do něj nový soubor se styly (zvolíme položku STYLESHEET v menu WEBSITE > ADD NEW ITEM a soubor nazveme `Default.css`). Název souboru není důležitý, protože s `.aspx`-stránkou se za běhu automaticky propojí všechny CSS-soubory v adresáři.

Následující kód obsahuje část tříd stylů definovaných v souboru `Default.css` (celý soubor se nachází mezi zdrojovými kódy k této knize, které lze stáhnout z Internetu):

```
body
{
   margin: 0px; font-family: Verdana; font-size: 12px;
}

#container
{
   background-color: #818689;
}

#container2
{
   background-color: #bcbfc0; margin-right: 200px;
}

#header
{
   padding: 0px; margin: 0px; width: 100%; height: 184px;
   background-image: url(images/HeaderSlice.gif);
}

#header2
{
   padding: 0px; margin: 0px; width: 780px; height: 184px;
   background-image: url(images/Header.gif);
}

#headermenu
{
   position: relative; top: 153px; left: 250px; width: 500px;
   padding: 2px 2px 2px 2px;
}

#breadcrumb
{
   background-color: #202020; color: White; padding: 3px; font-size: 10px;
}

#footermenu
```

```
{
   text-align: center; padding-top: 10px;
}

#loginbox
{
   position: absolute; top: 16px; right: 10px; width: 180px; height: 80px;
   padding: 2px 2px 2px 2px; font-size: 9px;
}

#themeselector
{
   position: absolute; text-align: right; top: 153px; right: 10px;
   width: 180px; height: 80px; padding: 2px 2px 2px 2px; font-size: 9px;
}

#footer
{
   padding: 0px; margin: 0px; width: 100%; height: 62px;
   background-image: url(images/FooterSlice.gif);
}

#leftcol
{
   position: absolute; top: 184px; left: 0px; width: 200px;
   background-color: #bcbfc0; font-size: 10px;
}

#centercol
{
   position: relative inherit; margin-left: 200px; padding: 0px;
   background-color: white; height: 500px;
}

#centercolcontent
{
   padding: 15px 6px 15px 6px;
}

#rightcol
{
   position: absolute; top: 184px; right: 0px; width: 198px; font-size: 10px;
   color: White; background-color: #818689;
}

.footermenulink
```

```
{
   font-family: Arial; font-size: 10px; font-weight: bold;
   text-transform: uppercase;
}

.headermenulink
{
   font-family: Arial Black; font-size: 12px; font-weight: bold;
   text-transform: uppercase;
}

/* další styly jsme kvůli stručnosti vypustili */
```

Povšimněte si, jak jisté elementy (jako `#loginbox`, `#themeselector`, `#leftcol` a `#rightcol`) používají absolutní umístění. Dále si všimněte, že pro hlavičku existují dva různé kontejnery s dvěma různými styly. První (`#header`) využívá šířku celé stránky (není-li explicitně specifikován atribut `width` a nepoužíváme-li absolutní umísťování, bude DIV-element vždy zaujímat 100 % šířky stránky) a má na pozadí obrázek široký 1 pixel, který se (implicitně, ve výchozím stavu) horizontálně opakuje. Druhý (`#header2`) je stejně velký jako obrázek v souboru `Header.gif`, který používá jako své pozadí, přičemž je umístěn nad prvním kontejnerem. První kontejner tedy slouží k tomu, aby pozadí druhého kontejneru s fixní šířkou pokračovalo po celé šířce stránky. To je nezbytné kvůli tomu, že chceme, aby rozvržení prvků na stránce bylo dynamické, a vyplňovalo tak celou šířku stránky. Pokud bychom použili rozvržení s pevně stanovenou šířkou, pak by nám stačil jen jediný kontejner.

> Všechny obrázky, na které se odkazujeme v souboru se styly, se nacházejí v podadresáři `Images` adresáře `App_Themes/TemplateMonster`. Tímto způsobem máme pohromadě všechny související objekty, které tvoří jeden motiv.

Nyní přidáme soubor se skinem jménem `Controls.skin` (označíme adresář `TemplateMonster` a zvolíme položku SKIN FILE v menu WEBSITE > ADD NEW ITEM). Všechny serverové styly budeme vkládat do tohoto souboru, budou se tedy aplikovat na řídicí prvky všech typů. Mohli bychom sice pro každý typ řídicího prvku vytvořit zvláštní soubor, ale díky jedinému souboru je jejich údržba snazší. Následující kód obsahuje dva anonymní skiny pro řídicí prvky typu `TextBox` a `SiteMapPath` a dva pojmenované skiny (`SkinID`) pro řídicí prvek typu `Label`:

```
<asp:TextBox runat="server" BorderStyle="dashed" BorderWidth="1px" />

<asp:Label runat="server" SkinID="FeedbackOK" ForeColor="green" />

<asp:Label runat="server" SkinID="FeedbackKO" ForeColor="red" />

<asp:SiteMapPath runat="server">
   <PathSeparatorTemplate>
      <asp:Image runat="server" ImageUrl="images/sepwhite.gif"
         hspace="4" align="middle" />
   </PathSeparatorTemplate>
</asp:SiteMapPath>
```

Účel prvních tří skinů je čistě demonstrativní, protože stejného efektu bychom dosáhli i při použití klasických CSS-stylů. Skin pro řídicí prvek typu `SiteMapPath` již CSS-styly tak snadno nahradit nelze, poněvadž se z něj nevygeneruje jen jediný HTML-element. Pomocí výše uvedeného skinu tedy deklarujeme, co se má použít jako oddělovač odkazů směřujících k aktuální stránce (zvolili jsme obrázek představující šipku).

Tvorba zkušební stránky Default.aspx

Nyní, kdy již máme vytvořenou kompletní vzorovou stránku a motiv, můžeme přistoupit k otestování pomocí zkušební obsahové stránky. Nejdříve do projektu přidáme novou webovou stránku nazvanou `Default.aspx` (v okně SOLUTION EXPLORER zvolíme položku WEB FORM v menu WEBSITE > ADD NEW ITEM), v dialogu ADD NEW ITEM označíme zaškrtávací pole s názvem SELECT MASTER PAGE, čímž se nám otevře druhý dialog, ze kterého můžeme vybrat vzorovou stránku (vybereme `Template.master`). Díky této volbě obsahuje naše nová stránka pouze řídicí prvky typu `Content`, které odpovídají řídicím prvkům typu `ContentPlaceHolder` ve vzorové stránce, a ne značky `<html>`, `<body>`, `<head>` a `<form>`, které jsou jinak standardně její součástí. Do středového řídicího prvku typu `ContentPlaceHolder` nyní můžeme vložit nějaký obsah:

```
<%@ Page Language="C#" AutoEventWireup="true"
   MasterPageFile="~/Template.master" CodeFile="Default.aspx.cs"
   Inherits="_Default" Title="Pivnice" %>

<asp:Content ID="Content1" ContentPlaceHolderID="RightContent" Runat="Server">
</asp:Content>

<asp:Content ID="MainContent" runat="server"
   ContentPlaceHolderID="MainContent">
   <asp:Image ID="imgBeers" runat="server"
      ImageUrl="~/Images/3beers.jpg" ImageAlign="left" hspace="8" />
   Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="LeftContent" Runat="Server">
</asp:Content>
```

Atribut `Theme` s hodnotou `TemplateMonster` bychom mohli přidat do direktivy `@Page`. Místo toho je však vhodnější provést toto nastavení v souboru `web.config`, a to pouze jednou, odkud se pak bude aplikovat na všechny stránky. V menu WEBSITE > ADD NEW ITEM tedy zvolíme položku WEB CONFIGURATION FILE. Z kódu stránky `Default.aspx` odstraníme atribut `MasterPageFile`, který také umístíme do souboru `web.config`:

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>
      <pages theme="TemplateMonster" masterPageFile="~/Template.master" />
      <!-- ostatní nastavení ... -->
   </system.web>
</configuration>
```

Proč při vytváření souboru `Default.aspx` vybírat v dialogu NEW ITEM vzorovou stránku, když ihned poté odstraňujeme atribut `MasterPageFile`? Protože tímto způsobem nám prostředí VS2005 vytvoří správné řídicí prvky typu `Content`, a ne klasickou stránku s HTML-kódem.

Vytvoření druhého motivu

K otestování mechanismu přepínatelných motivů popsaného v dřívější části kapitoly musíme mít více než jen jeden motiv. V adresáři `App_Themes` tudíž vytvoříme další podadresář pojmenovaný `PlainHtmlYellow` (vybereme náš projekt, klepneme pravým tlačítkem v menu ADD FOLDER a zvolíme položku THEME FOLDER), kam z adresáře `TemplateMonster` zkopírujeme celý soubor `Default.css`, který vzápětí upravíme tak, aby výsledný motiv vypadal odlišně. V našem příkladě jsme změnili většinu kontejnerů, takže na pozadí nezůstal žádný obrázek a hlavička i patička jsou vyplněné jednolitými barvami, stejně jako sloupky vpravo a vlevo. Kromě velikosti se některé elementy liší také svou pozicí. Konkrétně levý a pravý sloupek (používaly absolutní pozice) si nyní své umístění prohodí, takže kontejner leftcol bude přilepen u pravého a kontejner rightcol u levého okraje. Stačí změnit jen pár tříd definujících styly:

```
#leftcol
{
   position: absolute;
   top: 150px;
   right: 0px;
   width: 200px;
   background-color: #ffb487;
   font-size: 10px;
}

#rightcol
{
   position: absolute;
   top: 150px;
   left: 0px;
   width: 198px;
   color: White;
   background-color: #8d2d23;
   font-size: 10px;
}
```

Na tomto příkladě můžeme vidět sílu DIV-elementů a kaskádových stylů: změníme několik málo stylů a obsah, který byl původně na levé straně stránky, se přesune napravo. Jedná se o poměrně jednoduchý příklad, ale můžeme jít mnohem dál a prvky na stránce uspořádat úplně jinak (např. některé skrýt nebo jiné zvětšit).

Co se souboru s definicí skinu týče, stačí z adresáře `TemplateMonster` zkopírovat soubor `controls.skin` a vymazat definici řídicích prvků `TextBox` a `SiteMapPath`, které se tak vrátí k původnímu vzhledu. Rozdíl pak bude patrný při změně motivu za běhu webové aplikace. Pokud bychom v budoucnu chtěli na tyto prvky aplikovat jiný než výchozí vzhled, stačí jít zpět a přidat do tohoto souboru definici nového stylu, aniž by bylo nutné modifikovat cokoliv jiného.

Tvorba vlastního řídicího prvku ThemeSelector

Nyní máme vzorovou stránku s párem motivů, takže se můžeme pustit do vývoje vlastního řídicího prvku, který zobrazí seznam dostupných motivů a umožní uživateli, aby si jeden z nich vybral. Jakmile jej budeme mít, vložíme jej do vzorové stránky, konkrétně do DIV-kontejneru themeselector. Nejdříve však musíme vytvořit nový adresář s názvem `Chontrols`, do něhož budeme ukládat všechny námi vytvořené řídicí prvky, aby se nám nepletly mezi stránky a lépe se nám s nimi pracovalo (vybereme náš projekt, klepneme pravým tlačítkem a v menu ADD FOLDER zvolíme položku REGULAR FOLDER). K vytvoření nového řídicího prvku klepneme pravým tlačítkem na adresář `Controls`, v menu ADD NEW ITEM zvolíme položku WEB USER CONTROL a zadáme jméno `ThemeSelector.ascx`. Obsah souboru je velice jednoduchý a zahrnuje pouze řetězec a řídicí prvek typu `DropDownList`:

```
<%@ Control Language="C#" AutoEventWireup="true"
   CodeFile="ThemeSelector.ascx.cs"
   Inherits="ThemeSelector" %>
<b>Motiv:</b>
<asp:DropDownList runat="server" ID="ddlThemes" AutoPostBack="true" />
```

Všimněte si, že roletové menu má vlastnost `AutoPostBack` nastavenou na hodnotu true, takže ihned poté, co uživatel vybere novou položku, dojde k automatickému odeslání celé stránky na server. Skutečnou práci, která sestává z naplnění roletového menu názvy dostupných motivů a načtení zvoleného motivu, bude provádět kód v souboru s doprovodným kódem řídicího prvku a ve třídě bázové stránky, kterou za okamžik uvidíme. V souboru s doprovodným kódem musíme naplnit roletové menu polem řetězců, které vrátí pomocná metoda, a poté zvolit položku, která má tutéž hodnotu jako vlastnost `Theme` aktuální stránky:

```
public partial class ThemeSelector : System.Web.UI.UserControl
{
   protected void Page_Load(object sender, EventArgs e)
   {
      ddlThemes.DataSource = Helpers.GetThemes();
      ddlThemes.DataBind();

      ddlThemes.SelectedValue = this.Page.Theme;
   }
}
```

Metoda `GetTheme` je definována v souboru `Helpers.cs`, který se nachází v dalším speciálním adresáři s názvem `App_Code`. Soubory v něm uložené engine ASP.NET automaticky za běhu přeloží, takže je není potřeba překládat před spuštěním projektu. Soubory se zdrojovými kódy jazyka C# je dokonce možné upravovat i za běhu webové aplikace, provést znovunačtení stránky a nový požadavek zapříčiní opětovné přeložení upraveného souboru do nového dočasného úložiště a načtení do paměti. O novém kompilačním modelu si více řekneme v pozdější části knihy, zejména pak v kapitole 12, kde budeme probírat nasazení webové aplikace.

V metodě `GetThemes` používáme k získání pole s cestami ke všem podadresářům adresáře `~/App_Themes` metodu GetDirectories třídy System.IO.Directory (této metodě musíme předat fyzickou cestu, a ne URL-adresu – fyzickou cestu ovšem můžeme získat z URL-adresy pomocí metody

`Server.MapPath`). Získané pole řetězců obsahuje celé cesty, ne jen názvy adresářů, takže musíme každou položku přepsat pouze částí obsahující název adresáře (voláním statické metody `System.IO.Path.GetFileName`). Jakmile se pole poprvé naplní, uloží se do vyrovnávací paměti prostředí ASP.NET, takže následné požadavky jej odtud získají mnohem rychleji. Následující kód zachycuje celou definici třídy Helpers (`App_Code\Helpers`):

```
namespace MB.TheBeerHouse.UI
{
   public static class Helpers
   {
      /// <summary>
      /// Vrací pole s názvy všech lokálních motivů
      /// </summary>
      public static string[] GetThemes()
      {
         if (HttpContext.Current.Cache["SiteThemes"] != null)
         {
            return (string[])HttpContext.Current.Cache["SiteThemes"];
         }
         else
         {
            string themesDirPath =
               HttpContext.Current.Server.MapPath("~/App_Themes");

            // získej pole podadresářů s motivy v adresáři /app_themes
            string[] themes = Directory.GetDirectories(themesDirPath);
            for (int i = 0; i <= themes.Length - 1; i++)
            themes[i] = Path.GetFileName(themes[i]);

            // ulož pole do vyrovnávací paměti se závislostí na
            // dané cestě k adresáři
            CacheDependency dep = new CacheDependency(themesDirPath);
            HttpContext.Current.Cache.Insert("SiteThemes", themes, dep);

            return themes;
         }
      }
   }
}
```

Nyní, když máme náš řídicí prvek hotov, vrátíme se zpět na vzorovou stránku a v okně SOURCE VIEW přidáme úplně nahoru řádek, který na něj bude odkazovat:

```
<%@ Register Src="Controls/ThemeSelector.ascx" TagName="ThemeSelector"
   TagPrefix="mb" %>
```

Poté stačí jen deklarovat jeho instanci v místě, kde jej chceme mít – v našem případě tedy uvnitř kontejneru `themeselector`:

```
<div id="themeselector">
   <mb:ThemeSelector id="ThemeSelector1" runat="server" />
</div>
```

Kód, který se stará o přepnutí na nový motiv, nemůžeme umístit do události `SelectedIndexChanged` řídicího prvku typu `DropDownList`, protože to už bychom se v životním cyklu stránky nacházeli příliš daleko. Jak už jsme si řekli v části „Návrh“, nový motiv musíme aplikovat v obsluze události stránky `PreInit`. Dále místo vložení kódu do každé stránky jej stačí napsat jen jednou do naší bázové třídy. Zde pak přečteme hodnotu zvoleného indexu řídicího prvku typu `DropDownList`, kterou poté použijeme pro aplikaci nového motiv. Ovšem k řídicím prvkům a jejich hodnotám nemůžeme přistupovat přímo z obsluhy události `PreInit`, poněvadž se nacházíme v poměrně rané fázi životního cyklu stránky. Hodnotu našeho řídicího prvku tudíž musíme přečíst v rámci serverové události, ke které dojde později: nejvhodnějším místem se tak jeví událost `Load`. Nicméně v momentě, kdy se nacházíme v obsluze události `Load`, neznáme ID konkrétního řídicího prvku typu `DropDownList`. Musíme tedy najít způsob, jak jej identifikovat, načež můžeme číst jeho hodnotu ze surových dat zaslaných zpět na server přes kolekci `Request.Form`. Stále nám však zůstal ještě jeden problém: k získání hodnoty z kolekce musíme znát ID našeho řídicího prvku na straně klienta, které se však může měnit dle kontejneru, do něhož jej umístíme. Není tedy nejvhodnější zapsat jej do kódu napevno, protože později by mohlo dojít k jeho změně. Místo toho můžeme při jeho prvním vytvoření uložit ID do statického atributu třídy, takže bude k dispozici po celou dobu běhu webové aplikace, tedy i mezi jednotlivými požadavky (typu postback), dokud nebude aplikace uzavřena (přesněji dokud nedojde k uvolnění prvků sestavení z domény aplikace). Do adresáře `App_Code` tedy přidáme soubor `Globals.cs`, do kterého umístíme následující kód:

```
namespace MB.TheBeerHouse
{
   public static class Globals
   {
      public static string ThemesSelectorID = "";
   }
}
```

Poté se vrátíme zpět k doprovodnému kódu řídicího prvku `ThemeSelector` a přidáme kód pro uložení jeho ID do statického atributu třídy:

```
public partial class ThemeSelector : System.Web.UI.UserControl
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if (Globals.ThemesSelectorID.Length == 0)
         Globals.ThemesSelectorID = ddlThemes.UniqueID;

      ddlThemes.DataSource = Helpers.GetThemes();
```

```
      ddlThemes.DataBind();

      ddlThemes.SelectedValue = this.Page.Theme;
   }
}
```

Nyní můžeme pro své stránky vytvořit vlastní bázovou třídu, která je v podstatě jen další obvyklou třídou umístěnou v adresáři `App_Code` a která je odvozena od třídy `System.Web.UI.Page`. Přetížíme její metodu `OnPreInit`, v níž budeme provádět následující činnosti:

1. Ověříme, zdali je aktuální požadavek typu postback. Pokud ano, tak zkontrolujeme, byl-li způsoben naším roletovým menu typu `ThemeSelector`. Stejně jako v prostředí ASP.NET 1.x, všechny stránky s formulářem na straně serveru mají skryté pole `__EVENTTARGET`, které se nastaví na ID řídicího prvku jazyka HTML, který zapříčinil daný požadavek (nejde-li o tlačítko typu Submit). Pro ověření tohoto stavu stačí na základě ID načteného z třídy `Global` zkontrolovat, obsahuje-li element `__EVENTTARGET` z kolekce Form ID našeho roletového menu.
2. Jsou-li všechny podmínky bodu 1 splněny, obdržíme jméno vybraného motivu z elementu kolekce `Form`, jehož ID je shodné s hodnotou ID uloženou ve třídě `Globals`. Takto získané jméno pak použijeme pro nastavení vlastnosti `Theme` a uložíme jej do proměnné `Session`, aby i následně načítané stránky měly správný motiv a nevracely se k původnímu, výchozímu motivu.
3. Pokud aktuální požadavek není typu postback, zkontrolujeme, zdali je proměnná `Session`, kterou jsme použili v bodě 2, prázdná (null) či nikoliv. Pokud není, pak její hodnotu použijeme pro nastavení vlastnosti `Theme`.

Celý výše uvedený postup realizuje následující úryvek kódu:

```
namespace MB.TheBeerHouse.UI
{
   public class BasePage : System.Web.UI.Page
   {
      protected override void OnPreInit(EventArgs e)
      {
         string id = Globals.ThemesSelectorID;
         if (id.Length > 0)
         {
            // jedná-li se o postback způsobený roletovým menu
            // pro výběr motivu, vezmi zvolený motiv a použij jej pro
            // požadavek aktuální stránky
            if (this.Request.Form["__EVENTTARGET"] == id &&
               !string.IsNullOrEmpty(this.Request.Form[id]))
            {
               this.Theme = this.Request.Form[id];
               this.Session["CurrentTheme"] = this.Theme;
            }
            else
            {
               // nejde-li o postback nebo jde o postback způsobený
               // jiným řídicím prvkem, nastav motiv stránky dle hodnoty
```

```
            // v proměnné Session, obsahuje-li nějakou
            if (this.Session["CurrentTheme"] != null)
               this.Theme = this.Session["CurrentTheme"].ToString();
         }
      }

      base.OnPreInit(e);
   }
  }
}
```

Nevýhoda tohoto přístupu spočívá v tom, že zvolený motiv ukládáme do proměnné Session, která je po ukončení sezení smazána – konkrétně tedy ve chvíli, kdy uživatel zavře prohlížeč nebo kdy po dobu 20 minut (dobu lze upravit) neprovede na stránce žádný požadavek. Mnohem lepším řešením by bylo použít vlastnost Profile, která kromě jiných výhod uchovává hodnoty trvale i mezi jednotlivými sezeními. Více si o tomto novém prvku technologie ASP.NET 2.0 řekneme v kapitole 4 (kde zároveň upravíme kód tak, aby používal tuto novou vlastnost).

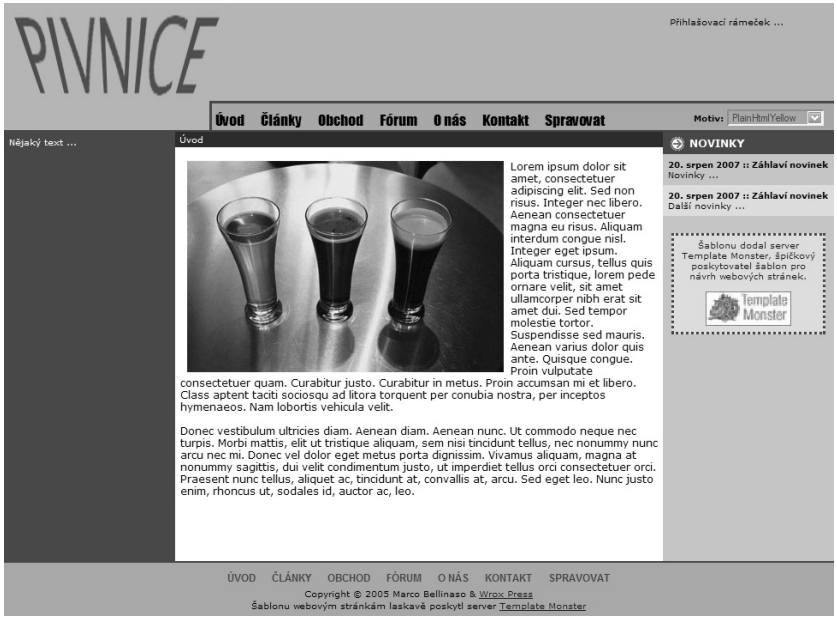
Poslední věcí, kterou musíme udělat, je úprava třídy v doprovodném kódu stránky `Default.aspx`, aby místo výchozí třídy `Page` používala naši třídu `BasePage`, z níž budeme volat metody původní třídy `Page`. Stačí jen změnit jediné slovo (tedy `Page` na `BasePage`):

```
public partial class _Default : MB.TheBeerHouse.UI.BasePage
{
   protected void Page_Load(object sender, EventArgs e)
   { }
}
```

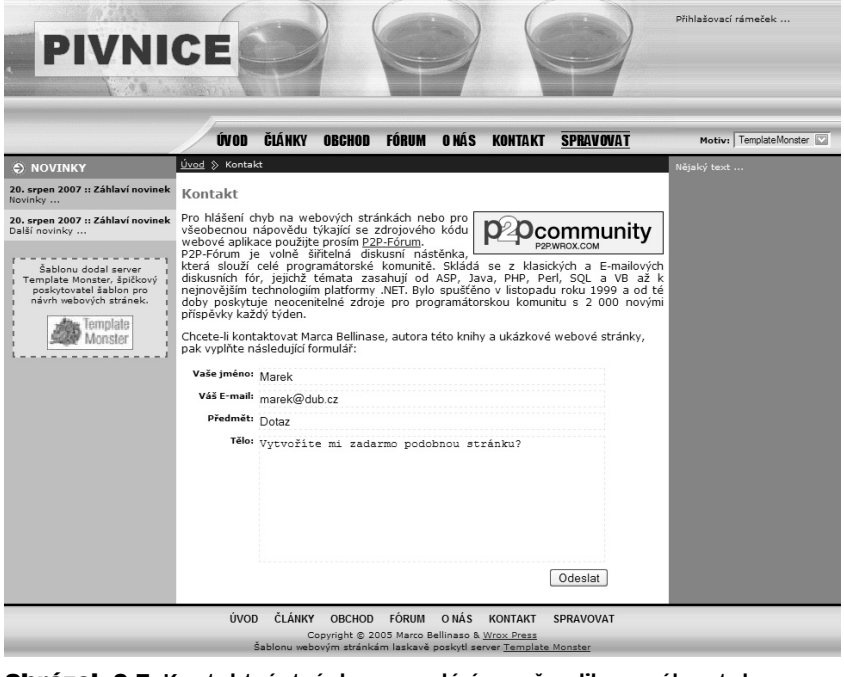
A jsme hotovi! Spustíme-li nyní náš projekt, spatříme úvodní stránku podobně jako na obrázku 2.4 (až na přihlašovací rámeček, který zatím nic neobsahuje – naplníme jej až v kapitole 5), na kterou je aplikován motiv `TemplateMonster`. Pokud v roletovém menu `ThemeSelector` zvolíme položku `PlainHtmlYellow`, měla by se úvodní stránka změnit podobně, jako je tomu na obrázku 2.6.

Další malá úprava stylu

Jediná stránka (`Default.aspx`) nestačí pro otestování všeho, co jsme v této kapitole probrali a implementovali. Například jsme neviděli řídicí prvek typu `SiteMapPath` v akci, protože nám neukáže žádný odkaz, dokud se samozřejmě nepřesuneme pryč z úvodní stránky. Pro jeho otestování můžeme snadno implementovat stránky `Contact.aspx` a `About.aspx`. V této kapitole se zaměříme na stránku `Contact.aspx`, která nám poslouží jako ukázkový příklad, protože chceme v motivu `TemplateMonster` trošku poupravit styl, abychom jej ještě více odlišili od motivu `PlainHtmlYellow`. Výslednou stránku zachycuje obrázek 2.7.



Obrázek 2.6: Naše webová stránka po přepnutí na motiv PlainHTMLYellow



Obrázek 2.7: Kontaktní stránka po malé úpravě aplikovaného stylu

Kódem, který řídí stránku a odesílá e-maily, se v této kapitole zabývat nebudeme, vrátíme se k němu až v kapitole následující. Nebudeme se věnovat ani vlastnímu obsahu souboru `.aspx`, protože obsahuje pouze řídicí prvek typu `Content`, ve kterém je jen pár odstavců textu a nějaké řídicí prvky typu TextBox s vlastními validátory. Místo toho se zaměříme na fakt, že pozadí textového rámečku Subject má žlutou barvu, což je dáno tím, že jde o vstupní pole s kurzorem. Zvýraznění aktivního řídicího prvku může usnadnit uživatelům rychlou orientaci, která nemusí být vždy jednoduchá, zvláště pokud pomocí tabulátoru procházejí řídicí prvky zobrazené v několika sloupcích a řádcích. Realizace tohoto efektu je docela jednoduchá: stačí obsloužit události vstupních polí na straně klienta onfocus resp. onblur a aplikovat resp. odebrat CSS-styl nastavením atributu `className`. Třída definující styl v našem příkladu nastaví barvu pozadí na žlutou a barvu textu na modrou. Přidáme ji do souboru `Default.css` v podadresáři `TemplateMonster`:

```
.highlight
{
   background-color: #fefbd2;
   color: #000080;
}
```

Události `onfocus` a `onblur` zpracujeme pomocí JavaScriptu na straně klienta. Stačí ke kolekci `Attribute` řídicího prvku přidat několik párů ve tvaru `jméno_atributu / hodnota`, takže se za běhu automaticky vygenerují v původním tvaru vedle ostatních atributů generovaných řídicím prvkem. Do třídy `Helpers` vytvořené výše můžeme přidat novou statickou metodu, do níž umístíme všechen potřebný kód, čímž si usnadníme jeho používání. Nová metoda `SetInputControlsHighlight` má následující parametry: odkaz na řídicí prvek, název třídy definující styl, který se má aplikovat na aktivní řídicí prvek, a hodnotu typu `Boolean` indikující, má-li tato metoda mít vliv jen na textová pole nebo také na řídicí prvky typu `DropDown`, `List`, `ListBox`, `RadioButton`, `CheckBox`, `RadioButtonList` a `CheckBoxList`. Je-li předaný řídicí prvek správného typu, přidá k němu metoda atributy `onfocus` a `onblur`. V opačném případě se volá rekurzivně pro všechny potomky, má-li nějaké. Tímto způsobem můžeme předat odkaz na objekt typu `Page` (který je sám řídicím prvkem, protože dědí z bázové třídy `System.Web.UI.Control`), Panel nebo nějaký jiný typ kontejnerového řídicího prvku a nechat si tak nepřímo zpracovat všechny jeho potomky. Následuje kompletní kód metody:

```
public static class Helpers
{
   public static string[] GetThemes() { ... }

   public static void SetInputControlsHighlight(Control container,
      string className, bool onlyTextBoxes)
   {
      foreach (Control ctl in container.Controls)
      {
         if ((onlyTextBoxes && ctl is TextBox) ||
            (!onlyTextBoxes && (ctl is TextBox || ctl is DropDownList ||
            ctl is ListBox || ctl is CheckBox || ctl is RadioButton ||
            ctl is RadioButtonList || ctl is CheckBoxList)))
         {
            WebControl wctl = ctl as WebControl;
```

```
            wctl.Attributes.Add("onfocus", string.Format(
              "this.className = '{0}';", className));
            wctl.Attributes.Add("onblur", "this.className = '';");
         }
         else
         {
            if (ctl.Controls.Count > 0)
               SetInputControlsHighlight(ctl, className, onlyTextBoxes);
         }
      }
   }
}
```

Ke spuštění uvedeného kódu v rámci události `Load` jakékoliv stránky musíme přetížit metodu `OnLoad` v již dříve vytvořené třídě `BasePage`:

```
namespace MB.TheBeerHouse.UI
{
   public class BasePage : System.Web.UI.Page
   {
      protected override void OnPreInit(EventArgs e) { ... }

      protected override void OnLoad(EventArgs e)
      {
         // přidej obsluhy událostí onfocus a onblur v javasciptu
         // všem vstupním řídicím prvkům,
         // aby se zvýraznily ty, které jsou aktivní
         Helpers.SetInputControlsHighlight(this, "highlight", false);
         base.OnLoad(e);
      }
   }
}
```

Tento kód se spustí vždy, bez ohledu na skutečnost, že soubor `Default.css` motivu `PlainHtmlYellow` neobsahuje třídu definující styl `highlight`. Aktivní řídicí prvek tohoto motivu tedy nebude mít žádný konkrétní styl.

Malé triky, jako je tento, lze implementovat snadno a rychle, mohou však výrazně zlepšit zážitek uživatele a udělat na něj pozitivní dojem. Navíc jednoduchost pramenící z existence vlastní bázové třídy pro obsahové stránky výrazně usnadňuje implementaci budoucích požadavků.

Shrnutí

V této kapitole jsme vybudovali základy pro vrstvu uživatelského rozhraní naší webové stránky. Navrhli jsme a implementovali vzorovou stránku se sdíleným HTML-kódem a společnými grafickými prvky, čímž jsme redukovali případnou změnu uspořádání a grafiky webu na úpravu jediného souboru. Pomocí motivů (další nový prvek zavedený společně se vzorovými stránkami od verze ASP.NET 2.0) jsme pro tutéž vzorovou stránku vytvořili pár odlišných vzhledů. Dá-

le jsme vyvinuli mechanismus umožňující uživatelům výběr vlastního oblíbeného motivu z roletového menu, aby tak mohli měnit vzhled našeho webu dle vlastní chuti. Prostřednictvím nového souboru `Web.sitemap` a řídicích prvků typu `Menu` a `SiteMapPath` jsme implementovali flexibilní a snadno udržovatelný navigační systém, který je v prostředí ASP.NET 2.0 také novinkou. Nakonec jsme použili vlastní třídu `BasePage`, abychom kromě společného vzhledu realizovali také společné chování stránek. Podtrženo sečteno, s poměrně malým počtem řádků kódu se nám podařilo vyvinout nové a pro webovou aplikaci podstatné prvky. Pokud bychom se o něco takového pokoušeli s technologií ASP.NET 1.x nebo starší, stálo by nás to přinejmenším stovky a možná i více než tisíc řádků kódu. V další kapitole budeme i nadále probírat základní prvky, tentokráte však vrstvy aplikační.