Stručný obsah

Obsah

Kapitola 7

Zabezpečení zdrojového kódu proti výskytu výjimek a jejich ošetřování 203

Kapitola 8

Práce s řetězci 237

Kapitola 17

O autorovi

Trey Nash pracuje jako starší technik v týmu Platforms Global Escalation Services společnosti Microsoft na vývoji operačních systémů Windows a nejrůznějších dalších produktů. Když zrovna horečnatě nepracuje na útrobách operačního systému, vyučuje ladění programů pro platformu .NET a ladění programů pro uživatelský i kernel mód platformy Windows. Ve svém předchozím zaměstnání pracoval jako vedoucí softwarový vývojář bezpečnostních řešení ve společnosti Credant Technologies, čelní společnosti zabývající se vývojem bezpečnostního softwaru. Absolvoval také pracovní stáž ve velké firmě vyvíjející software pro komunikační technologii Bluetooth, kde vyvíjel řešení pro systém Windows Vista. A předtím mu byla po pět let domovem společnost Macromedia Inc., ve které pracoval několik let v týmu zabývajícím se vývojem různých produktů a navrhoval řešení různých problémů pro mnoho aplikací, včetně aplikací pro Flash, Fireworks a Dreamweaver. Až do revolučního nástupu prostředí .NET se specializoval na technologii COM/DCOM a jazyky C/C++ a knihovnu ATL. Počítačům věnuje veškerý svůj čas už od té doby, co jako třináctiletý dostal svůj první počítač TI-99/4A a k úžasu a nelibosti svých rodičů proměnil svoji mladistvou posedlost v dobře placenou práci. Titul bakaláře a inženýra elektrotechniky získal na Texas A&M University. Když zrovna nesedí u počítače, kutí něco v garáži, hraje na klavír, piluje cizí jazyky (aktuálně ruštinu a islandštinu), nebo hraje lední hokej.

Odborný korektor

Damien Foggon je vývojář, autor a odborný korektor knih o nejnovějších technologiích. Podílel se již na více než padesáti knihách o prostředí .NET, programovacích jazycích C# a Visual Basic a prostředí ASP.NET. Je držitelem několika certifikátů MCPD odbornosti v prostředí .NET 2.0 a .NET 3.5. Najdete jej také online na adrese *http://blog.littlepond.co.uk*.

Poděkování

Psaní knihy je dlouhý a vysilující proces, v němž mě velmi podporovala moje rodina a moji přátelé, čehož si velmi vážím. Bez jejich podpory by byl celý tento proces mnohem náročnější a bezesporu také méně plodný.

Rád bych poděkoval zejména následujícím lidem, kteří se podíleli na prvních dvou vydáních této knihy (nejsou uvedeni v žádném konkrétním pořadí): Davidu Wellerovi, Stephenu Toubovi, Rexi Jaeschkeovi, Vladimiru Levinovi, Jerrymu Marescaovi, Chrisi Pelsovi, Christopheru T. McNabbovi, Bradu Wilsonovi, Peteru Partchovi, Paulu Stubbsovi, Rufusi Littlefieldovi, Tomasi Restrepovi, Johnu Lambertovi, Joan Murrayové, Sheri Cainové, Jessice D'Amico, Karen Gettmanové, Jimu Huddlestonovi, Richardu Dal Portovi, Gary Cornellovi, Bradu Abramsovi, Ellie Fountainové, Nicole Abramowitzové, celému týmu nakladatelství Apress a konečně také Shelley Nashové, Michaelu Pulkovi, Shawn Wildermuthové, Sofii Marchantové, Jimu Comptonovi, Dominicu Shakeshaftovi, Wes Dyerové, Kelly Winquistové a Lauře Cheuové.

Za pomoc s vývojem třetího vydání bych pak rád poděkoval následujícím lidem (opět nejsou uvedeni v žádném konkrétním pořadí): Jonathanu Hassellovi, Mary Tobinové, Damienu Foggonovi a Maite Cerverové.

Jestli jsem na někoho zapomněl, je to jenom moje chyba a neměl jsem to v úmyslu. Bez vás bych to nedokázal. Díky vám všem!

Předmluva

Visual C# .NET (C#) je jazyk snadno pochopitelný pro každého, kdo zná jiný objektově orientovaný jazyk. Jazyk C# bude jednoduchý dokonce i pro někoho, kdo zná Visual Basic 6.0 a chce se naučit také nějaký objektově orientovaný jazyk. Avšak i když jazyk C# spolu s aplikačním rámcem .NET Framework umožňují rychlý vývoj jednoduchých aplikací, chcete-li psát v jazyce C# robustní rafinované aplikace, které nejsou náchylné k chybám, stále je nutné, abyste měli velký objem znalostí a věděli, jak jazyk správně používat. V této knize vás naučím vše, co potřebujete vědět, a vysvětlím vám, jak své vědomosti nejlépe využít k tomu, abyste se rychle stali opravdovým odborníkem na jazyk C#.

Idiomy a návrhové vzory jsou při nabývání a aplikaci odborných znalostí nedocenitelné a já vám ukážu, jak mnohé z nich používat k vytváření robustních efektivních aplikací, které jsou odolné vůči chybám a zabezpečené proti výskytu výjimek. Ačkoliv programátoři pracující s jazyky Java a C++ budou mnohé z nich znát, některé jsou jedinečné v aplikačním rámci .NET a jeho běhovém prostředí Common Language Runtime (CLR). Předvedu vám, jak tyto nepostradatelné idiomy a techniky aplikovat při ucelené integraci vašich C# aplikací do runtime prostředí aplikačního rámce .NET s důrazem na nové schopnosti verze jazyka C# 3.0.

Návrhové vzory ukazují nejlepší postupy při navrhování aplikací, které mnoho programátorů objevilo a opět oživilo v průběhu času. Samotný aplikační rámec .NET ve skutečnosti implementuje mnoho velmi dobře známých návrhových vzorů. A za tu dobu, co zde jsou poslední tři verze rámce .NET a poslední dvě verze jazyka C#, se také mnoho nových idiomů a postupů objevilo. V této knize je podrobně rozeberu. Navíc je třeba brát v potaz, že nedocenitelná zásoba těchto vývojových technik se v průběhu času stále vyvíjí.

Od verze jazyka C# 3.0 můžete začít jednoduše využívat techniky funkcionálního programování prostřednictvím lambda-výrazů, rozšiřujících metod a jazyka LINQ. Lambda-výrazy zjednodušují deklaraci a konkretizaci delegátů funkcí v jednom místě. Prostřednictvím lambda-výrazů lze navíc jednoduše vytvářet funkcionály, což jsou funkce, které přijímají jako své vstupní parametry jiné funkce a obvykle vrací další funkci. Dokonce i když jste mohli implementovat funkcionální programovací techniky v jazyce C# již dříve (ačkoliv to nebylo úplně snadné), nové vlastnosti verze 3.0 tvoří prostředí, ve kterém může funkcionální programování vzkvétat v úzkém spojení s typicky imperativním programovacím stylem jazyka C#. Jazyk LINQ vám pak umožňuje prostřednictvím nativní syntaxe jazyka C# vyjadřovat operace datových dotazů (které jsou ve své podstatě typicky funkcionální). Jakmile zjistíte, jak jazyk LINQ funguje, uvědomíte si, že s jeho pomocí toho můžete dělat mnohem více než pouhé dotazy na data a že jej můžete použít k implementaci komplexních funkcionálních programů.

Rámec .NET a CLR představují jedinečné běhové prostředí, které lze použít na různých platformách. C# je pouze jedním z jazyků, které se na toto výkonné běhové prostředí zaměřují, a mnoho technik, které v této knize popisuji, proto lze využít i v jakémkoliv jiném jazyku zaměřeném na vývoj aplikací pro běhové prostředí rámce .NET.

Ti z vás, kteří mají velké zkušenosti s jazykem C++ a znají takové ideje, jako jsou kanonické tvary, zabezpečení vůči výjimkám, získávání prostředků při inicializaci (Resource Acquisition Is Initialization, RAII) a korektnost konstant (const-correctness), v této knize najdou popis způsobu aplikace těchto technik v jazyce C#. Vývojáři pracující v jazycích Java nebo Visual Basic, kteří strávili léta vývojem svých vlastních postupů, zde pak najdou popis způsobu jejich efektivní aplikace v jazyce C#.

Jak uvidíte, abyste se stali profesionály v C#, nemusíte strávit léta pokusy a hledáním chyb. Stačí, když získáte správné vědomosti a naučíte se je odpovídajícím způsobem využívat. Proto jsem pro vás napsal tuto knihu.

O této knize

Předpokládám, že již máte nějaké pracovní zkušenosti s některým z objektově orientovaných programovacích jazyků, například s jazykem C++, Java nebo Visual Basic .NET. Protože je syntaxe jazyka C# odvozena od syntaxe jazyků C++ a Java, nevěnuji se problematice syntaxe příliš podrobně a více do detailu ji rozebírám pouze v případech, ve kterých se od syntaxe jazyků C++ a Java naprosto liší. Pokud již o C# něco víte, můžete kapitoly 1 - 3 pouze prolistovat nebo dokonce úplně přeskočit.

1. kapitola, „Úvod do jazyka C#", je krátkým úvodem, ve kterém rozebírám, jak vypadá aplikace napsaná v jazyce C# a jaké jsou základní rozdíly mezi programovacím prostředím jazyka C# a nativním prostředím jazyka C++.

2. kapitola, „C# a CLR", navazuje na kapitolu první a věnuji se v ní rychlému rozboru spravovaného prostředí, ve kterém C# aplikace běží. V této kapitole dále najdete úvod do problematiky sestavení, základních stavebních kamenů aplikací, do kterých se soubory se zdrojovým kódem jazyka C# kompilují. Nakonec vysvětlím, jak jsou sestavení – díky metadatům – sebepopisná.

3. kapitola, „Přehled syntaxe jazyka C#", mapuje syntaxi jazyka C#. Seznámím vás v ní se dvěma základními druhy typů v CLR: s typy hodnotovými a referenčními. V této kapitole také popisuji jmenné prostory a způsob, jak můžete jejich prostřednictvím logicky organizovat typy a funkcionalitu vašich aplikací.

Kapitoly 4. - 13. obsahují detailní popis způsobu využití užitečných idiomů, návrhových vzorů a nejlepších postupů pro vaše programy a návrhy. Velmi jsem se snažil tyto kapitoly logicky seřadit, ale někdy se v některých z nich používají techniky nebo témata popsané v jedné z kapitol následujících. Tomu se téměř nelze vyhnout, ale snažil jsem se, aby k tomu docházelo co nejméně.

4. kapitola, „Třídy, struktury a objekty", obsahuje detailní informace o vytváření datových typů v jazyce C#. Dozvíte se v ní více o hodnotových a referenčních typech v CLR a také se v ní dotknu nativní podpory rozhraní v prostředí CLR a jazyce C#. Dočtete se, jak v jazyce C# funguje typová dědičnost a že každý objektový typ je odvozen od typu `System.Object`. Tato kapitola také obsahuje množství informací o spravovaném prostředí a o tom, co musíte vědět, abyste definovali typy, které jsou v něm užitečné. Mnohých z těchto témat se pouze dotýkám a později je podrobně rozebírám v kapitolách následujících.

5. kapitola, „Rozhraní a smluvní vztahy", nabízí podrobný rozbor rozhraní a jejich role v jazyce C#. Rozhraní zajišťují funkcionální smluvní vztahy, které se typy mohou rozhodnout implementovat. Ukážu vám nejrůznější způsoby, jakými může typ implementovat rozhraní, a jak se běhové prostředí rozhoduje, kterou metodu zavolá, když dojde k volání metody rozhraní.

6. kapitola, „Přetěžování operátorů“, obsahuje informace o tom, jak můžete přiřadit vaši vlastní funkcionalitu zabudovaným operátorům jazyka C# při jejich aplikaci na vaše vlastní typy. Dozvíte se v ní, jak operátory přetěžovat zodpovědně, protože ne všechny spravované jazyky, které kompilují kód pro CLR, mohou přetížené operátory používat.

7. kapitola, „Zabezpečení zdrojového kódu proti výskytu výjimek a jejich ošetřování“, je věnována popisu schopností ošetřování výjimek jazyka C# a prostředí CLR. Ačkoliv je samotná syntaxe výjimek podobná jejich syntaxi v jazyce C++, psaní zdrojového kódu bezpečného při výjimkách a zdrojového kódu, který se chová při výskytu výjimek neutrálně, je náročné - dokonce ještě náročnější než psaní zdrojového kódu bezpečného při výskytu výjimek v nativním C++. V této kapitole se dovíte, že lze psát zdrojový kód odolný vůči výskytu chyb a zdrojový kód, který se chová při výskytu výjimek neutrálně, aniž byste používali konstrukce příkazů `try`, `catch` nebo `finally`. V této kapitole také rozebírám některé nové schopnosti přidané do běhového modulu prostředí .NET 2.0, které vám umožňují vytvářet zdrojový kód odolný vůči výskytu chyb.

8. kapitola, „Práce s řetězci“, popisuje prvořadou pozici řetězcových typů v CLR a způsob jejich efektivního používání v jazyce C#. Velkou část této kapitoly věnuji možnostem formátování řetězců, které mají nejrůznější typy v .NET, a ukazuji, jak docílit prostřednictvím implementace rozhraní `IFormattable`, aby se vaše vlastní typy chovaly podobně. Seznámím vás také s globalizačními schopnostmi aplikačního rámce a ukážu vám, jak vytvářet vaše vlastní instance typu `CultureInfo` pro kultury a regiony, o kterých rámec .NET zatím neví.

9. kapitola, „Pole, kolekce a iterátory“, je věnovaná popisu nejrůznějších typů polí a kolekcí, které můžete v jazyce C# použít. Můžete vytvářet dva typy vícerozměrných polí i své vlastní typy kolekcí a využít při tom pomocné třídy kolekcí. Ukážu vám, jak definovat dopředné, zpětné a obousměrné iterátory prostřednictvím nové syntaxe iterátorů zavedené ve verzi jazyka C# 2.0., takže vaše typy kolekcí budou dobře fungovat s příkazy `foreach`.

10. kapitola, „Delegáty, anonymní funkce a události“, popisuje mechanismus, který se v jazyce C# používá pro zajištění zpětných volání. Všechny životaschopné aplikační rámce vždy nabízely mechanismus zajišťující možnost implementace zpětných volání. Jazyk C# jde v tomto ohledu ještě dále a zapouzdřuje zpětná volání do objektů, které lze volat, tzv. *delegátů*. Od verze jazyka C# 2.0 navíc můžete vytvářet delegáty prostřednictvím zkrácené syntaxe tzv. *anonymních funkcí*. Anonymní funkce se podobají lambda-funkcím ve funkcionálním programování. Ukážu vám také, jak aplikační rámec delegáty využívá k tomu, aby zajistil notifikační mechanismus událostí založený na návrhovém vzoru pozorovatel (observer; používá se také označení vydavatel-předplatitel, publisher-subscriber), který vám umožňuje oddělit v návrhu zdroj události od jejího spotřebitele.

11. kapitola, „Genericita“, je úvodem do pravděpodobně nejzajímavější nové vlastnosti přidané do verze jazyka C# 2.0 a do CLR. Těm z vás, kteří mají zkušenosti s šablonami v C++, budou genericita povědomá, i když se od šablon v několika ohledech diametrálně liší. Prostřednictvím generických konstrukcí můžete dodat funkcionalitu, ve které se definují specifičtější typy za chodu programu. Generické typy jsou nejužitečnější při použití s typy kolekcí a kolekce jsou díky nim v porovnání se kolekcemi v předchozích verzích prostředí .NET mnohem efektivnější. Počínaje verzí jazyka C# 4.0 je způsob jejich použití ještě intuitivnější díky podpoře kovariance a kontravariance. Přiřazení jednoho generického typu jinému, jestliže dává v rámci typového systému intuitivní smysl, je nyní možné, a díky tomu klesá dřívější potřeba množství konverzních metod.

12. kapitola, „Aplikace s více podprocesy“, rozebírá tvorbu aplikací o více podprocesech (vláknech) ve spravovaném virtuálním prováděcím prostředí jazyka C#. Máte-li zkušenosti s problematikou podprocesů v nativním prostředí Win32, zjistíte, že mezi prostředím Win32 a spravovaným virtuálním běhovým prostředím jazyka C# existují v tomto ohledu signifikantní rozdíly. Spravované prostředí vám navíc nabízí mnohem propracovanější infrastrukturu, která dělení do podprocesů značně zjednodušuje. Dozvíte se, že delegáty zajišťují při použití v souladu s návrhovým vzorem „Dlužník“ (I Owe You - IOU) výbornou přístupovou cestu do fondu vláken. Při současném provádění různých proprocesů je nepochybně nejdůležitější správná synchronizace. V této kapitole proto rozeberu nejrůznější nástroje pro synchronizaci, které mají vaše aplikace k dispozici. Souběžný způsob provádění zdrojového kódu je v současnosti jedním z nejdůležitějších témat, protože hardwarový průmysl se aktuálně místo extrémně časově náročného vývoje rychlejších procesorů zaměřuje na vývoj procesorů s více jádry. Proto ve 12. kapitole rozeberu paralelizační koncepci Parallel Extensions a knihovnu Task Parallel Library (TPL).

13. kapitola, „Hledání kanonických tvarů v C#“, je dizertací na téma nejlepších postupů při definování nových typů tak, aby je bylo možné co nepřirozeněji používat a aby je jejich spotřebitelé nemohli nechtěně zneužívat. Tomuto tématu se letmo věnuji i v jiných kapitolách, v 13. kapitole pak podrobně. Na jejím konci najdete také kontrolní seznam problémů, které byste měli brát při definování nových typů v potaz.

14. kapitola, „Rozšiřující metody“, je věnována této nové vlastnosti jazyka zavedené ve verzi 3.0. Protože můžete rozšiřující metody volat jako metody instancí typu, který rozšiřují, lze jejich prostřednictvím rozšiřovat smluvní vztahy, k jejichž dodržování se typy zavazují. Avšak tím možnosti použití rozšiřujících metod rozhodně nekončí. V této kapitole vám předvedu, jak rozšiřující metody začínají otevírat svět funkcionálního programování v C#.

15. kapitola, „Lambda-výrazy“, se věnuje další nové vlastnosti jazyka C# přidané do jeho třetí verze. Díky lambda-výrazům můžete deklarovat a konkretizovat delegáty prostřednictvím syntaxe, která je velmi stručná a vizuálně deskriptivní. Ačkoliv lze ke stejnému účelu využít i anonymní metody, jejich aplikace vyžaduje mnohem více psaní a není tak syntakticky elegantní. Počínaje verzí jazyka C# 3.0 také můžete lambda-výrazy konvertovat do stromů. Jazyk má tedy zabudovanou schopnost konverze zdrojového kódu do datových struktur. Tato schopnost je užitečná i sama o sobě, ale nesrovnatelně užitečnější ve spojení s jazykem LINQ. Lambda-výrazy ve spojení s rozšiřujícími metodami pak završují implementaci funkcionálního programování do jazyka C#.

16. kapitola, „LINQ: Integrovaný dotazovací jazyk“, je završením rozboru nových vlastností zavedených ve verzi jazyka C# 3.0. Díky LINQ-výrazům vytvořeným prostřednictvím klíčových slov orientovaných na tento zabudovaný dotazovací jazyk můžete konzistentním způsobem integrovat do vašeho zdrojového kódu datové dotazy. LINQ tvoří spojnici mezi typicky imperativním světem programování v jazyce C# a funkcionálním světem programování datových dotazů. LINQ-výrazy můžete použít při manipulaci s běžnými objekty i s daty pocházejícími z SQL databází, Datasetů a XML (mimo jiné).

17. kapitola, „Dynamické typy“, rozebírá nové dynamické typy přidané do C# 4.0. S dynamickými typy je spojená jednodušší integrace s dynamickými jazyky v prostředí .NET, včetně objektů COM Automation. Pryč jsou dny psaní nepřirozeného a obtížně čitelného kódu ve snaze integrovat tyto komponenty - implementace dynamických typů dělá všechnu tuto mechanickou práci za vás. Implementace dynamických typů využívá prostředí Dynamic Language Runtime (DLR), které je také základem pro dynamické jazyky jako IronRuby a IronPython (mimo jiné). A když použijete dynamické typy s typy v DLR, jako je například typ `ExpandoObject`, můžete v jazyce C# vytvářet a implementovat typy, které jsou opravdu dynamické.