O'REILLY®

Concurrency in C# Cookbook

C#并发经典实例（影印版）

東南大學出版社

Stephen Cleary 著

C#并发经典实例（影印版）

Concurrency in C# Cookbook

Stephen Cleary 著

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo O'REILLY®

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目(CIP)数据

C#并发经典实例:英文/(美)克利里(Cleary, S.)著. —影印本. —南京:东南大学出版社,2015.2

书名原文:Concurrency in C# Cookbook

ISBN 978-7-5641-5384-7

Ⅰ.①C… Ⅱ.①克… Ⅲ.①C 语言—程序设计—英文 Ⅳ.①TP312

中国版本图书馆 CIP 数据核字(2014)第 294384 号

江苏省版权局著作权合同登记

图字:10-2014-152 号

C#并发经典实例(影印版)

出版发行:东南大学出版社
地　　址:南京四牌楼 2 号　　邮编:210096
出 版 人:江建中
网　　址:http://www.seupress.com
电子邮件:press@seupress.com
印　　刷:常州市武进第三印刷有限公司
开　　本:787 毫米×980 毫米　16 开本
印　　张:13
字　　数:255 千字
版　　次:2015 年 2 月第 1 版
印　　次:2015 年 2 月第 1 次印刷
书　　号:ISBN 978-7-5641-5384-7
定　　价:42.00 元

本社图书若有印装质量问题,请直接与营销部联系。电话(传真):025-83791830

Praise for *Concurrency in C# Cookbook*

"The next big thing in computing is making massive parallelism accessible to mere mortals. Developers have more power available to us than ever before, but expressing concurrency is still a challenge for many. Stephen turns his attention to this problem, helping us all better understand concurrency, threading, reactive programming models, parallelism, and much more in an easy-to-read but complete reference."

— *Scott Hanselman*
Principal Program Manager, ASP.NET and Azure Web Tools, Microsoft

"The breadth of techniques covered and the cookbook format make this the ideal reference book for modern .NET concurrency."

— *Jon Skeet*
Senior Software Engineer at Google

"Stephen Cleary has established himself as a key expert on asynchrony and parallelism in C#. This book clearly and concisely conveys the most important points and principles developers need to understand to get started and be successful with these technologies."

— *Stephen Toub*
Principal Architect, Microsoft

Preface

I think the animal on this cover, a common palm civet, is applicable to the subject of this book. I knew nothing about this animal until I saw the cover, so I looked it up. Common palm civets are considered pests because they defecate all over ceilings and attics and make loud noises fighting with each other at the most inopportune times. Their anal scent glands emit a nauseating secretion. They have an endangered species rating of "Least Concern," which is apparently the politically correct way of saying, "Kill as many of these as you want; no one will miss them." Common palm civets enjoy eating coffee cherries, and they pass the coffee beans through. Kopi luwak, one of the most expensive coffees in the world, is made from the coffee beans extracted from civet excretions. According to the Specialty Coffee Association of America, "It just tastes bad."

This makes the common palm civet a perfect mascot for concurrent and multithreaded developement. To the uninitiated, concurrency and multithreading are undesirable. They make well-behaved code act up in the most horrendous ways. Race conditions and whatnot cause loud crashes (always, it seems, either in production or a demo). Some have gone so far as to declare "threads are evil" and avoid concurrency completely. There are a handful of developers who have developed a taste for concurrency and use it without fear; but most developers have been burned in the past by concurrency, and that experience has left a bad taste in their mouth.

However, for modern applications, concurrency is quickly becoming a requirement. Users these days expect fully responsive interfaces, and server applications are having to scale to unprecedented levels. Concurrency addresses both of these trends.

Fortunately, there are many modern libraries that make concurrency *much* easier! Parallel processing and asynchronous programming are no longer exclusively the domains of wizards. By raising the level of abstraction, these libraries make responsive and scalable application development a realistic goal for every developer. If you have been burned in the past when concurrency was extremely difficult, then I encourage you to give it another try with modern tools. We can probably never call concurrency easy, but it sure isn't as hard as it used to be!

Who Should Read This Book

This book is written for developers who want to learn modern approaches to concurrency. I do assume that you've got a fair amount of .NET experience, including an understanding of generic collections, enumerables, and LINQ. I do *not* expect that you have any multithreading or asynchronous programming knowledge. If you do have some experience in those areas, you may still find this book helpful because it introduces newer libraries that are safer and easier to use.

Concurrency is useful for any kind of application. It doesn't matter whether you work on desktop, mobile, or server applications; these days concurrency is practically a requirement across the board. You can use the recipes in this book to make user interfaces more responsive and servers more scalable. We are already at the point where concurrency is ubiquitous, and understanding these techniques and their uses is essential knowledge for the professional developer.

Why I Wrote This Book

Early in my career, I learned multithreading the hard way. After a couple of years, I learned asynchronous programming the hard way. While those were both valuable experiences, I do wish that back then I had some of the tools and resources that are available today. In particular, the `async` and `await` support in modern .NET languages is pure gold.

However, if you look around today at books and other resources for learning concurrency, they almost all start by introducing the most low-level concepts. There's excellent coverage of threads and serialization primitives, and the higher-level techniques are put off until later, if they're covered at all. I believe this is for two reasons. First, many developers of concurrency such as myself did learn the low-level concepts first, slogging through the old-school techniques. Second, many books are years old and cover now-outdated techniques; as the newer techniques have become available, these books have been updated to include them, but unfortunately placed them at the end.

I think that's backward. In fact, this book *only* covers modern approaches to concurrency. That's not to say there's no value in understanding all the low-level concepts. When I went to college for programming, I had one class where I had to build a virtual CPU from a handful of gates, and another class that covered assembly programming. In my professional career, I've never designed a CPU, and I've only written a couple dozen lines of assembly, but my understanding of the fundamentals still helps me every day. However, it's best to start with the higher-level abstractions; my first programming class was not in assembly language.

This book fills a niche: it is an introduction to (and reference for) concurrency using modern approaches. It covers several different kinds of concurreny, including parallel, asynchronous, and reactive programming. However, it does not cover any of the old-school techniques, which are adequately covered in many other books and online resources.

Navigating This Book

This book is intended as both an introduction and as a quick reference for common solutions. The book is broken down as follows:

- Chapter 1 is an introduction to the various kinds of concurrency covered by this book: parallel, asynchronous, reactive, and dataflow.
- Chapters 2-5 are a more thorough introduction to these kinds of concurrency.
- The remaining chapters each deal with a particular aspect of concurrency, and act as a reference for solutions to common problems.

I recommend reading (or at least skimming) the first chapter, even if you're already familiar with some kinds of concurrency.

Online Resources

This book acts like a broad-spectrum introduction to several different kinds of concurrency. I've done my best to include techniques that I and others have found the most helpful, but this book is not exhaustive by any means. The following resources are the best ones I've found for a more thorough exploration of these technologies.

For parallel programming, the best resource I know of is *Parallel Programming with Microsoft .NET* by Microsoft Press, which is available online (*http://bit.ly/parallel-prog*). Unfortunately, it is already a bit out of date. The section on Futures should use asynchronous code instead, and the section on Pipelines should use TPL Dataflow.

For asynchronous programming, MSDN is quite good, particularly the "Task-based Asynchronous Pattern" (*http://bit.ly/micro-TAP*) document.

Microsoft has also published an "Introduction to TPL Dataflow," (*http://bit.ly/intro-tpl*) which is the best description of TPL Dataflow.

Reactive Extensions (Rx) is a library that is gaining a lot of traction online and continues evolving. In my opinion, the best resource today for Rx is an ebook by Lee Campbell called *Introduction to Rx* (*http://www.introtorx.com/*).

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip, suggestion, or general note.

This element indicates a warning or caution.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/concur-c-ckbk.*

To comment or ask technical questions about this book, send email to *bookques tions@oreilly.com.*

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com.*

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

This book simply would not exist without the help of so many people!

First and foremost, I'd like to acknowledge my Lord and Savior Jesus Christ. Becoming a Christian was the most important decision of my life! If you want more information on this subject, feel free to contact me via my personal web page (*http://stephencleary.com/*).

Second, I thank my family for allowing me to give up so much time with them. When I started writing, I had some author friends of mine tell me, "Say goodbye to your family for the next year!" and I thought they were joking. My wife, Mandy, and our children, SD and Emma, have been very understanding while I put in long days at work followed by writing on evenings and weekends. Thank you so much. I love you!

Of course, this book would not be nearly as good as it is without my editor, Brian MacDonald, and our technical reviewers: Stephen Toub, Petr Onderka ("svick"), and Nick Paldino ("casperOne"). So if any mistakes get through, it's totally their fault. Just kidding! Their input has been invaluable in shaping (and fixing) the content, and any remaining mistakes are of course my own.

Finally, I'd like to thank some of the people I've learned these techniques from: Stephen Toub, Lucian Wischik, Thomas Levesque, Lee Campbell, the members of Stack Overflow and the MSDN Forums, and the attendees of the software conferences in and around my home state of Michigan. I appreciate being a part of the software development community, and if this book adds any value, it is only because of so many who have already shown the way. Thank you all!

Table of Contents

Preface. vii

1. Concurrency: An Overview. 1
   1.1. Introduction to Concurrency 1
   1.2. Introduction to Asynchronous Programming 3
   1.3. Introduction to Parallel Programming 7
   1.4. Introduction to Reactive Programming (Rx) 10
   1.5. Introduction to Dataflows 12
   1.6. Introduction to Multithreaded Programming 14
   1.7. Collections for Concurrent Applications 15
   1.8. Modern Design 15
   1.9. Summary of Key Technologies 16

2. Async Basics. 19
   2.1. Pausing for a Period of Time 20
   2.2. Returning Completed Tasks 22
   2.3. Reporting Progress 23
   2.4. Waiting for a Set of Tasks to Complete 24
   2.5. Waiting for Any Task to Complete 27
   2.6. Processing Tasks as They Complete 28
   2.7. Avoiding Context for Continuations 32
   2.8. Handling Exceptions from async Task Methods 33
   2.9. Handling Exceptions from async Void Methods 34

3. Parallel Basics. 37
   3.1. Parallel Processing of Data 37
   3.2. Parallel Aggregation 39
   3.3. Parallel Invocation 41
   3.4. Dynamic Parallelism 42

3.5. Parallel LINQ 44

4. **Dataflow Basics** 47
4.1. Linking Blocks 48
4.2. Propagating Errors 49
4.3. Unlinking Blocks 51
4.4. Throttling Blocks 52
4.5. Parallel Processing with Dataflow Blocks 53
4.6. Creating Custom Blocks 54

5. **Rx Basics** 57
5.1. Converting .NET Events 58
5.2. Sending Notifications to a Context 60
5.3. Grouping Event Data with Windows and Buffers 62
5.4. Taming Event Streams with Throttling and Sampling 64
5.5. Timeouts 66

6. **Testing** 69
6.1. Unit Testing async Methods 70
6.2. Unit Testing async Methods Expected to Fail 71
6.3. Unit Testing async void Methods 73
6.4. Unit Testing Dataflow Meshes 74
6.5. Unit Testing Rx Observables 76
6.6. Unit Testing Rx Observables with Faked Scheduling 78

7. **Interop** 83
7.1. Async Wrappers for "Async" Methods with "Completed" Events 83
7.2. Async Wrappers for "Begin/End" methods 85
7.3. Async Wrappers for Anything 86
7.4. Async Wrappers for Parallel Code 88
7.5. Async Wrappers for Rx Observables 89
7.6. Rx Observable Wrappers for async Code 90
7.7. Rx Observables and Dataflow Meshes 92

8. **Collections** 95
8.1. Immutable Stacks and Queues 98
8.2. Immutable Lists 100
8.3. Immutable Sets 102
8.4. Immutable Dictionaries 104
8.5. Threadsafe Dictionaries 106
8.6. Blocking Queues 108
8.7. Blocking Stacks and Bags 110
8.8. Asynchronous Queues 112

8.9. Asynchronous Stacks and Bags 115
8.10. Blocking/Asynchronous Queues 117

9. Cancellation. 121
9.1. Issuing Cancellation Requests 122
9.2. Responding to Cancellation Requests by Polling 125
9.3. Canceling Due to Timeouts 126
9.4. Canceling async Code 127
9.5. Canceling Parallel Code 128
9.6. Canceling Reactive Code 130
9.7. Canceling Dataflow Meshes 132
9.8. Injecting Cancellation Requests 133
9.9. Interop with Other Cancellation Systems 134

10. Functional-Friendly OOP. 137
10.1. Async Interfaces and Inheritance 137
10.2. Async Construction: Factories 139
10.3. Async Construction: The Asynchronous Initialization Pattern 141
10.4. Async Properties 144
10.5. Async Events 147
10.6. Async Disposal 150

11. Synchronization. 155
11.1. Blocking Locks 160
11.2. Async Locks 162
11.3. Blocking Signals 164
11.4. Async Signals 165
11.5. Throttling 167

12. Scheduling. 169
12.1. Scheduling Work to the Thread Pool 169
12.2. Executing Code with a Task Scheduler 171
12.3. Scheduling Parallel Code 173
12.4. Dataflow Synchronization Using Schedulers 174

13. Scenarios. 175
13.1. Initializing Shared Resources 175
13.2. Rx Deferred Evaluation 177
13.3. Asynchronous Data Binding 178
13.4. Implicit State 180

Index. 183

CHAPTER 1

Concurrency: An Overview

Concurrency is a key aspect of beautiful software. For decades, concurrency was possible but difficult. Concurrent software was difficult to write, difficult to debug, and difficult to maintain. As a result, many developers chose the easier path and avoided concurrency. However, with the libraries and language features available for modern .NET programs, concurrency is much easier. When Visual Studio 2012 was released, Microsoft significantly lowered the bar for concurrency. Previously, concurrent programming was the domain of experts; these days, every developer can (and should) embrace concurrency.

1.1. Introduction to Concurrency

Before continuing, I'd like to clear up some terminology that I'll be using throughout this book. Let's start with *concurrency*.

Concurrency
    Doing more than one thing at a time.

I hope it's obvious how concurrency is helpful. End-user applications use concurrency to respond to user input *while* writing to a database. Server applications use concurrency to respond to a second request *while* finishing the first request. You need concurrency any time you need an application to do one thing *while* it's working on something else. Almost every software application in the world can benefit from concurrency.

At the time of this writing (2014), most developers hearing the term "concurrency" immediately think of "multithreading." I'd like to draw a distinction between these two.

Multithreading
    A form of concurrency that uses multiple threads of execution.

Multithreading literally refers to using multiple threads. As we'll see in many recipes in this book, multithreading is *one* form of concurrency, but certainly not the only one. In fact, direct use of the low-level threading types has almost no purpose in a modern application; higher-level abstractions are more powerful and more efficient than old-school multithreading. As a consequence, I'll minimize my coverage of outdated techniques in this book. None of the multithreading recipes in this book use the `Thread` or `BackgroundWorker` types; they have been replaced with superior alternatives.

> As soon as you type `new Thread()`, it's over; your project already has legacy code.

But don't get the idea that multithreading is dead! Multithreading lives on in the *thread pool*, a useful place to queue work that automatically adjusts itself according to demand. In turn, the thread pool enables another important form of concurrency: *parallel processing*.

Parallel Processing
: Doing lots of work by dividing it up among multiple threads that run concurrently.

Parallel processing (or parallel programming) uses multithreading to maximize the use of multiple processors. Modern CPUs have multiple cores, and if there's a lot of work to do, then it makes no sense to just make one core do all the work while the others sit idle. Parallel processing will split up the work among multiple threads, which can each run independently on a different core.

Parallel processing is one type of multithreading, and multithreading is one type of concurrency. There's another type of concurrency that is important in modern applications but is not (currently) familiar to many developers: *asynchronous programming*.

Asynchronous Programming
: A form of concurrency that uses futures or callbacks to avoid unnecessary threads.

A *future* (or *promise*) is a type that represents some operation that will complete in the future. The modern future types in .NET are `Task` and `Task<TResult>`. Older asynchronous APIs use callbacks or events instead of futures. Asynchronous programming is centered around the idea of an *asynchronous operation*: some operation that is started that will complete some time later. While the operation is in progress, it does not block the original thread; the thread that starts the operation is free to do other work. When the operation completes, it notifies its future or invokes its completion callback event to let the application know the operation is finished.

Asynchronous programming is a powerful form of concurrency, but until recently, it required extremely complex code. The `async` and `await` support in VS2012 make asynchronous programming almost as easy as synchronous (nonconcurrent) programming.