

Část VI

Webové služby

KAPITOLA 32

Tvorba webových služeb

Už léta bojují softwaroví vývojáři a architekti o vytvoření softwarových komponent, které by mohly být vzdáleně zavolány prostřednictvím lokální sítě a Internetu. Během tohoto procesu bylo vytvořeno několik nových technologií a několik komplikovaných proprietárních řešení. Ačkoliv některé z těchto technologií byly docela úspěšné (pokud jde o spouštění back-end systémů v interních sítích), ani jedné z nich se nepodařilo zvládnout složité problémy spojené s Internetem, který představuje širokou, a někdy nespolehlivou síť počítačů, na nichž běží všechny možné typy hardware a operačních systémů.

Zde přichází na scénu webové služby XML (XML web services). Pokud chcete komunikovat s nějakou webovou službou, stačí poslat prostřednictvím HTTP příslušnou zprávu XML. Protože každé zařízení, které je určeno pro Internet podporuje HTTP protokol, a protože každý programovací jazyk umožňuje přistupovat ke XML parseru, existují určitá omezení ohledně typů aplikací, které mohou používat webové služby. V podstatě většina programovacích prostředí obsahuje sadu nástrojů vyšší úrovně, díky kterým je komunikace s webovou službou stejně snadná jako volání lokální funkce.

Tato kapitola přináší nejenom přehled webových služeb, ale také problémů, které tyto služby řeší. Pokud je tato problematika pro vás zcela nová, naučíte se, jak vytvářet a používat webové služby v ASP.NET. Zatím se však nebudete detailněji zabývat nižší úrovní – základními protokoly. Místo toho začnete webovou službu rovnou používat, přičemž v další kapitole se naučíte, jak ji rozšířit.

ZMĚNY TÝKAJÍCÍ SE WEBOVÝCH SLUŽEB V .NET 2.0

Pokud jste již programovali s webovými službami v .NET 1.x, pravděpodobně vás zajímá, co všechno se v změnilo v .NET 2.0. Z praktického hlediska je zde překvapivě málo nových věcí, výchozí infrastruktura je v podstatě úplně stejná. Najdete zde však spoustu užitečných zlepšení, z nichž mnoho se týká fungování webových služeb s komplexními typy (což jsou vlastní třídy dat, které se používají pro zasílání nebo získávání informací pro webovou metodu). Najdete zde popsány všechny tyto změny:

- **Podpora pro vlastnosti procedur.** Pokud máte webovou službu, která používá vlastní typ, .NET vytvoří kopii této třídy u klienta. V .NET 1.x je tato automaticky generovaná třída sestavena z veřejných vlastností. V .NET 2.0 se místo toho používají vlastnosti procedur. Tato malá změna sice neovlivní způsob fungování webové služby, nicméně vám umožní použít tuto třídu ve scénářích vázání dat.
- **Sdílení typů.** .NET nyní rozeznává, pokud dvě webové služby používají stejný komplexní typ a generuje pouze jednu třídu na straně klienta. Nejlepší na tom je, že díky tomu, že obě třídy používají stejné typy, můžete snadno z jedné webové služby získat nějaký objekt a přímo jej poslat k jiné webové službě.
- **Vlastní serializace.** Nyní můžete zapojit vaše vlastní třídy do procesu serializace tak, aby mohly kompletně řídit svoji vlastní reprezentaci v XML. Ačkoli jste serializaci mohli provádět i v ASP.NET 1.x, nebyla tato serializace zdokumentována a dostatečně podporována.
- **Bohaté objekty.** Potřebujete si vyměňovat komplexní objekty společně s metodami a konstruktory? Nyní je to možné, pokud vytvoříte novou komponentu nazývanou jako importér schématu (schema importer). Importér schématu ověřuje schéma webové služby a sděluje třídě, jaké typy by měla použít.
- **Contract-first vývoj.** Nyní si můžete vybudovat webovou službu .NET, která bude srovnatelná s již existujícím WSDL.

Všechna tato zlepšení podrobně popisuje kapitola 33.

Mnoho dramatických změn je ovšem stále ještě před vámi. Vývojáři Microsoftu dokončují Indigo, což je nový model pro distribuci zpráv, který obsahuje funkcionalitu webové služby a další technologie .NET, jako je například vzdálený přístup. Ačkoli bude určité poskytnut přirozený způsob, jak přejít od webové služby ASP.NET k modelu Indigo, mnoho detailů implementace bude změněno. Indigo není součástí .NET 2.0, nýbrž by mělo být dodáváno s další verzí Windows (která nebude k dispozici dříve než koncem roku 2006). Microsoft však naznačil, že ještě předtím by mohl vydat sadu nástrojů Indigo určenou pro jiné verze Windows. Další informace získáte ve vývojářském centru Microsoft Indigo na adrese <http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo>.

Přehled webových služeb

Zatímco HTML stránky (nebo HTML výstup vygenerovaný webovými formuláři ASP.NET) by měly být používány koncovými uživateli, webové služby jsou používány jinými aplikacemi. Jsou to části obchodní logiky, ke kterým se lze dostat přes Internet. Například stránky internetového obchodu můžou využívat webovou službu nějaké poštovní společnosti pro výpočet ceny za dopravu zásilky. Stránka se zpravodajstvím může získávat nadpisy a články od externích dodavatelů zpráv a zobrazovat je v reálném čase na svých vlastních stránkách. Web může dokonce poskytovat aktuální hodnoty akcií, které jsou získávány ze specializovaných

finančních nebo investičních stránek. To ovšem není žádná vzdálená budoucnost. Všechny tyto scénáře na webu už rutinně fungují a největší internetové společnosti, jako jsou Amazon, Google a eBay nabízejí vývojářům k použití své vlastní webové služby.

Pomocí webových služeb můžete prostřednictvím několika řádek kódu použít obchodní logiku někoho jiného, aniž byste ji museli vytvářet od úplného začátku. Tato technika je podobná technice, kterou používají programátoři v případě knihoven API, tříd a komponent. Hlavní rozdíl je v tom, že webové služby mohou být umístěny na vzdáleném serveru a zpravovány jinou společností.

Historie webových služeb

Ačkoliv webové služby představují novou technologii, můžete hodně pochopit, pokud znáte jejich nedávnou historii. Dvě z největších změny v oblasti vývoje softwaru, které proběhly během několika posledních dekád, se týkaly vývoje objektově orientovaného programování a technologie založené na komponentách.

Objektově orientované programování se připojilo k hlavnímu programovacímu proudu někdy v osmdesátých letech minulého století. Mnozí lidé v objektově orientovaném programování viděli řešení softwarové krize, která vyplývala z narůstající složitosti a velikosti softwarových aplikací. Mnoho projektů se opozdilo a přeskročilo svůj rozpočet, přičemž výsledný produkt byl často nespolehlivý. Objektově orientovaný kód sliboval začlenění objektů do kódu, díky čemuž mohli vývojáři vytvářet komponenty, které byly opakovaně použitelné, rozšiřitelné a snadno udržitelné.

V devadesátých letech minulého století se objevila technologie komponent, která umožnila budovat aplikace sestavováním komponent. Technologie založená na komponentech je skutečným rozšířením objektově orientovaných principů za hranice libovolného jazyka, takže se stává jádrem infrastruktury, které může kdokoli použít. Zatímco objektově orientovaný jazyk umožňoval vývojářům opakovaně používat objekty ve svých aplikacích, technologie založená na komponentách jim umožňovala snadno sdílet zkompileované objekty mezi aplikacemi. Vznikly dvě dominantní technologie založené na komponentách – COM (Component Object Model) a CORBA (Common Object Request Broker Architecture). Od té doby se objevily další technologie založené na komponentách (jako například JavaBeans a .NET), nicméně jsou navrženy jako proprietární řešení pro konkrétní programovací prostředí.

Krátce poté, co byly COM a CORBA vytvořeny, byly tyto standardy použity u distribuovaných komponent tak, aby aplikace mohla pracovat s objekty umístěnými v různých počítačích zapojených do sítě. Ačkoli COM i CORBA jsou po technické stránce velice sofistikované, je často velmi obtížné je nastavit a podporovat v prostředí lokální sítě, nehledě na fakt, že nemohou fungovat společně. S objevem Internetu se pak logicky objevil dramatický nárůst dalších problémů. I přesto všechno je vývojáři začali používat při vytváření distribuovaných aplikací, tzn. WAN aplikací (Wide Area Networks), které se ovšem velmi pomalu rozšiřovaly, a které také byly méně spolehlivé. Protože technologie COM i CORBA jsou velice komplexní a současně proprietární, ani jedna z nich se v tomto prostředí neprosadila natolik, aby se to dalo považovat za úspěch.

Webové služby jsou novou technologií, která má za cíl vyřešit tyto problémy rozšířením technologie objektů a komponent do webového prostředí. Webová služba je v zásadě jednotkou aplikační logiky (komponentou), která může být vzdáleně zavolána přes Internet. Očekávání vkládaná do webových služeb se v mnohém shodují s očekáváními vkládanými do technologie komponent, přičemž jejich cílem je usnadnit sestavování aplikací pomocí hotové aplikační logiky, sdílet funkcionalitu mezi partnery a vytvářet mnohem modulárnější aplikace. Na rozdíl od dřívějších technologií založených na komponentách jsou webové služby navrženy výhradně pro tento účel. To znamená, že jako vývojáři v .NET budete stále používat .NET model komponent, pomocí kterého budete schopni sdílet zkompileované assembly mezi jednotlivými aplikacemi .NET. Pokud

však budete chtít sdílet funkcionalitu mezi aplikacemi, které běží na různých platformách, a které jsou poskytovány různými společnostmi, budou pro vás webové služby naprosto ideálním řešením.

Webové služby také kladou mnohem větší důraz na součinnost (interoperability). Všechny platformy pro vývoj softwaru, které umožňují programátorům vytvářet webové služby, používají jako základní kámen otevřené standardy, které jsou založeny na XML. Tím je zajištěno, že pomocí .NET můžete vytvořit nějakou webovou službu a následně ji zavolat z nějakého klienta založeného na Javě, nebo vytvořit webovou službu v Javě, a tu následně zavolat z .NET aplikace.

POZNÁMKA Webové služby nejsou omezeny pouze na .NET Framework. Standardy webových služeb byly specifikovány ještě před uvedením platformy .NET, a jsou používány a podporovány i dalšími výrobci softwaru. .NET Framework je ovšem speciálním případem, protože v sobě ukrývá veškerý propojovací kód, což vám usnadní nejenom zprovoznění vašich webových služeb přes internet, ale také vám to usnadní přístup k webovým službám, které poskytují jiné společnosti. Jak dále uvidíte, nemusíte znát XML a SOAP do naprostých detailů, abyste mohli úspěšně budovat webové služby, ačkoliv je pravda, že určitá úroveň znalostí vám rozhodně neuškodí. ASP.NET provádí abstrakci podrobnosti a vytváří obalové třídy, které vystavují jednoduchý, objektově orientovaný, model takovým způsobem, aby bylo snadné posílat, získávat a interpretovat zprávy SOAP.

Distribuované výpočty a webové služby

Abyste zcela pochopili důležitost webových služeb, musíte porozumět požadavkům distribuovaných výpočtů. Distribuované výpočty představují rozdělení aplikační logiky do jednotek, které jsou vykonávány na dvou nebo více počítačích v síti. Myšlenka distribuovaných výpočtů se zrodila už dříve, přičemž pro distribuované a současně opakované použití aplikační logiky bylo vyvinuto mnoho komunikačních technologií. Použití distribuované aplikační logiky má mnoho výhod. Zde jsou uvedeny pouze nejdůležitější výhody takového přístupu:

- **Vysoká rozšiřitelnost.** Prostřednictvím distribuované aplikační logiky je zátěž rozložena na více počítačů. To sice nezlepší výkon dané aplikace u jednotlivých uživatelů (v podstatě jej může o něco snížit), nicméně to prakticky vždy zvýší rozšiřitelnost – tím se dosáhne toho, aby aplikace současně mohla sloužit podstatně většímu počtu uživatelů.
- **Snadné rozmístění.** Části distribuované aplikace mohou být upgradovány, aniž by bylo potřeba upgradovat celou aplikaci. Centrálně umístěná komponenta může být aktualizována, aniž by bylo nutné aktualizovat stovky (nebo dokonce tisíce) klientů.
- **Vylepšená bezpečnost.** Distribuované aplikace často přesahují hranice společnosti či organizace. Můžete například používat distribuované komponenty, které vašemu obchodnímu partnerovi umožní získat katalog produktů vaší společnosti. Nebylo by ovšem bezpečné nechat vašeho obchodního partnera, aby se přímo napojil do databáze vaší společnosti. Místo toho musí obchodní partner použít komponentu, která běží na vašich serverech, a kterou máte pod vlastní kontrolou, a kterou můžete v případě potřeby odpovídajícím způsobem omezit.

Internet zvýšil důležitost a použitelnost distribuovaných výpočtů. Jednoduchost a všudypřítomnost Internetu z něj dělá logickou volbu při výběru základního kamene distribuovaných aplikací.

Před objevením webových služeb byly dominantní protokoly COM (při použití v síti nazývaný jako DCOM – Distributed COM) a CORBA. Ačkoli CORBA a DCOM mají mnoho společného, liší se v několika detailech, z čehož vyplývá, že je velmi obtížné dosáhnout vzájemné spolupráce těchto dvou protokolů. Tabulka 32-1

uvádí přehled podobností a rozdílů mezi CORBA, DCOM a webovou službou. V tabulce také najdete velké množství zkratk.

Tabulka 32-1. Srovnání jednotlivých distribuovaných technologií.

Charakteristika	CORBA	DCOM	Webová služba
Mechanismus RPC (vzdálené volání procedury)	IIOB (Internet Inter-ORB Protocol)	DCE-RPC (Distributed Computing Environment Remote Procedure Call)	HTTP (Hypertext Transfer Protocol)
Kódování	CDR (Common Data Representation)	NDR (Network Data Representation)	XML (Extensible Markup Language)
Popis rozhraní	IDL (Interface Definition Language)	IDL (Interface Definition Language)	WSDL (Web Service Description Language)
Zpřístupnění	Služba pojmenování (Naming service) a obchodování (trading service)	Registr	UDDI (Universal Description, Discovery, and Integration)
Prívětivost k fire-wallu?	Ne	Ne	Ano
Složitost protokolů	Vysoká	Vysoká	Nízká
Použití mezi platformami?	Částečně	Ne	Ano

Protokoly CORBA i COM umožňují vyvolat vzdálené objekty. CORBA používá standard pojmenovaný jako IIOB (Internet Inter-ORB Protocol), DCOM používá variantu standardu s názvem DCERPC (Distributed Computing Environment Remote Procedure Call). Kódování dat v CORBA je založeno na formátu s názvem CDR (Common Data Representation). V DCOM je kódování dat založeno na podobném, ale nekompatibilním formátu s názvem NDR (Network Data Representation). Tyto vrstvy standardů významně zvyšují složitost těchto protokolů.

Existují také rozdíly mezi jazyky, které oba protokoly podporují. Protokol DCOM podporuje širokou škálu jazyků (C++, Visual Basic atd.), nicméně byl hlavně používán v operačních systémech Microsoftu. Protokol CORBA také podporoval různé platformy, byl však spojen především s aplikacemi založenými na Javě. Výsledkem bylo, že vývojáři měli k dispozici dvě platformy, které byly technicky schopny podporovat systémy distribuovaných objektů, nicméně však nemohly spolupracovat společně.

Problémy spojené s technologiemi distribuovaných komponent

Součinnost představuje pouze část problémů spojených s protokoly CORBA a DCOM. Tyto protokoly se potýkají také s dalšími technickými potížemi – oba byly vyvinuty dříve než Internet a jako takové nejsou navrženy tak, aby počítaly s volně propojenou, a někdy nespolehlivou a velice zatíženou sítí. Oba tyto protokoly jsou orientované na spojení. To znamená, že DCOM klient udržuje spojení s DCOM serverem, aby mohl realizovat vícenásobná volání. DCOM komponenta na straně serveru si může v paměti uchovávat informace o klientovi. Tím vzniká bohatý a flexibilní programovací model, nicméně to však není vhodný způsob pro navrhování rozsáhlých aplikací, které používají bezstavové protokoly Internetu. Pokud klient jednoduše

zmizí, aniž by řádně ukončil spojení, je zbytečně plýtváno důležitými zdroji. A podobně v případě, kdy se na jednu pokouší připojit tisíce uživatelů, je server snadno zahlcen a rychle vyčerpá svou paměť nebo kapacitu spojení.

Další problém je vtom, že oba protokoly jsou mimořádně komplexní. Kombinují v sobě technologii distribuovaného objektu s bezpečnostními funkcemi sítě a řízením životního cyklu. Používání webových služeb je ovšem mnohem jednodušší, protože nejsou natolik sofistikované. Neznamená to však, že nemůžete vytvořit bezpečnou webovou službu. Při vytváření takové bezpečné webové služby se však musíte opřít o nějaký další standard, například o SSL (implementované webovým serverem) nebo WSSecurity a XML Encryption (které jsou implementovány programovacím rámcem).

POZNÁMKA Existuje zde nebezpečí, že vývojáři budou zaplaveni vznikajícími standardy, které nejsou potřebné pro základní webové služby, nýbrž pro sofistikované aplikace webové služby. Tento model však ještě pořád představuje nejlepší kompromis mezi složitostí a jednoduchostí. Výhodou je, že architekti mohou vyvíjet inovace webových služeb (jako je například podpora transakcí), aniž by byli nuceni přistupovat ke kompromisům na základní úrovni součinnosti, kterou poskytují základní standardy webových služeb.

Výhody webových služeb

Webové služby jsou zajímavé z několika hledisek. Z technologického hlediska se webové služby snaží vyřešit problémy těsně spojených technologií, jako jsou například CORBA a DCOM. Jsou to problémy jako je například průchod přes firewally, zpracování složitých transportních protokolů nižší úrovně a integrace různorodých platform. Webové služby jsou také zajímavé z organizačního a ekonomického hlediska, protože otvírají dveře novým způsobům obchodování a integraci systémů mezi jednotlivými organizacemi.

DCOM a CORBA jsou vhodné pro budování podnikových aplikací, kde software běží na stejné platformě a podobně spravované lokální síti. Nejsou však vhodné pro vytváření aplikací, které propojují jednotlivé platformy, komunikují přes Internet, a které potřebují Internet pro dosažení větší rozšiřitelnosti. Pro tyto účely jednoduše nejsou určeny.

Zde tedy přicházejí webové služby, které představují další logický krok ve vývoji distribuovaných technologií založených na komponentách. Jejich největší výhody jsou následující:

- **Webové služby jsou jednoduché.** Jejich jednoduchost spočívá v tom, že mohou být snadno podporovány širokou škálou platform.
- **Webové služby jsou volně spojeny.** Webová služba může rozšířit svoje rozhraní a přidat nové metody, aniž by to jakkoliv ovlivnilo klienty (pokud ovšem webová služba poskytuje staré metody a parametry).
- **Webové služby jsou bezstavové.** Klient vznese vůči webové službě požadavek, ta mu vrátí výsledek a spojení je ukončeno. Neexistuje permanentní spojení. To umožňuje se snadno přizpůsobit mnoha klientům a k poskytování webových služeb využívat serverovou farmu. Výchozí HTTP protokol, který je používán webovými službami, je rovněž bezstavový. Je samozřejmě možné poskytnout nějaký stav pomocí dodatečných technik, jako jsou například techniky používané u webových stránek ASP. NET včetně cookies. Tyto techniky však nejsou obecně standardizovány.
- **Webové služby jsou přívětivé k firewallu.** Firewally mohou pro technologie distribuovaných objektů představovat problém. Jediné, co přes firewally prakticky vždy projde, je HTTP provoz na portech

80 a 443. Protože webové služby zcela standardně používají HTTP protokol, mohou bez problémů procházet přes firewall, aniž by k tomu potřebovaly změnit nějaké nastavení firewallu.

POZNÁMKA Firewall je ovšem možné použít k blokování přenosu zpráv SOAP. Je to možné díky tomu, že HTTP záhlaví zprávy nějaké webové služby rozpoznává zprávy SOAP a administrátor může firewall nakonfigurovat tak, aby zablokoval přenos těchto zpráv. Pro konkrétní scénáře typu "obchodník-s-obchodníkem" je možné firewall nastavit tak, aby byl povolen přenos zpráv SOAP pouze z vybraných IP adres.

Je samozřejmé, že jednoduchost webových služeb si vybírá svou daň. Webové služby nemají všechny funkce mnohem komplexnějších technologií distribuovaných komponent. Nenačtete zde například podporu oboustranné komunikace, což značí, že webový server nemůže zpětně zavolat klienta poté, co se klient odpojil. V tomto ohledu jsou těsně spojené protokoly jako například DCOM a CORBA mnohem výkonnější než webové služby. .NET Framework ovšem nabízí novou technologii nazvanou jako remoting (vzdálené volání), která je ideální pro komunikaci mezi distribuovanými .NET aplikacemi a interní sítí. Remoting je následovníkem DCOM na platformě .NET.

KDY POUŽÍT WEBOVÉ SLUŽBY

Microsoft doporučuje se zamyslet nad následujícími příklady, pokud z nějakého důvodu nevíte, zdali je vhodné použít webové služby. Potřebujete-li překročit hranice jedné platformy (například při komunikaci mezi aplikacemi v .NET a Javě) nebo pokud se naopak máte na tyto hranice spoléhat (například při komunikaci mezi dvěma společnostmi), má použití webových služeb velký význam. Webové služby jsou rovněž dobrou volbou, potřebujete-li použít předem vybudované funkce ASP.NET, jako je například cachování nebo různé funkce IIS (zajištění bezpečnosti prostřednictvím SSL či autentizace Windows atd.). Webové služby mají smysl i tehdy, chcete-li nechat svoji vlastní aplikaci otevřenou pro případnou budoucí integraci funkcí, které vymyslel a vytvořil někdo jiný.

Pokud chcete sdílet funkcionalitu mezi dvěma aplikacemi .NET, webové služby mohou být neúměrně velké a mohou zapříčinit zbytečné přetěžování zdrojů. Chcete-li například docílit toho, aby webové aplikace měly přístup ke specifické obchodní logice, je mnohem lepší vytvořit nějakou assembly (ve formě zkompilovaného souboru DLL) a použít ji v obou aplikacích. Tímto se vyhnete přetěžování neprocesové nebo síťové komunikace, která může velmi významná.

A konečně – pokud chcete distribuovat nějakou vaši funkcionalitu tak, aby se k ní dalo vzdáleně přistupovat, a pokud klient i server používají .NET Framework, možná byste raději měli zvážit použití technologie vzdáleného volání .NET (.NET remoting). Vzdálené volání vám sice neposkytuje stejnou úroveň podpory otevřených standardů jako třeba SOAP, dává vám ovšem svobodu použít různé typy komunikace, proprietární datové typy .NET, stavové objekty a rychlejší komunikaci založenou na TCP/IP. Stručně řečeno – vzdálené volání vám nabízí více funkcí a možnost zvýšit výkon řešení založených na .NET.

Vydělávání peněz s webovými službami

Nová technologie je ztracena, neposkytuje-li nové příležitosti lidem, kteří se zabývají vyděláváním peněz. Webové služby přinášejí z pohledu obchodníka tyto nové možnosti:

- **Nové platební struktury.** Uživatel webové služby si může předplatit používání této služby. Příkladem může být dodávání zpráv z Associated Press. Další možnosti jsou platby za prohlížení (pou-

živá se také pojem mikroplatby). Příslušný poskytovatel takové služby může provádět účtování při každém požadavku na využití služby.

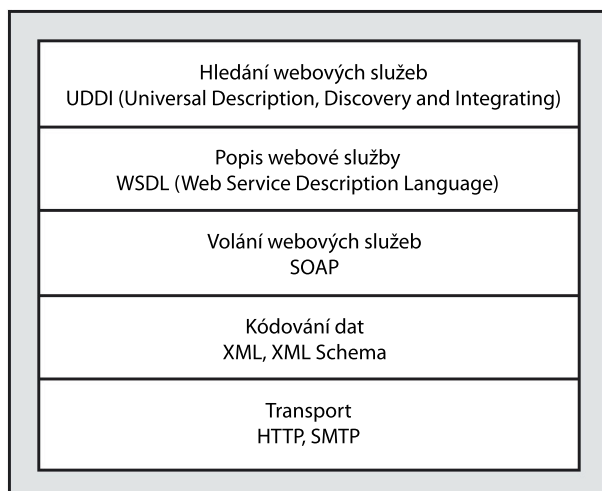
- **Interakci a spolupráci v reálném čase.** V dnešní době jsou data obvykle zkopírována a používána lokálně. Webové služby ovšem umožňují požádat v reálném čase o nějaká vzdálená data. Příkladem může být posílání počítačových her z internetového obchodu. Internetový obchod může být dále propojen se skladem a poskytovat aktuální počet položek na skladě. To umožňuje takovému internetovému obchodu poskytovat lepší služby. Určitě vás nevyvede z míry nic více než situace, kdy si něco objednáte přes Internet, zaplatíte, a na druhý den se dozvíte, že zboží, o které máte zájem, není momentálně skladem (nebo je rovnou vyprodáno).
- **Agregované služby.** Vaše webová služba se může přičlenit k jiným webovým službám či k různým odvozeným komponentám, které jsou vystaveny pomocí proprietárních protokolů atd. Typickým příkladem agregované služby je porovnávací služba, která vám poskytuje informace o nejlepších cenách nějakého zboží. Dalším příkladem může být služba, která obsahuje několik souvisejících služeb. Představte si například, že se stěhujete do nového domu. Někdo by vám mohl poskytnout službu, která aktualizuje vaši poštovní adresu, vypátrá stěhovací společnost, která přestěhuje váš majetek, zobrazí mapu vašeho nového okolí atd.

Webové služby v žádném případě nejsou jedinou technologií, která dokáže poskytovat tato řešení. Dnes existuje mnoho různých řešení, které jsou založeny na použití existujících technologií. Webové služby ovšem používají všeobecně respektované standardy, čímž mají potřebnou sílu k tomu, aby se všechny tyto typy služeb mohly stát široce dostupnými.

Standardy používané webovými službami

Klíčem k úspěchu webových služeb je to, že jsou založeny na otevřených standardech, a že za těmito standardy stojí velké společnosti, jako je Microsoft, IBM a Sun. Otevřené standardy však automaticky nevedou ke spolupráci. Společnosti nejprve musí všechny tyto standardy implementovat, a to kompatibilním způsobem.

Při budování webových služeb se používá několik specifikací. Na obrázku 32-1 vidíte standardy, které v současnosti používají webové služby.



Obrázek 32-1. Standardy, které dnes používají webové služby.

O protokolech SOAP, WSDL a UDDI, které podporují webové služby, se dozvíte mnohem více v další kapitole. Než však začnete budovat a používat webové služby, je důležité alespoň rámcově pochopit úlohu, kterou tyto standardy sehrávají. Tabulka 32-2 uvádí stručný přehled těchto standardů, přičemž v několika dalších sekcích této kapitoly si je popíšeme o něco podrobněji. Podstatně podrobněji se jimi ovšem budeme zabývat až v další kapitole.

Tabulka 32-2. Standardy webových služeb.

Standard	Popis
WSDL	Používá se na vytvoření definice rozhraní webové služby. WSDL dokument oznámí klientovi, které metody jsou ve webové službě přítomny, které parametry a návratové hodnoty jednotlivé metody používají, a jakým způsobem s nimi má komunikovat.
SOAP	Formát zprávy, který je použit pro kódování informací (jako jsou například hodnoty dat) před jejich zasláním webové službě.
HTTP	Prostřednictvím tohoto protokolu probíhá veškerá komunikace webových služeb. SOAP zprávy jsou například zasílány přes HTTP kanály.
DISCO	Používá se k vytvoření tzv. discovery dokumentů, které poskytují odkazy na více koncových bodů webové služby. Tento standard je specifický pro Microsoft a posléze bude nahrazen podobným standardem pojmenovaným jako WS-Inspection.
UDDI	Standard pro vytváření obchodních rejstříků, které registrují společnosti a webové služby, které nabízejí, a také odpovídající URL adresy jejich WSDL kontaktů.

Hledání webových služeb

V jednoduchých aplikacích možná budete již předem znát URL webové služby, kterou chcete použít. V tom případě ji můžete zakódovat, nebo ji umístit do konfiguračního souboru. Není již potřeba provést žádné další kroky.

Za jiných okolností možná budete chtít v běhovém režimu hledat webovou službu, kterou potřebujete. Můžete například použít standardizovanou službu, která je poskytována různými společnostmi poskytujícími hosting a není vždy k dispozici na stejném URL. Nebo možná pouze budete potřebovat snadný způsob, jak najít všechny webové služby poskytované obchodním partnerem. V obou případech musíte použít discovery (zpřístupnění), pomocí něhož programově lokalizujete webové služby, které potřebujete.

Zpřístupnění webové služby vám usnadní dvě specifikace:

- **DISCO (což je zkratka ze slova discovery).** Standard DISCO vytváří jediný soubor, který seskupuje seznam příbuzných webových služeb. Společnost může na svém serveru zveřejnit soubor DISCO, který obsahuje odkazy na všechny poskytované webové služby. Klienti, kteří chtějí najít všechny dostupné webové služby, musí o tento soubor pouze požádat. Tento postup je užitečný, když klient už zná společnost, která webové služby nabízí a chce zjistit, které služby jsou zpřístupněny, a také najít odkazy k podrobnějším informacím o těchto službách. Vyhledávání webových služeb přes Internet není sice moc efektivní, nicméně se to může hodit u lokálních sítí, kde se klient připojí na server a hned vidí, které služby jsou k dispozici a kde.
- **UDDI (Universal Description, Discovery, and Integration).** UDDI je centralizovaný adresář, kam různé společnosti vkládají webové služby, které nabízejí klientům. Je to místo, kde může potenci-

onální klient vyhledávat služby podle svých specifických potřeb. Různé organizace a skupiny společností mohou používat různé UDDI rejstříky. Chcete-li z UDDI adresáře získat nějaké informace nebo do něj zaregistrovat své komponenty, použijete rozhraní webové služby.

Discovery je jednou z nejnovějších a nejméně zralých částí ze skupiny protokolů webových služeb. Standard DISCO je podporován pouze Microsoftem a je plánováno, že v dalších verzích .NET bude nahrazen podobným, ovšem mnohem obecnějším standardem s názvem WS-Inspection. Standard UDDI je navržen pro webové služby, které by měly být sdíleny buď veřejně, nebo mezi konsorciem společností či organizací. Není začleněn do .NET Frameworku, ačkoliv si můžete stáhnout samostatnou komponentu .NET pro vyhledávání UDDI adresářů a registraci vašich komponent (viz <http://msdn.microsoft.com/library/en-us/uddi/uddi/portal.asp>). Protože zatím nejsou k dispozici žádné zavedené UDDI adresáře, a protože mnoho webových služeb bylo navrženo pro použití v rámci jediné společnosti, nebo pouze mezi několika vzájemně spolupracujícími obchodními partnery, je pravděpodobné, že mnoho dnešních webových služeb nebude v UDDI adresářích vůbec publikováno.

Popis webové služby

Aby se klient dozvěděl, jak může přistoupit k webové službě, musí vědět, jaké metody jsou k dispozici, musí vědět, jaké parametry tyto metody používají, a musí také vědět, jaký datový typ tyto parametry používají. WSDL (Web Service Description Language) je jazyk založený na XML, který popisuje všechny tyto údaje. Popisuje zprávu o požadavku, kterou musí klient předložit webové službě a zprávu o odpovědi, kterou vrací webová služba. Definuje také přenosový protokol, který musíte použít (v typickém případě je to HTTP) a umístění webové služby.

WSDL je komplexní standard. Jak ale uvidíte dále v této kapitole, existují určité nástroje, které dokážou zpracovat WSDL informace, a které automaticky generují pomocné třídy, které skryjí propojení nižší úrovně, které je nutné pro interakci s webovými službami.

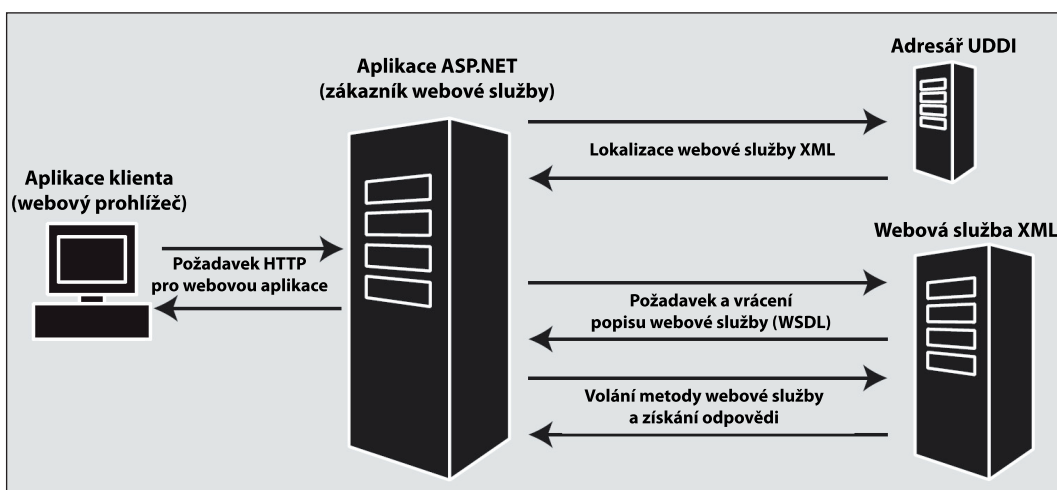
SOAP

Chcete-li komunikovat s webovou službou, musíte vytvořit takovou zprávu o požadavku a zprávu o odpovědi, které budou parsovány a pochopeny libovolnou platformou. SOAP (Simple Object Access Protocol) je jazyk založený na XML, který používáte pro vytvoření těchto zpráv.

Je důležité pochopit, že SOAP definuje zprávy, které používáte pro výměnu dat (tedy formát zpráv), avšak už nepopisuje, jakým způsobem zprávu zasíláte (přenosový protokol). U webových služeb ASP.NET se používá přenosový protokol HTTP. Řečeno jinými slovy – pokud klient chce komunikovat s webovou službou, otevře HTTP spojení a zašle SOAP zprávu.

.NET podporuje také HTTP GET a HTTP POST, což jsou dva jednodušší přístupy pro interakci s webovými službami, které ovšem nejsou standardizovány, a které nenabízejí stejně bohatou sadu funkcí. V obou těchto případech je pro komunikaci používán HTTP kanál, přičemž data jsou zasílána jako jednoduchá kolekce dvojic název/hodnota (a nikoliv jako plnohodnotná SOAP zpráva). Jediným místem, kde se můžete v prostředí .NET setkat s tímto přístupem, je jednoduchá, na prohlížeči založená, webová stránka, kterou ASP.NET poskytuje pro otestování vašich webových služeb. V ASP.NET verze 1.1 soubor machine.config standardně povoluje pouze HTTP požadavky POST z lokálního počítače a kompletně blokuje HTTP požadavky typu GET.

Obrázek 32-2 zobrazuje celý životní cyklus webové služby. Zákazník nejprve najde webovou službu, a to buď přímo – přes URL webové služby, nebo použije UDDI server či DISCO soubor.



Obrázek 32-2. Životní cyklus webové služby.

Dále klient získá WSDL dokument webové služby, který popisuje způsob interakce s webovou službou. Oba tyto úkoly probíhají v návrhovém režimu. Když aplikaci spustíte a dojde ke skutečné interakci s webovou službou, klient pošle SOAP zprávu, která spustí odpovídající webovou metodu.

Budování základní webové služby

V následujících sekcích se dozvíte, jak s ASP.NET vytvořit webovou službu a jak ji otestovat v prohlížeči. V následujícím příkladě vytvoříte webovou službu s názvem `EmployeesService`, která sečte a vrátí počet zaměstnanců v určitém městě (nebo součet všech zaměstnanců). V tomto příkladu bude použita tabulka `Employees` z ukázkové databáze Northwind SQL Serveru.

Třída webové služby

Webová služba začíná jako bezstavová třída s jednou nebo dvěma metodami. Právě tyto metody pak volá vzdálený klient pro vyvolání vašeho kódu.

Webová služba `EmployeeService` je navržena tak, aby umožnila vzdáleným klientům získávat informace o zaměstnancích společnosti. Poskytuje metodu `GetEmployees()`, která vrací sadu dat (`DataSet`) se všemi informacemi o zaměstnancích. Poskytuje rovněž metodu `GetEmployeesCount()`, která jednoduše vrací počet zaměstnanců z databáze. K poskytnutí těchto dvou metod nepotřebujete nic víc, než základní kód ADO.NET. Zde je kompletní výpis třídy:

```
public class EmployeesService
{
    private string connectionString;
    public EmployeesService()
    {
        string connectionString =
            WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
```

```

    }
    public int GetEmployeesCount()
    {
        SqlConnection con = new SqlConnection(connectionString);
        string sql = "SELECT COUNT(*) FROM Employees";
        SqlCommand cmd = new SqlCommand(sql, con);
        // Otevři spojení a získej hodnotu.
        con.Open();
        int numEmployees = -1;
        try
        {
            numEmployees = (int)cmd.ExecuteScalar();
        }
        finally
        {
            con.Close();
        }
        return numEmployees;
    }
    public DataSet GetEmployees()
    {
        // Vytvoř příkaz a spojení.
        string sql = "SELECT EmployeeID, LastName, FirstName, Title, " +
            "TitleOfCourtesy, HomePhone FROM Employees";
        SqlConnection con = new SqlConnection(connectionString);
        SqlDataAdapter da = new SqlDataAdapter(sql, con);
        DataSet ds = new DataSet();
        // Vyplň DataSet.
        da.Fill(ds, "Employees");
        return ds;
    }
}

```

Obě metody `GetEmployees()` a `GetEmployeesCount()` jsou veřejné, což značí, že můžete použít třídu z libovolné webové stránky stejného projektu. (Pokud tento soubor zkompilujete do samostatné DLL assembly, můžete ji použít v libovolném projektu, který obsahuje referenci na tuto assembly.) Než však bude tato třída připravena pro použití ve webové službě, potřebujete ještě provést několik kroků.

Požadavky na webovou službu

Než vaši třídu transformujete na webovou službu, musíte se ujistit, že je kompatibilní s požadavky výchozích standardů založených na XML, které používá. Protože webovou službu vykonává ASP.NET, jenž hostuje CLR, můžete použít jakýkoliv platný kód .NET. To znamená, že můžete použít třídy .NET, pro přístup k datům pomocí ADO.NET nebo použít regulární výrazy pro ověření dat.

Existují však určitá omezení ohledně informací, které může váš kód akceptovat (v podobě parametrů), a které může vracet (v podobě návratové hodnoty). Je to proto, že webové služby jsou vybudovány na standardech

pro výměnu dat, které jsou založeny na XML. Výsledkem je, že sada datových typů, kterou mohou webové služby používat, je omezena na sadu datových typů rozeznávaných standardem schémat XML. To znamená, že můžete používat jednoduché datové typy, jako jsou například řetězce a čísla, a že nemáte k dispozici žádný automatický způsob pro zasílání proprietárních objektů .NET, jako jsou FileStream, Image nebo EventLog. Toto omezení je ovšem velice smysluplné. Je totiž jasné, že jiné programovací jazyky nemají k dispozici žádný způsob pro interpretaci těchto komplexnějších tříd, takže i kdybyste náhodou vymysleli způsob posílání těchto komplexních tříd, klient by pravděpodobně nebyl schopen je interpretovat, což by narušilo celkovou součinnost. (Vzdálené volání .NET je příkladem technologie distribuovaných komponent, která vám umožní použít typy specifické pro .NET. Za toto pohodlí ovšem platíte tím, že nejsou podporováni klienti, kteří .NET nepoužívají.)

Tabulka 32-3 uvádí podporované datové typy webových služeb.

Tabulka 32-3. Datové typy webových služeb pro parametry a návratové hodnoty.

Datový typ	Popis
Základní	Jednoduché datové typy C# – celá čísla (short, int, long), celá čísla bez znaménka (ushort, uint, ulong), desetinná čísla (float, double, decimal) a několik dalších různých typů (bool, string, char, byte a DateTime).
Pole (Arrays)	Můžete používat pole jakéhokoliv podporovaného typu. Můžete rovněž použít i ArrayList (který je jednoduše zkonvertován na sadu), nemůžete však použít specializovanější kolekce, jako například Hashtable. Dále můžete použít binární data prostřednictvím bajtových polí. Binární data jsou automaticky kódována jako Base64, aby mohla být vložena do zprávy XML webové služby.
Vlastní objekty (Custom Objects)	Jakýkoliv objekt, který vytvoříte na základě vlastní třídy nebo struktury, můžete předat dále. Jediným omezením je to, že jsou přenášeny pouze členy veřejných dat, přičemž všechny ostatní veřejné členy a vlastnosti musí používat některý z jiných podporovaných datových typů. Použijete-li třídu, která obsahuje vlastní metody, tyto metody nebudou ke klientovi přeneseny a nebudou pro něj dostupné.
Výčty (Enumerations)	Výčtový datový typ (který je definován v jazyku C# klíčovým slovem enum) je podporován. Webová služba však používá název řetězce hodnoty výčtu (nikoliv podkladové číslo).
XmlNode	Objekty založené na System.Xml.XmlNode reprezentují částí XML dokumentu. Tento typ můžete použít pro zasílání libovolného XML.
DataSet a DataTable	Tyto typy můžete použít pro vrácení informace z relační databáze. Jiné datové objekty ADO.NET, jako například DataColumn a DataRow, nejsou podporovány. Jestliže používáte sadu dat (DataSet) nebo datovou tabulku (DataTable), jsou automaticky zkonvertovány na XML podobným způsobem, jako kdybyste použili metodu GetXml() nebo WriteXml().

Třída EmployeesServices se řídí těmito pravidly. Jediné datové typy, které používá pro parametry a návratové hodnoty, jsou int a sada dat (DataSet), které jsou oba podporovány. Někteří programátoři webových služeb se nelogicky snaží na sto honů vyhýbat sadě dat (DataSet), nicméně se jedná o racionální a široce používaný přístup.

POZNÁMKA Podporované datové typy webové služby jsou založeny na typech specifikovaných ve standardu schémat XML. Dobře se mapují do základní sady datových typů C#.

Dalším požadavkem je, aby vaše webová služba byla bezstavové. Architektura webové služby v podstatě funguje stejným způsobem jako architektura webové stránky – na začátku požadavku je vytvořen nový objekt webové služby a tento objekt je zničen, jakmile je požadavek zpracován a byla vrácena odpověď. Třída `EmployeesServices` velmi dobře zapadá do tohoto modelu, protože si ve třídě členských proměnných neuchovává žádný stav. Jedinou výjimkou je proměnná `connectionString`, která je inicializována s požadovanou hodnotou při každém vytvoření třídy.

SADA DAT A WEBOVÉ SLUŽBY XML

Jistě si všimnete, že sada dat (`DataSet`) je jednou z mála specializovaných tříd .NET, které jsou podporovány webovými službami. Je to proto, že `DataSet` má schopnost se automaticky serializovat do XML. Proti této podpoře však existuje významná námitka – klienti, kteří nepoužívají .NET, a kteří mohou používat webovou službu, která vrací sadu dat, nemusí být schopni využít sadu dat XML! Je to způsobeno tím, že jiné jazyky nejsou schopny automaticky konvertovat sadu dat na snadno ovladatelné objekty. Místo toho jsou nuceny použít své vlastní programovací API pro XML. Ačkoliv tato API teoreticky fungují, v praxi to může být zdlouhavé, zejména v případě komplexního a proprietárního XML. Z tohoto důvodu se vývojáři obvykle snaží vyhnout sadě dat při vytváření webových služeb, u kterých potřebují, aby podporovaly klienty na široké paletě platform.

Stojí za zmínku, že Microsoft mohl použít přístup se sadou dat u mnoha jiných tříd .NET, aby jim umožnil její serializaci do XML. Microsoft však tyto funkce moudře nepřidal, protože si uvědomil, že ačkoliv by tento krok programátorům značně usnadnil vytváření aplikací, které jsou založeny na standardech webové služby, nebyl by praktický v mezipatformových scénářích.

Pořád ale ještě zůstává nezodpovězená otázka, proč se Microsoft rozhodl podporovat sadu dat ve své sadě nástrojů pro webové služby? Důvod je ten, že sada dat zvládá jedno z nejběžnějších použití webových služeb – vrácení aktuálních informací z relační databáze. Microsoftu se zdálo, že přínos z přidání této funkce bohatě vyváží náklady spojené s řešením problémů se součinností u vývojářů, kteří si pečlivě nepromysleli architekturu své webové služby.

Podpora obecných typů

Webové služby podporují obecné typy. Tato podpora však nemusí přesně odpovídat vašemu očekávání. Je zcela přijatelné vytvořit webovou službu, která akceptuje nebo vrací obecný typ. Chcete-li například vrátit kolekci objektů `EmployeeDetails`, můžete použít obecnou třídu `List`, jak vidíte zde:

```
public List<EmployeeDetails> GetEmployees()  
{ ... }
```

V tomto případě .NET zachází s vaší kolekcí objektů `EmployeeDetails` stejným způsobem, jako se sadou objektů `EmployeeDetails`:

```
public EmployeeDetails[] GetEmployees()  
{ ... }
```

Samozřejmě – aby tento postup fungoval, je potřeba zcela dodržovat pravidla serializace tříd `EmployeeDetails` nebo `List`. Pokud například tyto třídy mají nějakou neserializovatelnou vlastnost, nemůže být celý objekt serializován.

Důvodem, proč v tomto případě .NET podporuje obecné typy, je to, že pro .NET je poměrně snadné určit v době kompilace skutečné typy tříd. Tím je umožněno, aby .NET použilo tuto metodu pro zjištění struktury zpráv XML a přidalo příslušné informace do WSDL dokumentu (jak uvidíte v další kapitole).

.NET však nepodporuje obecné metody. Například následující metoda není povolena:

```
public List<T> GetEmployees<T>()  
{ ... }
```

V tomto případě je metoda `GetEmployees()` obecná. Umožňuje volajícímu zvolit typ, který bude používán touto metodou. Protože tato metoda může být teoreticky použita s libovolným typem dokumentu, neexistuje způsob, jak ji správně zdokumentovat a předem určit správný formát XML zprávy.

Vystavení webové služby

Když jste si nyní ověřili, že třída `EmployeesService` je připravena pro web, je čas ji zkonvertovat na webovou službu. Prvním klíčovým krokem je přidat atribut `System.Web.Services.WebMethod` ke každé metodě, kterou chcete vystavit jakožto část vaší webové služby. Tato webová služba informuje ASP.NET o tom, aby tuto metodu učinila přístupnou pro zkoumání a vzdálenému vyvolání. Zde uvádíme kód revidované třídy se dvěma metodami:

```
public class EmployeesService  
{  
    [WebMethod()]  
    public int GetEmployeesCount()  
    { ... }  
    [WebMethod()]  
    public DataSet GetEmployees()  
    { ... }  
}
```

Tyto dvě jednoduché změny dokončují transformaci vaší třídy na webovou službu.

Klient však pořád nemá vstupní bod k vaší webové službě – řečeno jinými slovy, zatím neexistuje způsob, pomocí kterého může jiná aplikace spustit vaši webovou službu. Chcete-li to umožnit, potřebujete vytvořit .asmx soubor, který vystaví vaši webovou službu.

POZNÁMKA

V tomto příkladě obsahuje webová služba kód pro přístup datům. Jestliže však plánujete použít stejný kód ve webové aplikaci, stojí za to přidat další vrstvu, která by používala databázové komponenty. Pokud chcete implementovat tento návrh, měli byste nejprve vytvořit samostatnou databázovou komponentu (jak je popsáno ve druhé části knihy), a poté tuto komponentu přímo použít na vašich webových stránkách a ve vaší webové službě.

ASP.NET implementuje webové služby jako soubory s příponou .asmx. Podobně jako u webové stránky, můžete i kód webové služby umístit přímo do souboru .asmx nebo do třídy v souboru s kódem v pozadí, na který se pak .asmx soubor odkazuje (což je přístup Visual Studia).

Je možné například vytvořit soubor s názvem EmployeesService.asmx a spojit jej s vaší třídou EmployeesService. Každý .asmx soubor začíná direktivou WebService, která deklaruje jazyk na straně serveru, který je používán v souboru a třídě. Může deklarovat také další informace, jako je například soubor s kódem v pozadí nebo informaci, zdali chcete v průběhu kompilace generovat ladící symboly. V tomto ohledu se to hodně podobá direktivě Page v .aspx souborech.

Zde uvádíme příklad .asmx souboru s EmployeesService:

```
<%@ WebService Language="C#" Class="EmployeesService" %>
```

V tomto případě máte dvě volby. Kód třídy můžete vložit ihned za atribut WebService, nebo jej můžete zkompilovat do nějaké assembly v adresáři Bin. Pokud jste třídu EmployeesService vložili do projektu Visual Studia, bude automaticky zkompilována jako DLL část webové aplikace, takže do .asmx souboru nemusíte přidávat už nic dalšího.

V této chvíli je vše hotovo. Vaše webová služba je kompletní, dostupná a připravena k použití v jiných aplikacích.

TIP Neexistuje žádný limit ohledně toho, kolik webových služeb můžete přidat do jedné webové aplikace, takže můžete libovolně kombinovat webové služby a webové stránky.

Webové služby ve Visual Studiu

Pokud používáte Visual Studio, pravděpodobně neabsolvujete proces vytváření třídy, její konverzi na webovou službu a přidání .asmx souboru. Místo toho vytvoříte .asmx soubor a kód v pozadí v jednom jediném kroku. Uděláte to tak, že zvolíte Website > Add New Item. Můžete specifikovat, zdali kód webové služby umístíte přímo do .asmx souboru nebo do samostatného souboru s kódem v pozadí (stejně jako u webové stránky).

Ještě jsme ale nezmínili dva další aspekty týkající se webové služby. Za prvé – třída webové služby je odvozena z System.Web.Services.WebService. Za druhé – atribut WebService je aplikován na deklaraci třídy. Ani jeden z těchto z těchto aspektů sice není nutný, nicméně jejich význam si ujasníme v následujících sekcích této kapitoly.

Odvozování z třídy WebService

Když vytváříte webovou službu ve Visual Studiu, vaše třída webové služby je automaticky odvozena ze základní třídy WebService, jak vidíte zde:

```
public class EmployeesService : System.Web.Services.WebService  
{ ... }
```

Dědění z třídy WebService je výhoda, která vám umožní dostat se k vestavěným objektům ASP.NET (jako jsou Application, Session a User) stejně snadno, jako v případě webového formuláře.

Tyto objekty jsou poskytovány jako vlastnosti třídy `WebService` a vaše webová služba je získává děděním. Pokud nepotřebujete použít žádný z těchto objektů (nebo pokud jste ochotni je získat přes statickou vlastnost `HttpContext.Current`), nepotřebujete dědit.

Zde uvádíme příklad toho, jak můžete ve webové službě přistupovat ke stavu `Application`, pokud odvozujete ze základní třídy `WebService`:

```
// Ulož číslo jako session.  
Session["Counter"] = 10;
```

Zde je ekvivalentní kód, který byste museli použít, pokud by vaše třída webové služby nebyla odvozena z `WebService`:

```
// Ulož číslo jako session.  
HttpContext.Current.Session["Counter"] = 10;
```

Tato technika ve skutečnosti nebude fungovat tak, jak byste chtěli (řečeno jinými slovy – klient si neudrží stejnou relaci během opakovaného volání webové metody), pokud nepodniknete několik dalších kroků, které jsou popsány v sekci "EnableSession".

Tabulka 32-4 uvádí vlastnosti, které získáte děděním z `WebService`.

Tabulka 32-4. Vlastnosti `WebService`.

Vlastnost	Popis
<code>Application</code>	Instance třídy <code>HttpApplicationState</code> , která poskytuje přístup ke globálnímu stavu webové aplikace.
<code>Context</code>	Instance třídy <code>HttpContext</code> pro aktuální požadavek.
<code>Server</code>	Instance třídy <code>HttpServerUtility</code> .
<code>Session</code>	Instance třídy <code>HttpSessionState</code> , která poskytuje přístup k aktuálnímu stavu relace.
<code>User</code>	Objekt <code>IPrincipal</code> , který vám umožní prozkoumat oprávnění a role uživatele, který byl autentizován.

Protože .NET Framework podporuje pouze jedinou dědičnost a vaše webová služba dědí z `WebService`, znamená to, že nemůže dědit z jiných tříd. Je to skutečně jediný argument proti dědění z `WebService`.

POZNÁMKA Na dědění z `WebService` je zajímavě to, že `WebService` je odvozena z třídy `System.MarshalByRefObject`. Tato třída je základní třídou používanou pro vzdálené volání .NET (.NET remoting). Výsledkem je to, že když vytváříte třídu, která je odvozena z `WebService`, získáte možnost použít vaši třídu několika různými způsoby. Můžete ji použít jako libovolnou jinou lokální třídu (a přistupovat k ní přímo ve vašich stránkách), nebo ji můžete vystavit jako část webové služby, nebo ji můžete vystavit jako distribuovaný objekt v hostiteli vzdáleného volání .NET (.NET remoting host). Chcete-li se o vzdáleném volání .NET dozvědět více informací, podívejte se na knihu *Advanced .NET Remoting* (Apress, 2005).

Dokumentování webové služby

Webové služby jsou samopopisné, což znamená, že ASP.NET automaticky poskytne klientovi veškeré potřebné informace o tom, jaké metody jsou k dispozici a jaké parametry vyžadují. Zajišťuje to standard nazvaný jako WSDL, který je založen na XML, a který si budeme podrobněji popisovat v další kapitole. Ačkoliv dokument WSDL popisuje mechanismus webové služby, nepopisuje její účel, smysl informací, které jsou dodávány jednotlivým metodám, a nepopisuje ani informace, které tyto metody vracejí. Většina webových služeb poskytuje tyto informace v samostatných vývojářských dokumentech. Můžete však – respektive byste měli – do vaší webové služby vložit alespoň minimum informací použitím atributů `WebMethod` a `WebService`.

Jednotlivým metodám můžete přidat popisy prostřednictvím vlastnosti `Description` atributu `WebMethod`, zatímco popis pro celou webovou službu jako takovou můžete přidat pomocí vlastnosti `Description` atributu `WebService`. Webové službě můžete přiřadit popisný název prostřednictvím vlastnosti `Name` atributu `WebService`. Zde uvádíme příklad, jak můžete tyto informace vložit do `EmployeesService`:

```
[WebService(Name="Employees Service",
Description="Retrieve the Northwind Employees")]
public class EmployeesService : System.Web.Services.WebService
{
    [WebMethod(Description="Returns the total number of employees.")]
    public int GetEmployeesCount()
    { ... }
    [WebMethod(
Description="Returns the full list of employees.")]
    public DataSet GetEmployees()
    { ... }
}
```

Tyto popisy jsou přidány do vašeho WSDL dokumentu, který popisuje vaši webovou službu. Ukážeme si to na automaticky generované testovací stránce, kterou budete používat v další sekci.

Vaši webovou službu byste měli doplnit o další věc – o jedinečný jmenný prostor XML. Ten umožní, aby vaše webová služba (a XML zprávy, které generuje) byla jednoznačně identifikována. S jmennými prostory XML jste se poprvé seznámili v kapitole 12. Webové služby ASP.NET standardně používají jmenný prostor XML `http://tempuri.org/`, který je vhodný pouze pro testování. Pokud nenastavíte vlastní jmenný prostor, uvidíte na testovací stránce varovné hlášení. Všimněte si, že jmenný prostor XML nemá žádný vztah ke konceptu jmenných prostorů .NET. Neovlivní ani funkci vašeho kódu, a ni způsob, jakým klient použije vaši webovou službu. Jmenný prostor XML pouze identifikuje vaši webovou službu. Jmenné prostory XML obvykle vypadají jako URL adresy. Tyto adresy nemusí být na Internetu funkční.

V ideálním případě se váš jmenný prostor odkazuje na URL adresu, která je pod vaší kontrolou. Součástí jmenného prostoru je často název domény vaší společnosti na Internetu. Pokud například používá vaše společnost webovou stránku `http://www.mycompany.com`, můžete vaši webovou službu `Employees` přiřadit tento jmenný prostor: `http://www.mycompany.com/EmployeesService`.

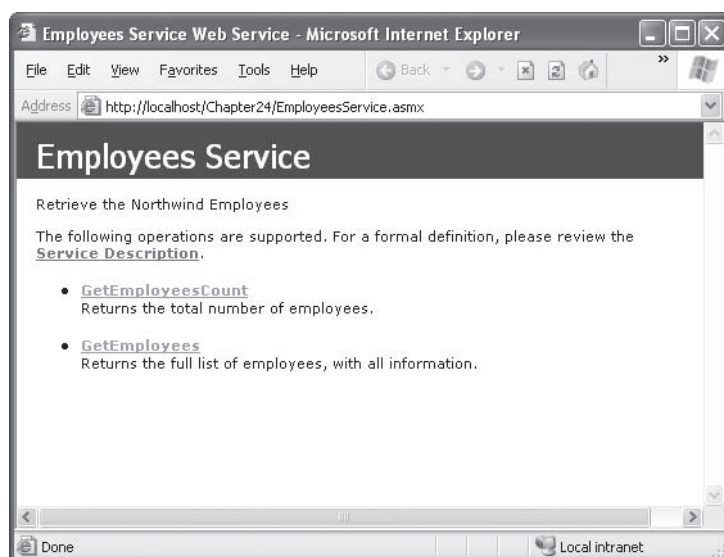
Jmenný prostor je specifikován prostřednictvím atributu `WebService`, jak vidíte zde:

```
[WebService (Name="Employees Service",Description="Retrieve the Northwind
Employees",Namespace="http://www.apress.com/ProASP.NET/")]
public class EmployeesService : System.Web.Services.WebService
{ ... }
```


Testování webové služby

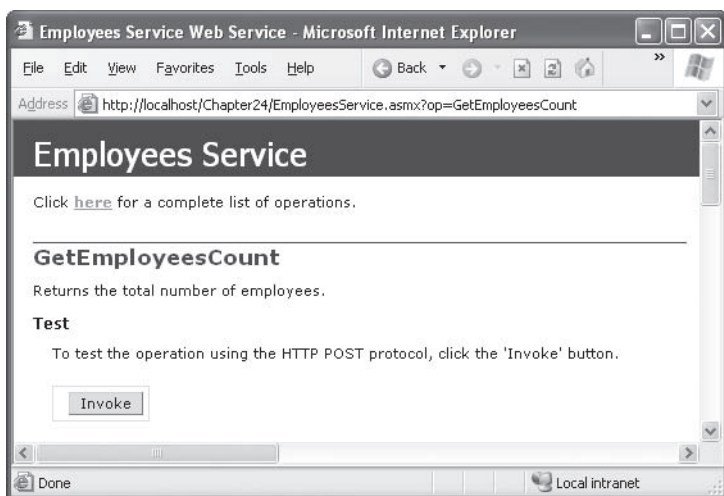
Když jsme si nyní ukázali, jak vytvořit jednoduchou webovou službu, je čas ji otestovat. Kvůli jejímu testování naštěstí nemusíte vytvářet nějakou klientskou aplikaci, protože .NET obsahuje testovací webovou stránku, kterou ASP.NET automaticky použije, když v prohlížeči zadáte URL adresu směřující na .asmx soubor. Tato testovací stránka umí přechíst a zobrazovat informace o webové službě (jako například názvy metod, které poskytuje)

Chcete-li vyzkoušet testovací stránku, požádejte ve vašem prohlížeči o soubor `EmployeesService.asmx`. (Ve Visual Studiu stačí nastavit tuto stránku jako startovací stránku pro vaši aplikaci a poté ji spustit.) Obrázek 32-3 zobrazuje tuto testovací stránku.



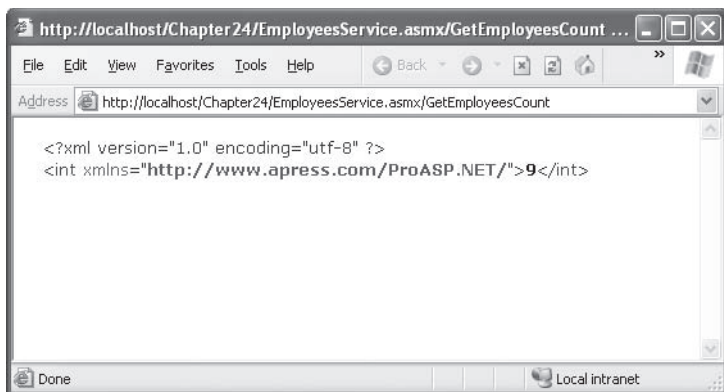
Obrázek 32-3. Testovací stránka webové služby.

Všimněte si, že stránka zobrazuje dvě webové metody s popisy. Všimněte si také toho, že název stránky je současně názvem webové služby. Pokud kliknete na jednu z metod, uvidíte stránku, která vám umožní danou metodu otestovat (a doplnit data pro jakékoliv parametry metody). Obrázek 32-4 zobrazuje stránku, která vám umožní otestovat metodu `GetEmployeesCount()`.



Obrázek 32-4. Testování webové metody.

Kliknete-li na tlačítko Invoke, objeví se nová stránka s XML dokumentem, který obsahuje požadovaná data. Podíváte-li se na obrázek 32-5, uvidíte záznamy devíti zaměstnanců. Pokud se podíváte na URL adresu v prohlížeči, zjistíte, že obsahuje soubor .asmx, který je následovaný názvem metody webové služby.



Obrázek 32-5. Výsledky testování metody `GetEmployeesCount()`.

Celý proces si můžete opakovat a zavolat metodu `GetEmployees()`. V tomto případě uvidíte mnohem podrobnější kód XML, který představuje celý obsah sady dat (jak vidíte na obrázku 32-6).

Je vidět, že díky této pomocné stránce je testování základní webové služby velice jednoduché a nevyžaduje od vás vytvoření klientské aplikace.



Obrázek 32-6. Výsledky testování metody `GetEmployees()`.

Testovací stránky nejsou součástí standardů webových služeb – představují pouze třešničku na dortu poskytovanou ASP.NET. Testovací stránku ASP.NET realizuje v podstatě za běhu, pomocí webové stránky `c:\[WinDir]\Microsoft.NET\Framework\[Version]\Config\DefaultWsdHelpGenerator.aspx`. V některých případech možná budete chtít změnit vzhled nebo chování této stránky. To provedete velmi jednoduše – zkopírujete soubor `DefaultWsdHelpGenerator.aspx` do adresáře vaší webové aplikace, upravte jej podle potřeby, a pak změňte soubor `web.config` přidáním značky `<wsdlHelpGenerator>` tak, aby se aplikace odkazovala na novou stránku určenou pro realizaci:

```
<configuration>
  <system.web>
    <webServices>
      <wsdlHelpGenerator href="MyWsdHelpGenerator.aspx"/>
    </webServices>
    <!-- Other settings omitted. -->
  </system.web>
</configuration>
```

Tato technika se nejčastěji používá pro změnu vzhledu testovací stránky. Můžete ji například využít tehdy, když chcete do této testovací stránky přidat logo vaší společnosti nebo informace o copyrightu.

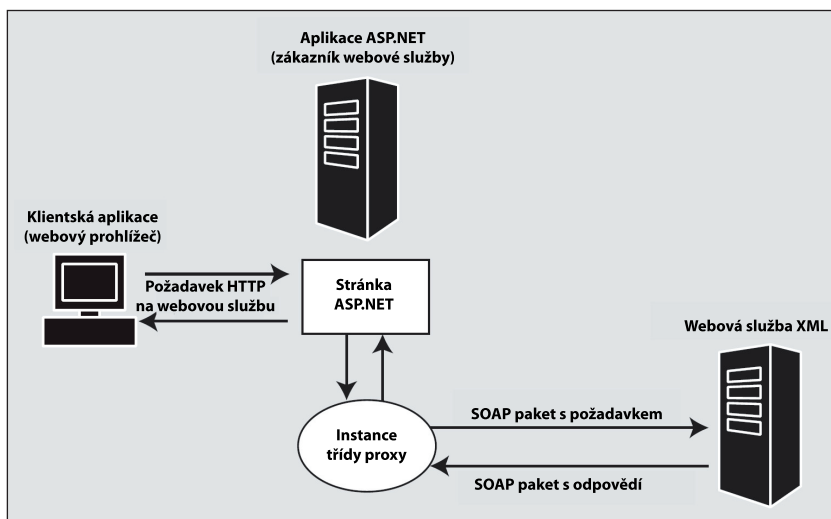
Používání webové služby

Než může klient použít webovou službu, musí být schopen vytvářet, posílat, dostávat zprávy založené na XML a chápat je. Tento proces je principiálně sice jednoduchý, nicméně v praxi je dost obtížný. Pokud byste jej chtěli sami implementovat, museli byste znova a znova psát stejný kód infrastruktury nižší úrovně.

.NET implementovat nabízí řešení v podobě jednoúčelové komponenty pojmenované jako třída proxy (proxy class), která vykoná za vaši aplikaci všechnu těžkou práci. Třída proxy zaobaluje volání metod webové služby. Hlídá vytváření správného formátu SOAP zpráv a řídí přenos zpráv po síti (pomocí HTTP). Pokud obdrží zprávu o odpovědi, zkonvertuje výsledky zpět na odpovídající datové typy .NET.

POZNÁMKA Pokud se chcete dostat k webové službě z jiného počítače, webová služba musí být dostupná. To znamená, že se nemůžete spoléhat pouze na vestavěný webový server Visual Studio, který při každém spuštění používá jiný port. Místo toho musíte pro vaši webovou službu vytvořit virtuální adresář, což bylo popsáno v kapitole 18. Jakmile jste provedli tento krok, můžete prostřednictvím vašeho webového prohlížeče požádat o webovou službu zadáním názvu virtuálního adresáře, abyste se ujistili o její dostupnosti. Poté můžete přidat odkaz na webovou službu podle kroků uvedených dále v této sekci.

Obrázek 32-7 graficky zobrazuje tento proces. V tomto příkladě je v prohlížeči spuštěna webová stránka ASP.NET, která na pozadí používá webovou službu z jiného serveru. Webová stránka ASP.NET používá pro kontakt s touto externí webovou službou třídu proxy.



Obrázek 32-7. Třída proxy webové služby.

POZNÁMKA

Díky třídě proxy (proxy class) můžete ve webové službě zavolat nějakou metodu stejně snadno jako v lokální komponentě. Toto chování samozřejmě není vždy přínosem. Webové služby mají odlišné charakteristiky než lokální komponenty. Například zavolání nějaké webové metody vám zabere značné množství času, protože každé volání musí být zkonvertováno na XML a zasláno po síti. Nebezpečí spočívá v tom, že čím více je tato skutečnost ukryta před vývojáři, tím méně je pravděpodobné, že ji vezmou v úvahu při navrhování svých aplikací.

Třidu proxy můžete v .NET vytvořit dvěma způsoby:

- Můžete použít nástroj příkazového řádku `wsdl.exe`.
- Můžete použít schopnost pro webovou referenci ve Visual Studiu.

Pomocí obou těchto přístupů dosáhnete stejných výsledků, protože používají stejné třídy z .NET Frameworku. Tyto třídy – které lze najít ve jmenných prostorech `System.Web.Services` – můžete v podstatě i spojit pro programátorské vygenerování vaší vlastní třídy proxy, ačkoliv tento přístup není z mnoha důvodů příliš praktický.

V následujících sekcích se dozvíte, jak můžete použít nástroj `wsdl.exe` a Visual Studio pro vytvoření proxy tříd. Dále se naučíte, jak používat webové služby ve třech typech klientů – na webové stránce ASP.NET, v aplikaci Windows a na klasické stránce ASP.

TIP

Rozdíl mezi přístupem `wsdl.exe` a webovou referencí je v tom, že když použijete ve webové aplikaci webovou referenci, nebudete schopni se podívat na aktuální proxy kód (který je vygenerován později v procesu kompilace). To znamená, že pokud chcete vylepšovat kód proxy třídy (nebo se na něj pouze podívat), a pokud vytváříte webového klienta, musíte použít `wsdl.exe`. Toto omezení neplatí pro jiné typy klientů, kteří nepoužívají kompilační model ASP.NET, a proto je kód třídy proxy přidáván přímo do projektu.

Generování třídy proxy pomocí `wsdl.exe`

Nástroj `wsdl.exe` vezme webovou službu a vygeneruje zdrojový kód třídy proxy buď v jazyce C# nebo v jazyce VB.NET. Název WSDL je odvozeno ze standardu webové služby (Web Services Description Language), který se používá pro popis funkcionality poskytované webovou službou. Více o WSDL se dozvíte v další kapitole.

Soubor `wsdl.exe` naleznete v adresáři .NET Frameworku, k němuž obvykle vede cesta podobná této: `c:\Program Files\Microsoft Visual Studio 2005\SDK\v2.0\Bin` (podle toho, jakou verzi Visual Studio máte nainstalovanou). Tento soubor je utilitou příkazového řádku, takže nejsnadněji se vám bude používat tehdy, když ve Windows otevřete okno příkazového řádku.

V ASP.NET můžete požádat o WSDL dokument specifikováním URL adresy webové služby a připojením parametru `?WSDL`. (Další způsob je použít odlišnou URL adresu nebo dokonce soubor s obsahem WSDL.) Zde je uvedena nejkratší možná syntaxe pro vygenerování této třídy:

```
wsdl http://localhost/Webservices1/EmployeesService.asmx
```

Tato třída je standardně vygenerována v jazyce C#. Pokud chcete třídu vygenerovat v jiném jazyce, můžete to provést přidáním parametru `/language`, jak vidíte zde:

```
wsdl /language:VB http://localhost/Webservices1/EmployeesService.asmx
```

Vygenerovaný soubor se obvykle jmenuje stejně jako webová služba (název webové služby je specifikován prostřednictvím vlastnosti Name atributu WebService). Název souboru můžete snadno změnit přidáním parametru /out k příkazu wsdl.exe. Pomocí parametru /namespace je možné změnit jmenný prostor vygenerované třídy. Zde uvádíme příklad, který je z důvodu rozměrů této knihy rozdělen do dvou řádků:

```
wsdl /namespace:ApressServices /out:EmployeesProxy.cs
http://localhost/WebServices1/EmployeesService.asmx
```

Tabulka 32-5 pak podává přehled podporovaných parametrů.

Tabulka 32-5. Parametry Wsdl.exe.

Parametr	Popis
<url nebo cesta>	URL nebo cesta k WSDL kontraktu, XSD schématu nebo k dokumentu .discomap.
/nologo	Potlačuje bannery.
/language:<language>	Jazyk, který má být použit pro generovanou třídu proxy. Můžete si vybrat mezi CS, VB, JS, nebo můžete poskytnout plně kvalifikované jméno pro třídu implementující System.CodeDom.Compiler.CodeDomProvider. Výchozí jazyk je C#. Zkrácenou forma parametru je /l.
/server	Vygeneruje abstraktní třídu pro implementaci webové služby na základě kontraktů. Standardně jsou vygenerovány klientské třídy proxy.
/namespace:<namespace>	Jmenný prostor vygenerované proxy nebo šablony. Výchozí je globální jmenný prostor. Zkrácenou formou je /n.
/out:<fileName>	Název souboru vygenerovaného proxy kódu. Výchozí název je odvozen z názvu služby. Zkrácenou formou je /o.
/protocol:<protocol>	Změní standardní protokol, který se má implementovat. Můžete si vybrat mezi SOAP (u SOAP 1.1), SOAP12 (u SOAP 1.2), HTTP-GET, HTTP-POST nebo vlastním protokolem, který je specifikován v konfiguračním souboru.
/username:<username>, /password:<password>, /domain:<domain>	Pověřovací doklady, které jsou nutné pro připojení k serveru, který požaduje autentizaci. Zkrácené formy těchto parametrů jsou /u, /p a /d.
/proxy:<URL>	URL proxy serveru, který má být použit pro HTTP požadavky. Výchozí je systémové nastavení proxy.
/proxyusername:<username>, /proxypassword:<password>, /proxydomain:<domain>	Pověřovací doklady, které jsou nutné pro připojení k proxy serveru, který požaduje autentizaci. Zkrácené formy těchto parametrů jsou /pu, /pp a /pd.
/appsettingurkey:<key>	Konfigurační klíč, který se má použít při generování kódu pro čtení výchozí hodnoty vlastnosti URL. Výchozí nastavení je nečíst z konfiguračního souboru. Zkrácenou formou parametru je /urlkey.

Parametr	Popis
/appsettingbaseurl:<baseURL>	Základní URL adresa, který má být použita při zjišťování fragmentu URL. Volba appsettingurlkey musí být v takovém případě specifikována. Fragment URL představuje výsledek zjištění relativní URL z appsettingbaseurl k URL adrese ve WSDL dokumentu. Zkrácenou formou parametru je /baseurl.
/fields	Je-li parametr nastaven, jakékoliv komplexní typy použité webovou službou budou místo veřejných vlastností sestaveny z veřejných polí. V kapitole 33 se mnohem podrobněji dozvíte, jak komplexní typy fungují s webovými službami.
/sharetypes	Umožní vám přidat odkaz na dvě nebo více webových služeb, které používají stejné komplexní typy. Tato technika je popsána v následující kapitole.
/serverinterface	Vygeneruje rozhraní pomocí dvou metod WSDL dokumentu. Toto rozhraní můžete implementovat pro vytvoření vaší webové služby. Tato technika je popsána v následující kapitole.

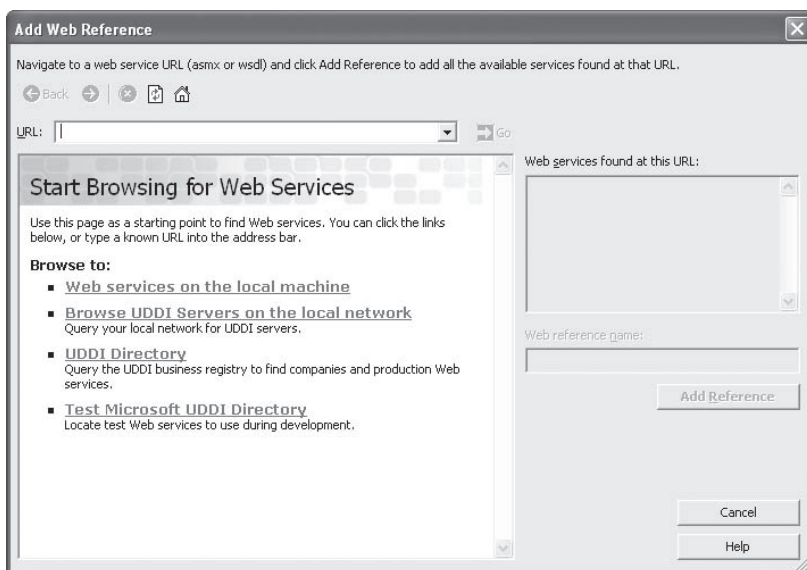
Jakmile jste vytvořili tento soubor, musíte jej zkopírovat do adresáře `App_Code`, aby byla tato třída dostupná všem stránkám vaší webové aplikace. Vytváříte-li bohatou klientskou aplikaci (jako je například formulářová aplikace Windows), měli byste tento soubor raději přidat přímo do projektu, aby byl zkompilován do výsledného EXE souboru.

Generování třídy proxy ve Visual Studiu

Ve Visual Studio vytváříte třídu proxy přidáním webové reference (web reference) do klientského projektu. Webové reference jsou podobné obyčejným odkazům, nicméně však nesměřují k různým assembly s obyčejnými typy .NET, nýbrž ukazují na URL webové služby s WSDL kontraktem.

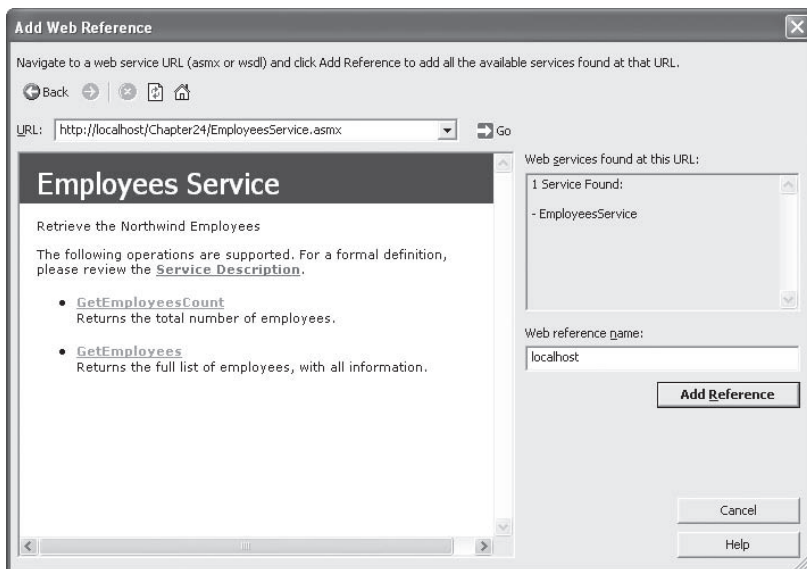
Chcete-li vytvořit webovou referenci, postupujte podle následovně:

1. Klikněte pravým tlačítkem do klientského projektu v okně Solution Explorer a vyberte položku Add Web Reference.
2. Nyní se vám otevře se dialogové okno Add Web Reference, jak vidíte na obrázku 32-8. Toto okno vám poskytuje volbu pro vyhledání webových registrů, nebo přímé vložení URL adresy. Obsahuje také odkaz, který vám umožní procházet všechny webové služby na lokálním počítači nebo vyhledat UDDI registr.



Obrázek 32-8. Dialogové okno Add Web Reference.

3. Vložíte-li URL adresu, která bude směřovat na soubor .asmx, dostanete se přímo na danou webovou službu. V okně se pak objeví testovací stránka (jak vidíte na obrázku 32-9) a zpřístupní se tlačítko Add Reference.



Obrázek 32-9. Přidání webové reference.

4. V textovém poli Web Reference Name můžete změnit jmenný prostor, ve kterém bude vygenerována třída proxy.

5. Pokud chcete přidat referenci na tuto webovou službu, klikněte na tlačítko Add Reference v dolní části okna.
6. Webová reference se nyní objeví ve skupině Web References vašeho projektu v okně Solution Explorer.

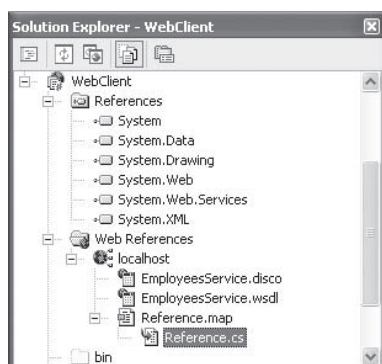
Vámi vytvořená webová reference používá WSDL kontrakt a informace, které jsou k dispozici v době přidání reference. Je-li webová služba změněna, musíte vaši třídu proxy aktualizovat. To provedete tak, že kliknete na webovou referenci pravým tlačítkem a zvolíte Update Web Reference. Na rozdíl od lokálních komponent nejsou webové reference automaticky aktualizovány při opětovné kompilaci aplikace.

TIP Při vytváření a testování webové služby ve Visual Studio je často nejjednodušší přidat webovou službu a klientskou aplikaci do stejného řešení. Umožní vám to testovat a měnit oboje najednou. Můžete dokonce použít integrovaný debugger, kterým nastavíte body přerušení v kódu klienta i serveru, jako by skutečně byly jedinou aplikací. Chcete-li si vybrat, kterou aplikaci má Visual Studio spustit po kliknutí na Start, klepněte pravým tlačítkem myši na název odpovídajícího projektu v okně Solution Explorer a vyberte Set As StartUp Project.

Když přidáte webovou referenci, Visual Studio uloží ve vašem projektu kopii WSDL dokumentu. Kde tuto informaci uloží, záleží na typu projektu.

Visual Studio ve webové aplikaci vytvoří složku App_WebReferences (pokud ještě neexistuje), a v ní vytvoří další složku s názvem webového reference (kterou zvolíte v dialogovém okně Add Web Reference). Visual Studio nakonec do této složky umístí soubory webové služby. Visual Studio však nevygeneruje proxy třídu. Ta je vytvořena a cachována v rámci kompilačního procesu ASP.NET, což je změna v chování oproti ASP.NET 1.x.

V jiném typu aplikace Visual Studio vytvoří složku WebReferences a v této složce vytvoří další složku s názvem webového reference. Do této složky umístí všechny podporované soubory, které vidíte ve webové aplikaci (nejdůležitější je kopie WSDL dokumentu). Vytvoří rovněž soubor s názvem Reference.cs (za předpokladu, že se jedná o C# aplikaci) se zdrojovým kódem třídy proxy, jak vidíte na obrázku 32-10. Tato třída je standardně skryta. Chcete-li ji zobrazit, vyberte Project/Show All Files.



Obrázek 32-10. WSDL kontrakt a třída proxy.

Dynamické URL

V předchozích verzích .NET byl URL adresa webové služby zakódována do konstruktoru třídy. Pokud však vytvoříte ve Visual Studio 2005 webovou referenci, umístění je vždy ukládáno do konfiguračního souboru. Tento postup je užitečný, protože vám umožňuje změnit umístění webové služby při rozmisťování aplikace, aniž byste museli opětovně generovat třídu proxy.

Přesné umístění tohoto nastavení závisí na typu aplikace. Je-li klientem webová aplikace, bude tato informace přidána do souboru web.config, jak vidíte zde:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="localhost.EmployeesService"
      value="http://localhost/Webservices1/EmployeesService.asmx"/>
  </appSettings>
  ...
</configuration>
```

Vytváříte-li jiný typ klientské aplikace, například aplikaci Windows, název konfiguračního souboru bude ve formátu [AppName].exe.config. Je-li vaše aplikace například pojmenována jako SimpleClient.exe, konfigurační soubor bude ve tvaru SimpleClient.exe.config. Tuto konvenci musíte dodržet.

TIP Visual Studio trochu čaruje s pojmenováváním konfiguračních souborů. V prostředí návrhu bude mít konfigurační soubor název App.config. Avšak při vytváření aplikace bude tento soubor zkopírován do vytvořeného adresáře a bude mu přiřazen odpovídající název (v souladu se spustitelným souborem). Jedinou výjimkou je případ, kdy je klientská aplikace webovou aplikací. Všechny webové aplikace používají konfigurační soubor názvem web.config bez ohledu na to, jaké názvy souborů používáte. To také uvidíte v návrhovém prostředí.

Chcete-li specifikovat název nastavení, musíte použít utilitu wsdl.exe s /appsettingurlkey. Můžete použít například tento příkaz pro příkazový řádek:

```
wsdl http://localhost/Webservices1/EmployeesService.asmx
/appsettingurlkey:WsUrl
```

V tomto případě je klíč uložen společně s klíčem WsUrl v sekci <appSettings>.

Třída proxy

Jakmile vytvoříte třídu proxy, stojí za to se blíže podívat na vygenerovaný kód, abyste zjistili, jak funguje.

Třída proxy má stejný název jako třída webové služby. Dědí z SoapHttpClientProtocol, která obsahuje vlastnosti jako například Credentials, Url a Timeout, o nichž se více dozvíte v následujících sekcích. Zde uvádíme deklaraci třídy proxy, která poskytuje komunikaci s EmployeesService:

```
public class EmployeesService :
System.Web.Services.Protocols.SoapHttpClientProtocol
{ ... }
```

Třída proxy obsahuje kopie všech metod webové služby. Verze obsažená v třídě proxy však neobsahuje obchodní kód. (Klient v podstatě nemůže žádným způsobem získat jakékoliv informace o interním zpracování kódu vaší webové služby, protože by to představovalo závažnou trhlinu v bezpečnosti). Třída proxy místo toho obsahuje kód potřebný pro požadavky vzdálené webové služby a pro konverzi výsledků. Zde je například uvedena metoda `GetEmployeesCount()` třídy proxy:

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute()]
public int GetEmployeesCount()
{
    object[] results = this.Invoke("GetEmployeesCount", new object[0]);
    return ((int)(results[0]));
}
```

Tato metoda vyvolá základní metodu `SoapHttpClientProtocol.Invoke()`, která vytvoří SOAP zprávu a bude čekat na odezvu. Zbývající část kódu zkonvertuje vrácený objekt na celé číslo.

POZNÁMKA Třída proxy má také další metody, které podporují asynchronní volání webových metod. Více o asynchronních voláních se dozvíte v kapitole 34, kde budou uvedeny také praktické příklady jejich použití.

Třída proxy je ukončena proxy kódem metody `GetEmployees()`. Všimněte si, že tento kód je téměř identický s kódem používaným pro metodu `GetEmployeesCount()` – jediným rozdílem je název metody, který byl předán metodě `Invoke()`. Všimněte si také, že návratová hodnota je konvertována na sadu dat (`DataSet`), nikoliv na celé číslo.

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute()]
public System.Data.DataSet GetEmployees()
{
    object[] results = this.Invoke("GetEmployees", new object[0]);
    return ((System.Data.DataSet)(results[0]));
}
```

Vytvoření klienta ASP.NET

Když máte nyní webovou službu a třídu proxy, je poměrně snadné vytvořit jednoduchého klienta – webovou stránku. Pokud používáte Visual Studio, je prvním krokem vytvoření nového webového projektu a přidání webové reference (web reference) na webovou službu. Používáte-li jiný nástroj, budete nejdříve muset zkompilovat třídu proxy pomocí `wsdl.exe`, a pak ji umístit do adresáře Bin nové webové aplikace.

V následujícím příkladě je použita jednoduchá webová stránka s tlačítkem a ovládacím prvkem `GridView`. Když uživatel klikne na tlačítko, webová stránka je zaslána zpět, vytvoří se třída proxy, z webové služby je získán `DataSet` zaměstnanců a výsledek je pak zobrazen vázáním na mřížku.

Než přidáte tento kód, je užitečné importovat jmenný prostor třídy proxy. Ve Visual Studiu je jmenným prostorem automaticky jmenný prostor aktuálního projektu a jmenný prostor, který jste specifikovali v dialogovém okně `Add Web Reference` (standardně je to `localhost`). Za předpokladu, že váš projekt má název `WebClient`, webová služba je na lokálním počítači a v dialogovém okně `Add Web Reference` jste neprovedli žádné změny, použijete tento jmenný prostor:

1232 Kapitola 32 – Tvorba webových služeb

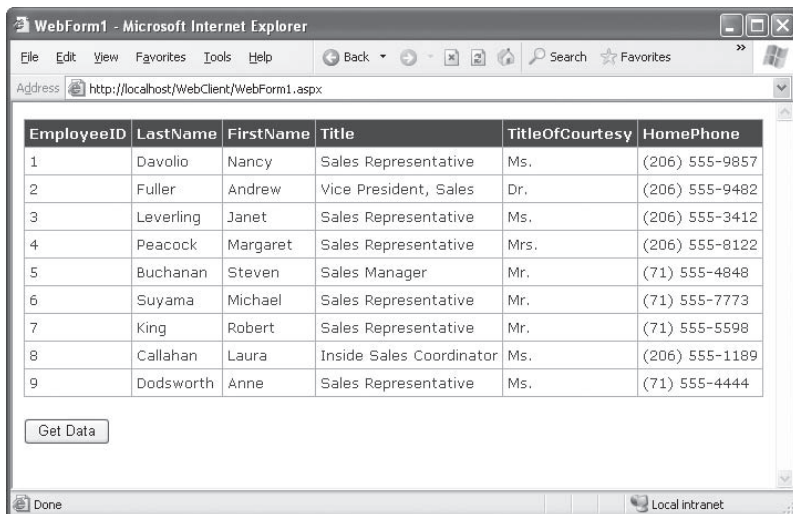
```
using WebClient localhost;
```

Nyní můžete přidat kód, který třída proxy použije k získávání dat:

```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
    // Vytvoř proxy.
    EmployeesService proxy = new EmployeesService();
    // Vyvolej webovou službu a získej výsledky.
    DataSet ds = proxy.GetEmployees();
    // Navaž výsledky.
    GridView1.DataSource = ds.Tables[0];
    GridView1.DataBind();
}
```

Protože třída proxy má stejný název jako třída webové služby, když klient vytvoří instanci třídy proxy, zdá se, že prostřednictvím klienta je instanciována aktuální webová třída. Pro zdůraznění rozdílu mezi nimi tento kód pojmenovává proxy proměnné objektu.

Spustíte-li stránku, uvidíte výsledek, který je zobrazen na obrázku 32-11.



Obrázek 32-11. Zobrazení dat z webové služby na webové stránce.

Zajímavé je, že vázání dat nemusíte provádět ručně. Pro přímé vázání k odpovídající třídě proxy můžete použít `ObjectDataSource` (popsaný v kapitole 9):

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
SelectMethod="GetEmployees" TypeName="localhost.EmployeesService" />
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="True"
DataSourceID="ObjectDataSource1" />
```

Z pohledu kódu webové stránky neexistuje rozdíl mezi vyvoláním webové služby a použitím obvyklé bezstavové třídy. Musíte si však uvědomit, že webová služba, která implementuje obchodní logiku, se může nacházet

na webovém serveru na druhé straně zeměkoule. Kvůli tomu musíte omezit počet volání této služby a být připraveni na zpracování výjimek, které vyplývají z síťových problémů a chyb spojení.

Vlastnost Timeout

Třída proxy obsahuje vlastnost Timeout, která vám umožňuje v milisekundách specifikovat maximální dobu, po kterou jste ochotni čekat. Výchozí časová prodleva je 10000 milisekund (10 sekund).

Při použití vlastnosti Timeout musíte nastavit zpracování chyb. Jestliže časová prodleva vyprší bez odezvy, vznikne výjimka, která umožňuje informovat uživatele o vzniklé chybě.

Zde uvádíme způsob, jak přepsat klienta – webovou stránku ASP.NET – tak, aby použil časovou prodlevu o délce tří sekund:

```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
    // Vytvoř proxy.
    EmployeesService proxy = new EmployeesService();
    // Tato časová prodleva bude platit pro všechna volání
    // metody webové služby.
    proxy.Timeout = 3000; // 3,000 milliseconds je 3 seconds.
    DataSet ds = null;
    try
    {
        // Vyvolej webovou službu a získej výsledky.
        ds = proxy.GetEmployees();
    }
    catch (System.Net.WebException err)
    {
        if (err.Status == WebExceptionStatus.Timeout)
        {
            lblResult.Text = "Web service timed out after 3 seconds.";
        }
        else
        {
            lblResult.Text = "Another type of problem occurred.";
        }
    }
    // Navaž výsledky.
    if (ds != null)
    {
        GridView1.DataSource = ds.Tables[0];
        GridView1.DataBind();
    }
}
```

Časovou prodlevu můžete také nastavit na hodnotu -1, což znamená, že budete čekat neomezenou dobu. To však vaši webovou aplikaci nepříjemně zpomalí, pokud se pokusíte o vykonání několika operací v situaci, kdy webová služba nereaguje.

Spojení přes proxy

Třída proxy obsahuje určitou inteligenci, která vám umožňuje přesměrovat její HTTP komunikaci pomocí speciálních internetových nastavení. Třída proxy standardně používá nastavení Internetu aktuálního počítače. V některých sítích to nemusí být ten nejlepší přístup. Tato nastavení můžete přepsat použitím vlastnosti Proxy třídy proxy webové služby.

TIP V tomto případě je slovo proxy používáno ve dvou významech – jednou ve významu proxy, která řídí komunikaci mezi klientem a webovou službou, a podruhé ve významu proxy serveru vaší organizace, který řídí komunikaci mezi počítačem a Internetem.

Pokud se například potřebujete připojit přes počítač s názvem ProxyServer, který používá port 80, ještě předtím, než zavoláte jakékoliv metody webové služby, můžete použít následující kód:

```
// Vytvoř proxy webové služby.  
EmployeesService proxy = new EmployeesService();  
// Specifikuj proxy server pro síťovou komunikaci.  
WebProxy connectionProxy = new WebProxy("ProxyServer", 80);  
proxy.Proxy = connectionProxy;
```

Třída WebProxy má mnoho dalších voleb, které vám umožňují podrobněji nastavit spojení a nastavit informace o autentizaci pro případ více komplikovanějších scénářích.

Vytvoření klienta formulářů Windows

Jednou z hlavních výhod webových služeb je způsob, jakým vám umožňují přístup k lokálním aplikacím, jako jsou například bohaté klientské aplikace, které jsou přístupné pomocí webu. Pomocí webové služby můžete vytvořit desktopovou aplikaci, která bude získávat nejnovější data z webového serveru. Tento proces je úplně průhledný. Protože vysokorychlostní připojení k Internetu je stále běžnější záležitostí, možná si ani neuvědomujete, které schopnosti aplikace jsou závislé na Internetu, a které nikoliv.

V aplikaci Windows můžete použít funkcionalitu webové služby stejným způsobem, jako v aplikaci ASP.NET. Nejprve vytvoříte třídu proxy pomocí Visual Studia nebo utility wsdl.exe. Dále přidáte kód pro vytvoření instance třídy proxy a zavoláte webovou metodu. Jediným rozdílem je uživatelské rozhraní používané v aplikaci.

Pokud se moc nevyznáte v desktopovém programování s .NET, bude pro vás příjemným zjištěním, že mnoho z toho, co jste se naučili při vývoji s ASP.NET, můžete použít i zde. Spousta webových ovládacích prvků (jako jsou popisky, tlačítka, textové políčka a seznamy) se podobá svým desktopovým .NET ekvivalentům a kód, který vytváříte pro interakci s těmito prvky, může být často přenesen z jednoho prostředí do druhého pouze s několika změnami. Nejvýznamnější rozdíl mezi desktopovým programováním a webovým programováním v .NET jsou kroky, které musíte podniknout ve webové aplikaci navíc, abyste zachovali informace mezi jednotlivými postbacky a při přenosu uživatele z jedné stránky na jinou.

Chcete-li začít ve Visual Studiu vytvářet svého klienta Windows, vytvořte nový projekt aplikace Windows, a poté přidejte webovou referenci. Webové projekty začínají jediným výchozím formulářem, který můžete navrhnout velice podobným způsobem jako webovou stránku. V našem případě musíte z Toolboxu přetáhnout kurzorem myši ovládací prvky Button a DataGridView.

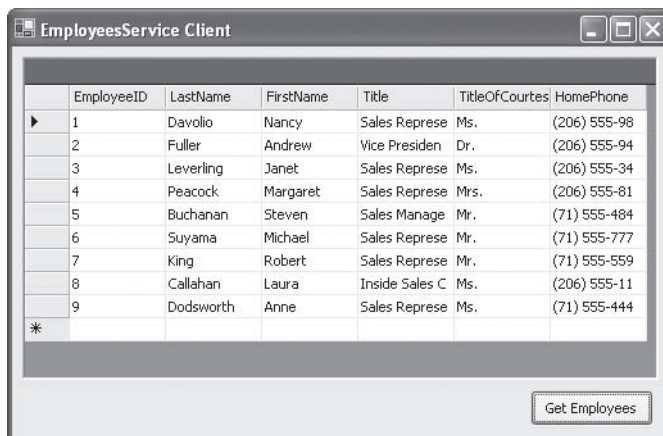
Začněte s importováním potřebného jmenného prostoru. Uděláte to podobně jako v případě ASP.NET stránky:

```
using WindowsClient.localhost;
```

Pro tlačítko přidejte kód pro obsluhu událostí. Tento kód získává sadu dat (DataSet) a zobrazuje ji ve formuláři. Pro aplikace Windows probíhá vázání dat poněkud odlišně. Po nastavení zdroje dat například nemusíte zavolat explicitní metodu DataBind(). Tento kód rovněž přináší určité vylepšení – v době, kdy probíhá volání webové služby, je v aplikaci použit kurzor přesýpacích hodin – díky tomu uživatel ví, že operace se právě provádí.

```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;
    // Vytvoř proxy.
    EmployeesService proxy = new EmployeesService();
    // Vyvolej webovou službu a získej výsledky.
    DataSet ds = proxy.GetEmployees();
    // Navaž výsledky.
    dataGridView1.DataSource = ds.Tables[0];
    this.Cursor = Cursors.Default;
}
```

Obrázek 32-12 zobrazuje výsledek, který uvidíte, když spustíte klienta Windows a kliknete na tlačítko, které získá data webové služby.



Obrázek 32-12. Zobrazení dat webové služby ve formuláři Windows.

Je samozřejmé, že při vývoji aplikací Windows máte spoustu jiných možností, které jsou popsány v mnoha vynikajících knihách. Z vašeho hlediska je zajímavé, že klient Windows komunikuje s webovou službou

úplně přesně jako aplikace ASP.NET. Tím se vám otevírá spousta nových možností pro sjednocení aplikací Windows a webových aplikací. Tuto aplikaci Windows můžete například rozšířit tak, aby uživateli umožňovala modifikovat data zaměstnanců. Poté můžete přidat k EmployeesService příslušné metody, které klientovi umožní odeslat změněná data a provést změny v databázi.

Je důležité pochopit, že kroky, které zajišťují používání ukázkové webové služby, jsou stejné jako u libovolné jiné webové služby třetí strany. Poskytovatelé webové služby nemusí distribuovat své třídy proxy, protože programovací platformy (jako .NET) obsahují nástroje k jejich automatickému vygenerování.

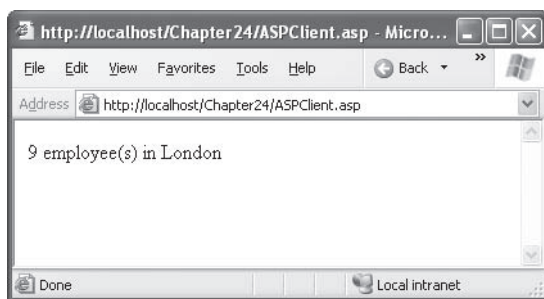
TIP Pokud byste chtěli zkusit použít jiné webové služby než webové služby .NET, můžete si je vyhledat v katalogu webových služeb XMethods (<http://www.xmethods.com>). Další užitečné webové služby můžete také najít v MapPointu společnosti Microsoft (http://msdn.microsoft.com/library/en-us/dnanchor/html/anch_mappointmain.asp). Je to zajímavá služba, která vám umožní se dostat velmi kvalitním mapám a geografickým informacím. Další službou od Microsoftu je TerraService (<http://terraservice.net/web-services.aspx>), která je založena na velice populární stránce TerraServer, na které si uživatelé webu mohou prohlédnout topografické mapy a satelitní fotografie zeměkoule. Pomocí TerraService můžete nejenom vyhledávat informace o různých místech na zeměkouli, ale také můžete stahovat satelitní fotografie určitých oblastí.

Vytvoření ASP klienta s MSXML

Nyní vám ukážeme, jak může být webová služba zavolána odvozenou aplikací libovolného typu či platformy. Následující příklad demonstruje základní přístup pro zobrazení dat ve stránce ASP:

```
<script language="VBScript" runat="Server">
Option Explicit
Dim URL
URL = "http://localhost/Webservices1/EmployeesService.asmx/GetEmployeesCount"
Dim objHTTP
Set objHTTP = CreateObject("Microsoft.XMLHTTP")
objHTTP.Open "POST", URL, False
objHTTP.Send
Dim numEmp
numEmp = objHTTP.responseXML.documentElement.Text
Response.Write(numEmp & " employee(s) in London")
</script>
```

Tento kód jednoduše nastaví URL tak, aby směřovala k webové metodě ve webové službě. Kód dále použije třídu Microsoft.XMLHTTP (z Microsoft XML Parseru, což je COM komponenta, která poskytuje třídy pro manipulaci s XML daty, zaslání HTTP příkazů a získávání odpovídajících odpovědí). Jejím prostřednictvím otevře HTTP spojení a synchronním způsobem pošle příkaz. V tomto případě se kód dostává k webové službě přes příkaz HTTP POST, který je webovými službami ASP.NET podporován pouze na lokálním počítači. Když je metoda send vrácena, text odpovědi je uložen do vlastnosti responseXML. Tato vlastnost je poskytována jako objekt MSXML2.DOMDocument s vlastností documentElement, která směřuje na kořenový uzel vrácených dat XML. Pomocí tohoto objektu můžete navigovat odpověď XML. Protože v tomto případě data obsahují celočíselný výsledek, můžete použít vlastnost text pro přečtení hodnoty tohoto prvku. Na obrázku 32-13 pak vidíte výsledek.



Obrázek 32-13. Zobrazení dat webové služby na ASP stránce.

Na tomto příkladu je zajímavé, že je použita knihovna Microsoft XML, která obsahuje třídy pro zaslání příkazů přes HTTP, získávání textu odpovědi a parsování XML. Je to všechno, co potřebujete. Pokud ještě nemáte tuto společnou komponentu, můžete si ji stáhnout z <http://msdn.microsoft.com/library/en-us/xmlsdk/html/xmmscxmllinstallregister.asp>. Všimněte si, že předchozí kód můžete s nepatrnými změnami použít v jakékoliv aplikaci VBScript nebo VBA, včetně makra v Microsoft Office nebo klienta WSH (Windows Scripting Host).

Nyní se zamyslete nad mnohem složitějším příkladem, a sice metodou `GetEmployees()`, která vrací úplnou sadu dat. Chcete-li pracovat s těmito daty, je potřeba se důkladněji podívat na odpověď XML. Zde je kód, který provádí cyklus prostřednictvím značek `<Employees>`, a který extrahuje několik informací o každém zaměstnanci. Z těchto informací je pak vytvářen seznam:

```
<script language="VBScript" runat="Server">
Option Explicit
Dim URL
URL = "http://localhost/Webservices1/EmployeesService.asmx/GetEmployeesCount"
Dim objHTTP
Set objHTTP = CreateObject("Microsoft.XMLHTTP")
objHTTP.Open "POST", URL, False
objHTTP.Send
Dim Doc
Set Doc = objHTTP.responseXML
Dim Child
For Each Child In Doc.documentElement.childNodes(1).childNodes(0).childNodes
    Response.Write(Child.childNodes(0).Text + "<br>")
    Response.Write(Child.childNodes(1).Text)
    Response.Write(Child.childNodes(2).Text + "<br><br>")
Next
</script>
```

Vytvoření ASP klienta pomocí SOAP Toolkit

V předchozím příkladě jsme si ukázali nejnižší společný jmenovatel pro vyvolání webové služby – zaslání zprávy přes HTTP a ruční parsování vráceného XML. Většina klientů má ovšem přístup k mnohem robustnějším sadám nástrojů, které přímo podporují SOAP. Jedním z příkladů takové sady je Microsoft SOAP To-

olkit, což je COM komponenta, kterou můžete použít k zavolání libovolného typu webové služby, která poskytuje platný WSDL dokument.

SOAP Toolkit podporuje webové služby .NET a webové služby vytvořené na jiných platformách. SOAP Toolkit vám poslouží, pokud chcete použít webové služby v aplikacích založených na COM, jako jsou aplikace vytvořené ve Visual Basicu 6, Visual C++ 6 a ASP. Nejnovější verze Microsoft SOAP Toolkit je k dispozici na adrese http://msdn.microsoft.com/webservices/_building/soaptk.

Chcete-li Microsoft SOAP Toolkit použít, potřebujete znát umístění WSDL služby, kterou chcete použít. SOAP Toolkit pomocí tohoto WSDL dokumentu dynamicky vygeneruje proxy. Proxy sice nevidíte, protože je vytvořena při běhu, nicméně ji můžete použít pro získání přístupu k modelu stejné webové služby vyšší úrovně.

V následujícím příkladu je stránka ASP přepsána tak, aby používala SOAP Toolkit. V tomto případě je WSDL dokument získán přímo z webového serveru. V zájmu zvýšení výkonu se doporučuje uložit lokální kopii WSDL souboru a použít ji při konfiguraci objektu SoapClient.

```
<script language="VBScript" runat="Server">
Option Explicit
Dim SoapClient
Set SoapClient = CreateObject("MSSOAP.SoapClient")
' Generování proxy.
Dim WSDLPPath
WSDLPPath = "http://localhost/Webservices1/EmployeesService.asmx?WSDL"
SoapClient.MSSoapInit WSDLPPath
' Čtení zaměstnanců.
Dim numEmp
numEmp = SoapClient.GetEmployeesCount()
Response.Write(numEmp & " employee(s) in London")
</script>
```

Všimněte si, že v našem příkladě nemusíte číst žádný XML. Místo toho klient přímo vyvolá metodu GetEmployeesCount() prostřednictvím objektu SoapClient. Tento přístup vám ovšem nepomůže při manipulaci se sadou dat, protože neexistuje COM ekvivalent této třídy. Místo toho se musíte vrátit k parsování XML, jak jste viděli v předchozím příkladě.

Je samozřejmé, že Java, C++, Delphi atd. používají své vlastní komponenty, knihovny a API pro vytváření HTTP spojení, zasílání příkazů a parsování XML textu, základní přístup je ovšem stejný.

Vylepšování webové služby

V předchozích sekcích jste se naučili, jak vytvořit základní webovou službu pomocí několika metod. Naučili jste se také vytvořit webovou stránku a aplikaci Windows, které budou tuto službu využívat. Ještě jsme se však nezabývali celou řadou funkcí webové služby. Webová metoda může například cachovat data, používat stav relace a provádět transakce.

Až do konce této kapitoly se budeme věnovat způsobům použití těchto technik.

Tajemství použití těchto funkcí spočívá v atributu WebMethod. V příkladech, které jsme uvedli doposud, byl atribut WebMethod použit pro označení metod, které chcete zpřístupnit jako část webové služby, a pro

přidání popisu prostřednictvím vlastnosti Description. Existuje však několik dalších vlastností atributu WebMethod. Jsou uvedeny v tabulce 32-6.

Tabulka 32-6. Vlastnosti atributu WebMethod.

Argument	Popis
Description	Popis metody.
MessageName	Alias jména metody, který je používán v případě, že máte přetěžované verze metody nebo tehdy, pokud chcete metodu zpřístupnit pod jiným názvem. Tato technika není doporučována.
CacheDuration	Počet sekund, které odpovídají době, po kterou bude odpověď metody uložena cache. Výchozí nastavení je nula, což znamená, že odpověď není cachována.
EnableSession	Získá nastavení, nebo sám nastaví, zdali metoda může přistupovat k informacím z kolekce Session.
BufferResponse	Získá nastavení, nebo sám nastaví, zdali je odpověď metody uložená do cache. Výchozí nastavení je true. Hodnota false by měla být nastavena pouze v situaci, ve které víte, že splnění požadavku bude trvat dlouho, a že tedy chcete dříve poslat sekci dat.
TransactionOption	Získá nastavení, nebo sám nastaví, zdali metoda podporuje transakci. Pokud ano, tak jakého typu. Povolené hodnoty jsou Disabled, NotSupported, Supported, Required a RequiresNew. Protože webové služby jsou bezstavové, mohou participovat pouze jako kořenový objekt transakce.

CacheDuration

Jak jste se již dozvěděli v kapitole 11, ASP.NET podporuje dva typy cachování – tzv. cachování výstupu (cachování výstupu) a cachování dat (data caching). Jak uvidíte v následujících sekcích, webové služby mohou používat oba tyto způsoby cachování.

Cachování výstupu

Nejjednodušším typem cachování webové služby je cachování výstupu. Cachování výstupu funguje u webové služby stejně jako v případě webových stránek – identické požadavky (v tomto případě požadavky na stejnou metodu se stejnými parametry) získají z cache identické odpovědi (dokud cachované informace nezaniknou). Tímto se může značně zvýšit výkon u stránek s vysokým provozem dokonce i tehdy, když cachujete odpověď pouze na několik vteřin.

Cachování výstupu byste měli používat pouze u funkcí, které jsou určeny pro rychlé získávání informací nebo zpracování dat. Neměli byste je používat u metody, která potřebuje provádět jiné funkce, jako je například změna položek relace, protokolování návštěv webu, či modifikaci databáze. Je to proto, že další volání cachované metody obdrží cachovaný výsledek a kód webové metody nebude vykonán.

Pokud chcete pro nějakou funkci aktivovat cachování, použijte vlastnost CacheDuration atributu WebMethod. Zde uvádíme příklad s metodou GetEmployees():

```
[WebMethod(CacheDuration=30)]
public DataSet GetEmployees()
```

```
{ ... }
```

V tomto příkladě je sada dat se zaměstnanci cachována po dobu 30 sekund. Každý uživatel, který během této doby zavolá metodu `GetProducts()`, obdrží z cache ASP.NET stejnou sadu dat.

Cachování výstupu se stane mnohem zajímavějším, pokud víte, jakým způsobem funguje s metodami, které vyžadují parametry. Zde uvádíme příklad, ve kterém je webová metoda `GetEmployeesByCity()` cachována po dobu deset minut:

```
[WebMethod(CacheDuration=600)]
public DataSet GetEmployeesByCity(string city)
{ ... }
```

V tomto případě je ASP.NET o něco inteligentnější. Opětovně použije pouze ty požadavky, které vrací stejnou hodnotu města. Zde uvádíme příklad, jak se mohou vyvinout tři požadavky webové služby:

1. Klient vyvolá `GetEmployeesByCity()` s parametrem pro Londýn. Webová metoda zavolá a kontaktuje databázi a výsledek uloží do paměti cache webové služby.
2. Klient vyvolá `GetEmployeesByCity()` s parametrem pro Kirkland. Webová metoda zavolá a kontaktuje databázi a výsledek uloží do paměti cache webové služby. Předtím cachovaná sada dat není použita, protože parametr pro město je odlišný.
3. Klient vyvolá `GetEmployeesByCity()` s parametrem pro Londýn. Za předpokladu, že od požadavku z kroku 1 ještě neuplynulo deset minut, ASP.NET automaticky použije první cachovaný výsledek. V tomto případě není vykonán žádný kód.

Podle frekvence používání této webové služby (a podle počtu jednotlivých měst) se rozhodnete, zdali má smysl cachovat tuto verzi `GetEmployeesByCity()`. Pokud existuje několik málo hodnot pro město, tento přístup má určitě smysl. Pokud je měst mnoho (řádově tucty) a paměť vašeho webového serveru je omezená, tento postup bude určitě neefektivní.

Cachování dat

ASP.NET také podporuje cachování dat, které vám umožní ukládat plnohodnotné objekty v cache. Podobně jako u kódu ASP.NET, můžete cachování dat i zde použít prostřednictvím objektu `Cache`, který je dostupný přes vlastnost `Context.Cache` kódu webové služby. Tento objekt může dočasně uchovávat informace, které se nákladným způsobem vytvářejí, takže webová metoda je může opětovně použít při dalších voláních jiných klientů. Data mohou být opětovně použita i jinými webovými službami nebo jinými webovými stránkami stejné aplikace.

Cachování dat má velký význam v případě verze `EmployeesService`, která poskytuje dvě metody `GetEmployees()`, z nichž jedna akceptuje parametr `city`. Abyste zajistili optimální výkon, ale současně omezili množství dat v cache, můžete v ní uchovávat jediný objekt – úplnou sadu dat se zaměstnanci. Jestliže klient poté zavolá verzi `GetEmployees()`, která požaduje parametr `city`, stačí pouze zobrazit řádky pro město, o které klient požádal.

V následujícím kódu si ukážeme, jak tento postup funguje. První krok je vytvoření soukromé metody, která používá cachování, s názvem `GetEmployeesDataSet()`. Pokud je v cache dostupná sada dat, `GetEmployeesDataSet()` použije tuto verzi a obejde databázi. V opačném případě bude vytvořena nová sada dat a naplněna všemi záznamy zaměstnanců. Zde uvádíme kompletní kód:

```
private DataSet GetEmployeesDataSet()
```

```

{
    DataSet ds;
    if (Context.Cache["EmployeesDataSet"] != null)
    {
        // Získej jej z paměti cache.
        ds = (DataSet)Context.Cache["EmployeesDataSet"];
    }
    else
    {
        // Získej jej z databáze.
        string sql = "SELECT EmployeeID, LastName, FirstName, Title, " +
            "TitleOfCourtesy, HomePhone, City FROM Employees";
        SqlConnection con = new SqlConnection(connectionString);
        SqlDataAdapter da = new SqlDataAdapter(sql, con);
        ds = new DataSet();
        da.Fill(ds, "Employees");
        // Trasuj, kdy byl vytvořen DataSet. Tuto informaci můžeš
        // získat u svého klienta a otestovat,
        // zdali funguje cachování.
        ds.ExtendedProperties.Add("CreatedDate", DateTime.Now);
        // Ulož jej do paměti cache na deset minut.
        Context.Cache.Insert("EmployeesDataSet", ds, null,
            DateTime.Now.AddMinutes(10), TimeSpan.Zero);
    }
    return ds;
}

```

Obě metody – `GetEmployees()` a `GetEmployeesByCity()` – mohou používat soukromou metodu `GetEmployeesDataSet()`. Rozdíl je v tom, že `GetEmployeesByCity()` prochází jednotlivé záznamy a ručně odstraňuje všechny záznamy, které neodpovídají dodanému názvu města. Zde uvádíme obě verze:

```

[WebMethod(Description="Returns the full list of employees.")]
public DataSet GetEmployees()
{
    return GetEmployeesDataSet();
}

[WebMethod(Description="Returns the full list of employees by city.")]
public DataSet GetEmployeesByCity(string city)
{
    // Zkopíruj DataSet.
    DataSet dsFiltered = GetEmployeesDataSet().Copy();
    // Ručně odstraň řádky.
    // Toto je dobrý přístup (lepší, než použití
    // metody DataTable.Select(), protože je odolný
    // vůči SQL injection útokům.
    foreach (DataRow row in dsFiltered.Tables[0].Rows)
    {

```

```
// Proved' srovnání nezávislé na velikosti písmen.
if (String.Compare(row["City"].ToString(), city.ToUpper(), true) != 0)
{
    row.Delete();
}
}
// Tyto řádky definitivně odstraň.
dsFiltered.AcceptChanges();
return dsFiltered;
}
```

Většinou byste měli stanovit dobu pro cachování informací podle toho, jak dlouho budou odpovídající data platná. Jestliže byly například získány ceny akcií, použijete mnohem nižší počet sekund než například u předpovědi počasí. Pokud byste ovšem uchovávali informaci, která se málokdy mění, například výsledky ročního sčítání lidu, vaše úvaha by byla zcela odlišná. V tomto případě je platnost informace prakticky trvalá, ovšem množství vrácených informací bude větší, než je kapacita cache ASP.NET. V takové situaci byste měli omezit dobu cachování tak, abyste zajistili, že budou uchovávány pouze nejpůlárnější požadavky.

Samozřejmě – pokud jde o cachování, měli byste se také rozhodovat podle toho, jak dlouho by trvalo opětovně vytvořit informace a kolik klientů bude webovou službu používat. Je pravděpodobné, že budete muset provést v reálných podmínkách pečlivé testování, abyste webovou službu vyladili do naprosté dokonalosti. Více informací o cachování dat můžete najít v kapitole 11.

TIP Cachování dat je globální pro celou aplikaci (na jediném webovém serveru). To znamená, že informace můžete ukládat v cache webové služby a získávat je na webové stránce této webové aplikace.

EnableSession

Pro webové služby ASP.NET je obvyklé deaktivovat stav relace. Webové služby v podstatě stav relace nepodporují. Velké množství webových služeb je totiž navrženo jako bezstavové, čímž se dosahuje značné rozšiřitelnosti. Občas se však můžete rozhodnout použít správu stavu (state management) pro získání informací specifických pro jednotlivé uživatele nebo pro optimalizaci výkonu ve specifických scénářích. V takovém případě musíte použít vlastnost EnableSession, jak vidíte zde:

```
[WebMethod(EnableSession=true)]
public DataSet StatefulMethod()
{ ... }
```

Co se ovšem stane, pokud máte webovou službu, která umožňuje ukládání stavu relace pouze u některých metod? Deaktivace správa stavu nařídí ASP.NET, aby ignorovalo jakékoliv informace v paměti relace a vyloučilo kolekci Session z aktuální procedury. Nezpůsobí vymazání existujících informací z kolekce (to se stane až po vypršení relace). Jediný přínos pro zvýšení výkonu vyplývá z toho, že nemusíte vyhledávat informace relace, když nejsou vyžadovány. Nemusíte podnikat stejné kroky, abyste umožnili vašemu kódu použít stav Application – tato globální kolekce stavu je vždy k dispozici.

Ošetření stavu relace není součástí specifikace SOAP. Výsledkem je, že se musíte spoléhat na podporu výchozí infrastruktury. Pokud jde o podporu stavu relace, ASP.NET se spoléhá na HTTP cookies. Cookie dané

relace ukládá ID relace, které ASP.NET použije pro přiřazení klienta ke stavu relace na serveru. Pokud však používáte stav webové služby, nemáte žádnou záruku, že klient bude podporovat cookies. V praxi je mnoho klientů podporovat nebude. Jestliže klient cookies nepodporuje, správa stavu ASP.NET nebude fungovat, a při každém novém požadavku bude vytvořena nová relace. Bohužel – váš kód nijak nedokáže rozpoznat tento chybový stav.

Chcete-li vyzkoušet stav relace (a prozkoumat potenciální problémy), můžete vytvořit jednoduchou webovou službu, jak je uvedeno zde. Ta ukládá jedinou personalizovanou informaci (jméno uživatele) a umožňuje vám jeho pozdější získání.

```
public class StatefulService : System.Web.Services.WebService
{
    [WebMethod(EnableSession=true)]
    public void StoreName(string name)
    {
        Session["Name"] = name;
    }
    [WebMethod(EnableSession=true)]
    public string GetName()
    {
        if (Session["Name"] == null)
        {
            return "";
        }
        else
        {
            return (string)Session["Name"];
        }
    }
}
```

Při testování webových metod StoreName() a GetName() prostřednictvím testovací stránky ASP.NET získáte očekávané chování. Když vyvoláte GetName(), obdržíte řetězec, který jste dodali při posledním vyvolání StoreName(). Je to proto, že webový prohlížeč podporuje cookies bez problémů.

Třída proxy tuto schopnost standardně nemá. Pokud chcete vidět, jak tento problém vypadá v praxi, přidejte referenci na StatefulService v klientovi Windows. Pak přidejte nové tlačítko s následujícím kódem obsluhy událostí:

```
private void cmdTestState_Click(object sender, System.EventArgs e)
{
    // Vytvoř proxy.
    StatefulService proxy = new StatefulService();
    // Nastav název.
    proxy.StoreName("John Smith");
    // Pokus se získat název.
    MessageBox.Show("You set: " + proxy.GetName());
}
```

Tento kód nebude fungovat tak, jak byste předem očekávali. Když jej spustíte, uvidíte prázdný řetězec, který je zobrazen na obrázku 32-14.



Obrázek 32-14. Selhání stavu služby.

Pokud chcete tento problém vyřešit, musíte explicitně upravit proxy webové služby tak, aby akceptovala cookie relace. To provedete tak, že vytvoříte kontejner cookie (což je instance třídy `System.Net.CookieContainer`).

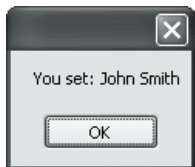
Pro opravu tohoto kódu můžete vytvořit kontejner cookie jako proměnnou na úrovni formuláře, jak vidíte zde. Tím zajistíte, že cookie bude přezívat po stejně dlouhou dobu jako třída (formulář), která jej zaobaluje. Zajistíte také to, že cookie může být opakovaně použita ve více metodách stejného formuláře, aniž by se ztratila aktuální relace webové služby.

```
public partial class Form1 : System.Windows.Forms.Form
{
    private System.Net.CookieContainer cookieContainer =
        new System.Net.CookieContainer();
    ...
}
```

Nyní stačí tento kontejner cookie jednoduše připojit k třídě proxy předtím, než zavoláte jakoukoliv webovou metodu.

```
private void cmdTestState_Click(object sender, System.EventArgs e)
{
    StatefulService proxy = new StatefulService();
    proxy.CookieContainer = cookieContainer;
    proxy.StoreName("John Smith");
    MessageBox.Show("You set: " + proxy.GetName());
}
```

Nyní obě volání webové metody použily stejnou relaci, přičemž jméno uživatele se objeví v okně pro zobrazení zpráv, jak vidíte na obrázku 32-15.



Obrázek 32-15. Úspěšná stavová služba.

Kontejner cookie musíte uchovat tak dlouho, dokud potřebujete uchovat cookie relace. Jestliže je například vašim klientem webová aplikace a vy chcete mít možnost vykonávat operace webové služby po každém odeslání zpět na server, aniž byste ztratili relace, budete muset uložit cookie relace do stavu relace aktuální stránky. Všimněte si, že stav relace webové aplikace se liší od stavu relace webové služby. Nejenom proto, že to jsou samostatné aplikace, ale mnohdy také proto, že jsou obvykle spuštěny na zcela samostatných webových serverech.

Je možné, že si nyní uvědomujete, jak složité je použití relace u webových služeb. Protože webové služby jsou zrušeny po každém zavolání metody, nepředstavují přirozený mechanismus pro uschovávání informací o stavu. Toto omezení můžete kompenzovat použitím kolekce Session, nicméně s tímto přístupem se objeví následující komplikace:

- Stav relace se ztratí po vypršení relace. Klient nemá možnost zjistit, kdy relace vyprší, což znamená, že webová služba se může chovat nepředvídatelně.
- Stav relace je spojen s určitým uživatelem, nikoliv se specifickou třídou či objektem. To může způsobit problémy, pokud se stejný klient snaží použít tutéž webovou službu dvěma různými způsoby, nebo pokud současně vytvoří dvě instance třídy proxy.
- Stav relace je uchováván pouze tehdy, když klient uchovává cookie dané relace. Řízení stavu, které používáte ve webové službě, nebude fungovat, pokud klient neprovede kroky popsane na předchozích stránkách.

Z těchto důvodů není příliš vhodné kombinovat webové služby a správu stavu.

BufferResponse

Vlastnost `BufferResponse` vám umožní řídit, kdy budou data, vrácená z webové služby, zaslána klientovi. Vlastnost `BufferResponse` je standardně nastavena na `true`. To znamená, že všechny výsledky jsou před zasláním klientovi serializovány. Nastavíte-li tuto vlastnost na `false` (jak uvidíte dále), začne ASP.NET vracet výstup tak, jak byl serializován.

```
[WebMethod(BufferResponse=false)]
public byte[] GetLargeStreamOfData()
{ ... }
```

Metoda webové služby ukončí vykonávání vždy dříve, než je cokoliv vráceno. Nastavení `BufferResponse` se vztahuje k serializaci, která se uskuteční po vykonání metody. Pokud je bufferování vypnuto, první část výsledku je serializována a poslána. Poté je serializována a poslána další část výsledku atd.

Nastavení `BufferResponse` na `false` má smysl pouze tehdy, když webová služba vrací velké množství dat. Přesto je však málokdy patrný jakýkoliv rozdíl, protože automaticky generovaná třída `.NET` proxy není schopna spustit postupné zpracovávání jednotlivě vracených dat. To znamená, že třída proxy bude stejně čekat, až získá kompletní informaci, kterou pak zašle zpět vaší aplikaci. Toto chování ovšem můžete změnit tak, že převzmete přímou kontrolu nad zpracováním XML zpráv prostřednictvím rozhraní `IXmlSerializable`, které bude popsáno v další kapitole.

TransactionOption

Webové služby mohou podobně jako jakýkoliv jiný `.NET` kód inicializovat transakce `ADO.NET`. Webové služby mohou také snadno participovat na transakcích `COM+`. Transakce `COM+` jsou zajímavé, protože vám umožňují provádět transakci mezi různými datovými zdroji (například mezi databází `SQL Serveru` a data-

bází Oracle). Transakce COM+ jsou také automaticky potvrzeny (commit) nebo zrušeny (rollback). Za tyto funkce a pohodlí ovšem zaplatíte určitou daň — protože COM+ transakce používají dvoustupňový potvrzovací protokol (commit protocol), jsou vždy pomalejší, než transakce ADO.NET inicializované klientem nebo transakce uložených procedur.

Ve webových službách je podpora transakcí COM+ poněkud omezená. Protože HTTP protokol je bezstavový, metody webové služby se mohou chovat v transakci pouze jako kořenový objekt. To znamená, že metoda webové služby sice může spustit transakci a použít ji pro vykonání série podobných úkolů, nicméně několik webových služeb nemůže být seskupeno do jedné transakce. Výsledkem je, že nad vytvořením transakční webové služby se budete muset více zamyslet. Například nemá smysl vytvořit finanční webovou službu se samostatnými metodami DebitAccount() a CreditAccount(), protože tyto metody nebude možné seskupit do jedné transakce. Místo toho ovšem můžete zajistit, aby byly oba úkoly provedeny společně pomocí transakční metody TransferFunds().

Chcete-li použít ve webové službě transakci, nejdříve musíte přidat referenci k assembly System.EnterpriseServices. Ve Visual Studiu to provedete tak, že v okně Solution Explorer kliknete pravým tlačítkem, vyberete References, pak zvolíte položku Add Reference, a poté položku System.EnterpriseServices. Poté byste měli importovat odpovídající jmenný prostor, abyste měli po ruce typy, které potřebujete (TransactionOption a ContextUtil):

```
using System.EnterpriseServices;
```

Transakci v metodě webové služby spustíte nastavením vlastnosti TransactionOption atributu WebMethod. TransactionOption je výčet, který poskytuje několik hodnot, které vám umožní specifikovat, zdali komponenta kódu používá, nebo vyžaduje transakce. Protože webové služby musí být kořenem transakce, většinu těchto možností nelze použít. Chcete-li vytvořit metodu webové služby, která automaticky spustí transakci, použijte následující atribut:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public DataSet TransactionMethod()
{ ... }
```

Transakce je automaticky potvrzena po ukončení webové metody. Transakce je zrušena, pokud se vyskytne neošetřená výjimka, nebo pokud explicitně nařídíte selhání transakce prostřednictvím následujícího kódu:

```
ContextUtil.SetAbort();
```

Většina databází podporuje transakce COM+. Jakmile takovou databázi použijete v transakční webové metodě, budou automaticky uvedeny v aktuální transakci. Jestliže je transakce zrušena, budou operace, které v databázi provádíte (například přidávání, modifikace či odstraňování záznamů) automaticky zrušeny. Některé operace (jako například zapisování souboru na disk) se však nedědí jako transakční. Znamená to, že pokud transakce selže, tyto operace nebudou zrušeny.

Nyní se zamyslete nad následující webovou metodou, která provádí dvě operace – vymazává záznamy z databáze a poté se pokouší číst ze souboru. Jestliže však souborová operace selže a výjimka není ošetřena, celá transakce bude zrušena a vymazané záznamy budou obnoveny. Zde uvádíme kód transakce:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public void UpdateDatabase()
{
    // Vytvoř ADO.NET objekty.
```

```

SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand("DELETE * FROM Employees", con);
// Aktualizuj. Bude to registrováno jako část transakce.
using (con)
{
    con.Open();
    cmd.ExecuteNonQuery();
}
// Pokus se dostat k souboru. Vygeneruje se tím výjimka, která nebude
// obsloužena.
// Webová metoda a změny budou zrušeny.
FileStream fs = new FileStream("does_not_exist.bin", IO.FileMode.Open);
// (Pokud se nevyskytly žádné chyby, změny databáze
// budou po skončení metody potvrzeny).
}

```

Jiný způsob pro zpracování tohoto kódu je zachycení chyby, provedení požadovaného vyčištění, a poté, podle potřeby, explicitní zrušení transakce:

```

[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public void UpdateDatabase()
{
    // Vytvoř ADO.NET objekty.
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand("DELETE * FROM Employees", con);
    // Aktualizuj.
    try
    {
        con.Open();
        cmd.ExecuteNonQuery();
        FileStream fs = new FileStream("does_not_exist.bin",
            IO.FileMode.Open);
    }
    catch
    {
        if (con.State != ConnectionState.Closed) con.Close();
        ContextUtil.SetAbort();
    }
    finally
    {
        con.Close();
    }
}

```

Potřebuje webová služba používat transakce COM+? Vše záleží na konkrétní situaci. Jsou-li v samostatných datových složkách vyžadovány vícenásobné aktualizace, možná budete muset použít transakce kvůli zajištění integrity vašich dat. Na druhé straně – pokud modifikujete hodnoty pouze v jediné databázi (například v da-

tabázi SQL Server 2000), můžete pravděpodobně použít vestavěné transakční funkce poskytovatele dat, jak bylo popsáno v kapitole 7.

POZNÁMKA V budoucnosti by mohly další standardy, které se objevily, jako například XLANG a WS-Transactions, zaplnit existující mezery definováním mezipatformového standardu, který umožní různým službám participovat na jediné transakci. K realizaci tohoto cíle ovšem zbývá ještě dlouhý kus cesty.

Shrnutí

V této kapitole jste se dozvěděli, co jsou to webové služby a proč jsou tak důležité pro obchodování. Dále jste se seznámili s postupem vytváření a použití webových služeb v .NET, a také s jejich testováním prostřednictvím webového prohlížeče. V následujících dvou kapitolách se podrobněji podíváme na jejich standardy a naučíme se rozšiřovat infrastrukturu webových služeb.