正则表达式入门（影印版）

Introducing

Regular Expressions

O'REILLY®

東南大學出版社

Michael Fitzgerald 著

Introducing

Regular Expressions

O'REILLY

正则表达式入门（影印版）

Introducing Regular Expressions

Michael Fitzgerald

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

正则表达式入门：英文/(美)菲茨杰拉德（Fitzgerald, M.）著.—影印本.—南京：东南大学出版社，2013.1

书名原文：Introducing Regular Expression

ISBN 978-7-5641-3891-2

I. ①正… II. ①菲… III. ①正则表达式－英文 IV. ①TP301.2

中国版本图书馆CIP数据核字（2012）第273548号

江苏省版权局著作权合同登记

图字：10-2012-163号

正则表达式入门（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：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：9.5
字　　数：186千字
版　　次：2013年1月第1版
印　　次：2013年1月第1次印刷
书　　号：ISBN 978-7-5641-3891-2
定　　价：38.00元（册）

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

Table of Contents

CHAPTER 1
What Is a Regular Expression?

Regular expressions are specially encoded text strings used as patterns for matching sets of strings. They began to emerge in the 1940s as a way to describe regular languages, but they really began to show up in the programming world during the 1970s. The first place I could find them showing up was in the QED text editor written by Ken Thompson.

> "A regular expression is a pattern which specifies a set of strings of characters; it is said to match certain strings." —Ken Thompson

Regular expressions later became an important part of the tool suite that emerged from the Unix operating system—the *ed*, *sed* and *vi* (*vim*) editors, *grep*, *AWK*, among others. But the ways in which regular expressions were implemented were not always so regular.

This book takes an inductive approach; in other words, it moves from the specific to the general. So rather than an example after a treatise, you will often get the example first and then a short treatise following that. It's a learn-by-doing book.

Regular expressions have a reputation for being gnarly, but that all depends on how you approach them. There is a natural progression from something as simple as this:

```
\d
```

a *character shorthand* that matches any digit from 0 to 9, to something a bit more complicated, like:

```
^(\(\d{3}\)|^\d{3}[.-]?)?\d{3}[.-]?\d{4}$
```

which is where we'll wind up at the end of this chapter: a fairly robust regular expression that matches a 10-digit, North American telephone number, with or without parentheses around the area code, or with or without hyphens or dots (periods) to separate the numbers. (The parentheses must be balanced, too; in other words, you can't just have one.)

Chapter 10 shows you a slightly more sophisticated regular expression for a phone number, but the one above is sufficient for the purposes of this chapter.

If you don't get how that all works yet, don't worry: I'll explain the whole expression a little at a time in this chapter. If you will just follow the examples (and those throughout the book, for that matter), writing regular expressions will soon become second nature to you. Ready to find out for yourself?

I at times represent Unicode characters in this book using their code point—a four-digit, hexadecimal (base 16) number. These code points are shown in the form *U+0000*. U+002E, for example, represents the code point for a full stop or period (.).

Getting Started with Regexpal

First let me introduce you to the Regexpal website at *http://www.regexpal.com*. Open the site up in a browser, such as Google Chrome or Mozilla Firefox. You can see what the site looks like in Figure 1-1.

You can see that there is a text area near the top, and a larger text area below that. The top text box is for entering regular expressions, and the bottom one holds the subject or target text. The target text is the text or set of strings that you want to match.

At the end of this chapter and each following chapter, you'll find a "Technical Notes" section. These notes provide additional information about the technology discussed in the chapter and tell you where to get more information about that technology. Placing these notes at the end of the chapters helps keep the flow of the main text moving forward rather than stopping to discuss each detail along the way.

Matching a North American Phone Number

Now we'll match a North American phone number with a regular expression. Type the phone number shown here into the lower section of Regexpal:

```
707-827-7019
```

Do you recognize it? It's the number for O'Reilly Media.

Let's match that number with a regular expression. There are lots of ways to do this, but to start out, simply enter the number itself in the upper section, exactly as it is written in the lower section (hold on now, don't sigh):

```
707-827-7019
```

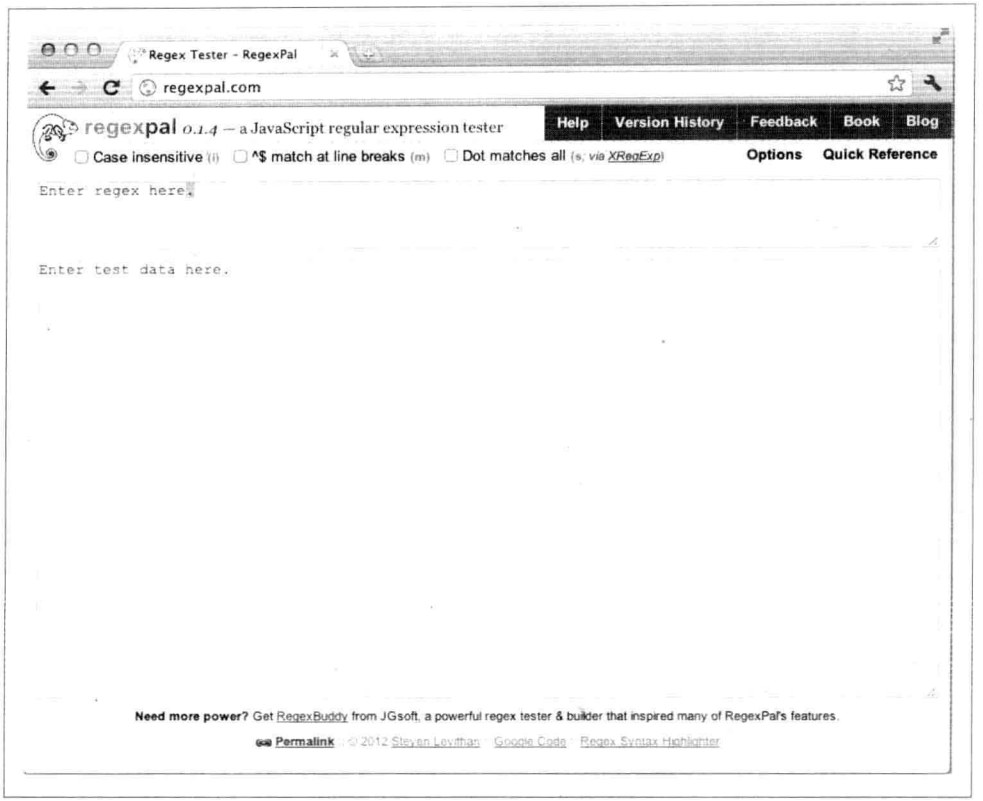

Figure 1-1. Regexpal in the Google Chrome browser

What you should see is the phone number you entered in the lower box highlighted from beginning to end in yellow. If that is what you see (as shown in Figure 1-2), then you are in business.

When I mention colors in this book, in relation to something you might see in an image or a screenshot, such as the highlighting in Regexpal, those colors may appear online and in e-book versions of this book, but, alas, not in print. So if you are reading this book on paper, then when I mention a color, your world will be grayscale, with my apologies.

What you have done in this regular expression is use something called a *string literal* to match a string in the target text. A string literal is a literal representation of a string.

Now delete the number in the upper box and replace it with just the number 7. Did you see what happened? Now only the sevens are highlighted. The literal character (number) 7 in the regular expression matches the four instances of the number 7 in the text you are matching.

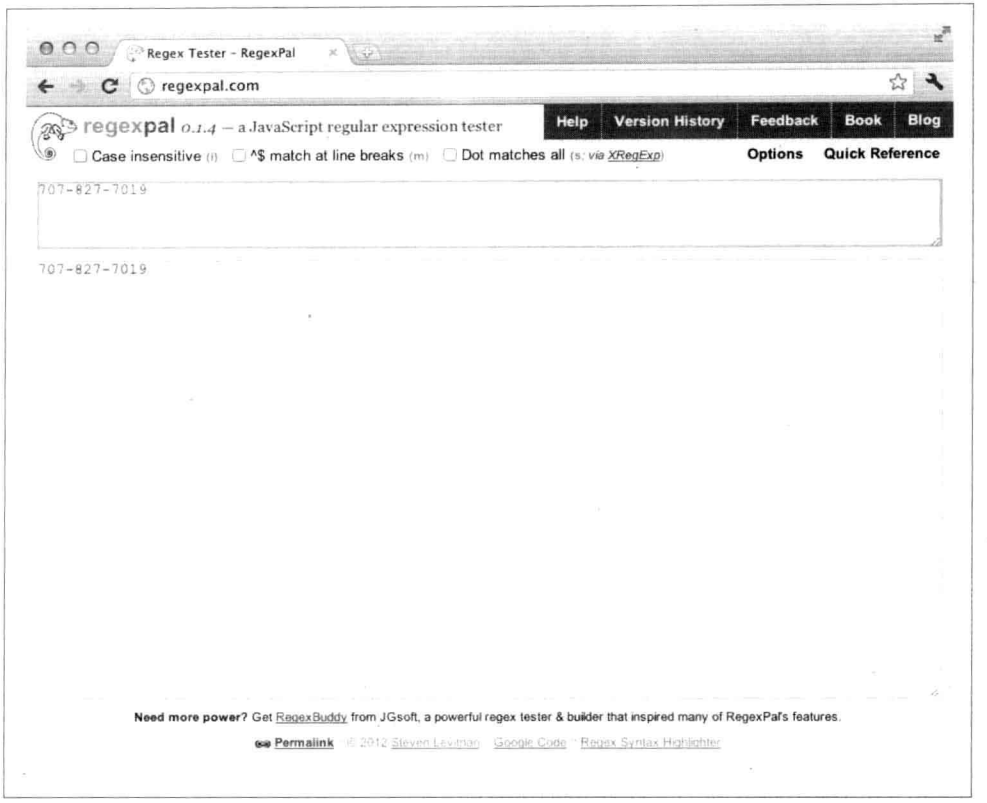

Figure 1-2. Ten-digit phone number highlighted in Regexpal

Matching Digits with a Character Class

What if you wanted to match all the numbers in the phone number, all at once? Or match any number for that matter?

Try the following, exactly as shown, once again in the upper text box:

```
[0-9]
```

All the numbers (more precisely *digits*) in the lower section are highlighted, in alternating yellow and blue. What the regular expression `[0-9]` is saying to the regex processor is, "Match any digit you find in the range 0 through 9."

The square brackets are not literally matched because they are treated specially as *metacharacters*. A metacharacter has special meaning in regular expressions and is reserved. A regular expression in the form `[0-9]` is called a *character class*, or sometimes a *character set*.

You can limit the range of digits more precisely and get the same result using a more specific list of digits to match, such as the following:

```
[012789]
```

This will match only those digits listed, that is, 0, 1, 2, 7, 8, and 9. Try it in the upper box. Once again, every digit in the lower box will be highlighted in alternating colors.

To match any 10-digit, North American phone number, whose parts are separated by hyphens, you could do the following:

```
[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
```

This will work, but it's bombastic. There is a better way with something called a shorthand.

Using a Character Shorthand

Yet another way to match digits, which you saw at the beginning of the chapter, is with `\d` which, by itself, will match all Arabic digits, just like `[0-9]`. Try that in the top section and, as with the previous regular expressions, the digits below will be highlighted. This kind of regular expression is called a *character shorthand*. (It is also called a *character escape*, but this term can be a little misleading, so I avoid it. I'll explain later.)

To match any digit in the phone number, you could also do this:

```
\d\d\d-\d\d\d-\d\d\d\d
```

Repeating the `\d` three and four times in sequence will exactly match three and four digits in sequence. The hyphen in the above regular expression is entered as a literal character and will be matched as such.

What about those hyphens? How do you match them? You can use a literal hyphen (-) as already shown, or you could use an escaped uppercase *D* (`\D`), which matches any character that is *not* a digit.

This sample uses `\D` in place of the literal hyphen.

```
\d\d\d\D\d\d\d\D\d\d\d\d
```

Once again, the entire phone number, including the hyphens, should be highlighted this time.

Matching Any Character

You could also match those pesky hyphens with a dot (.):

```
\d\d\d.\d\d\d.\d\d\d\d
```

The dot or period essentially acts as a wildcard and will match any character (except, in certain situations, a line ending). In the example above, the regular expression matches the hyphen, but it could also match a percent sign (%):

```
707%827%7019
```

Or a vertical bar (|):

```
707|827|7019
```

Or any other character.

As I mentioned, the dot character (officially, the full stop) will not normally match a new line character, such as a line feed (U+000A). However, there are ways to make it possible to match a newline with a dot, which I will show you later. This is often called the *dotall* option.

Capturing Groups and Back References

You'll now match just a portion of the phone number using what is known as a *capturing group*. Then you'll refer to the content of the group with a *backreference*. To create a capturing group, enclose a `\d` in a pair of parentheses to place it in a group, and then follow it with a `\1` to backreference what was captured:

```
(\d)\d\1
```

The `\1` refers back to what was captured in the group enclosed by parentheses. As a result, this regular expression matches the prefix `707`. Here is a breakdown of it:

- `(\d)` matches the first digit and captures it (the number *7*)
- `\d` matches the next digit (the number *0*) but does not capture it because it is not enclosed in parentheses
- `\1` references the captured digit (the number *7*)

This will match only the area code. Don't worry if you don't fully understand this right now. You'll see plenty of examples of groups later in the book.

You could now match the whole phone number with one group and several backreferences:

```
(\d)0\1\D\d\d\1\D\1\d\d\d
```

But that's not quite as elegant as it could be. Let's try something that works even better.

Using Quantifiers

Here is yet another way to match a phone number using a different syntax:

```
\d{3}-?\d{3}-?\d{4}
```

The numbers in the curly braces tell the regex processor *exactly* how many occurrences of those digits you want it to look for. The braces with numbers are a kind of *quantifier*. The braces themselves are considered metacharacters.

The question mark (?) is another kind of quantifier. It follows the hyphen in the regular expression above and means that the hyphen is optional—that is, that there can be zero or one occurrence of the hyphen (one or none). There are other quantifiers such as the plus sign (+), which means "one or more," or the asterisk (*) which means "zero or more."

Using quantifiers, you can make a regular expression even more concise:

```
(\d{3,4}[.-]?)+
```

The plus sign again means that the quantity can occur one or more times. This regular expression will match either three or four digits, followed by an optional hyphen or dot, grouped together by parentheses, one or more times (+).

Is your head spinning? I hope not. Here's a character-by-character analysis of the regular expression above:

- (open a capturing group
- \ start character shorthand (escape the following character)
- d end character shorthand (match any digit in the range 0 through 9 with \d)
- { open quantifier
- 3 minimum quantity to match
- , separate quantities
- 4 maximum quantity to match
- } close quantifier
- [open character class
- . dot or period (matches literal dot)
- - literal character to match hyphen
-] close character class
- ? zero or one quantifier
-) close capturing group
- + one or more quantifier

This all works, but it's not quite right because it will also match other groups of 3 or 4 digits, whether in the form of a phone number or not. Yes, we learn from our mistakes better than our successes.

So let's improve it a little:

```
(\d{3}[.-]?){2}\d{4}
```

This will match two nonparenthesized sequences of three digits each, followed by an optional hyphen, and then followed by exactly four digits.

Quoting Literals

Finally, here is a regular expression that allows literal parentheses to optionally wrap the first sequence of three digits, and makes the area code optional as well:

```
^(\(\d{3}\)|^\d{3}[.-]?)?\d{3}[.-]?\d{4}$
```

To ensure that it is easy to decipher, I'll look at this one character by character, too:

- `^` (caret) at the beginning of the regular expression, or following the vertical bar (`|`), means that the phone number will be at the beginning of a line.
- `(` opens a capturing group.
- `\(` is a literal open parenthesis.
- `\d` matches a digit.
- `{3}` is a quantifier that, following `\d`, matches exactly three digits.
- `\)` matches a literal close parenthesis.
- `|` (the vertical bar) indicates *alternation*, that is, a given choice of alternatives. In other words, this says "match an area code with parentheses or without them."
- `^` matches the beginning of a line.
- `\d` matches a digit.
- `{3}` is a quantifier that matches exactly three digits.
- `[.-]?` matches an optional dot or hyphen.
- `)` close capturing group.
- `?` make the group optional, that is, the prefix in the group is not required.
- `\d` matches a digit.
- `{3}` matches exactly three digits.
- `[.-]?` matches another optional dot or hyphen.
- `\d` matches a digit.
- `{4}` matches exactly four digits.
- `$` matches the end of a line.

This final regular expression matches a 10-digit, North American telephone number, with or without parentheses, hyphens, or dots. Try different forms of the number to see what will match (and what won't).

The capturing group in the above regular expression is not necessary. The group is necessary, but the capturing part is not. There is a better way to do this: a non-capturing group. When we revisit this regular expression in the last chapter of the book, you'll understand why.

A Sample of Applications

To conclude this chapter, I'll show you the regular expression for a phone number in several applications.

TextMate is an editor that is available only on the Mac and uses the same regular expression library as the Ruby programming language. You can use regular expressions through the Find (search) feature, as shown in Figure 1-3. Check the box next to *Regular expression*.

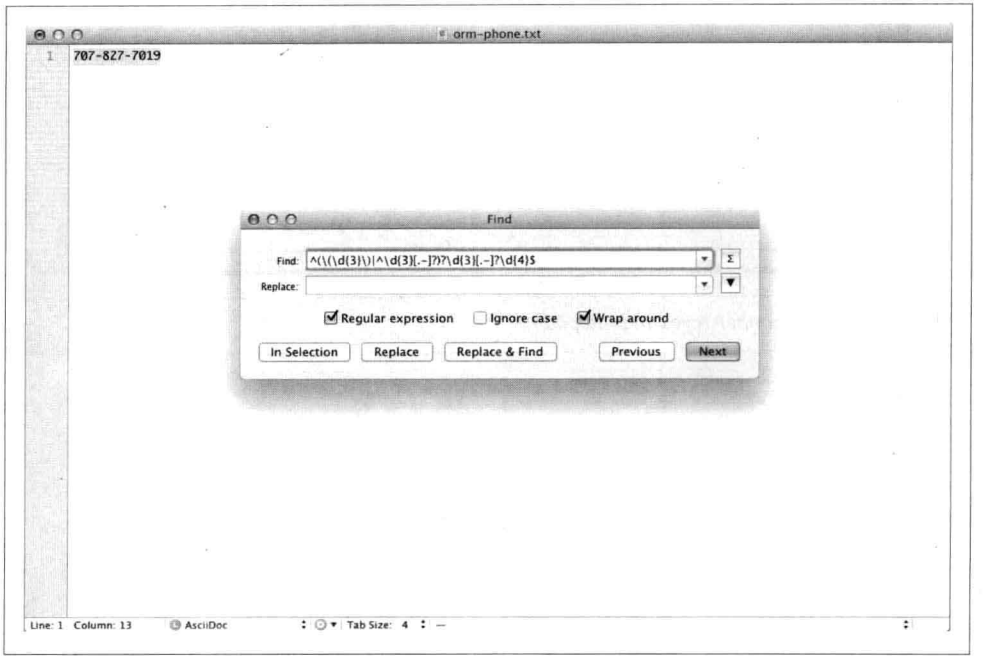

Figure 1-3. Phone number regex in TextMate

Notepad++ is available on Windows and is a popular, free editor that uses the PCRE regular expression library. You can access them through search and replace (Figure 1-4) by clicking the radio button next to *Regular expression*.

Oxygen is also a popular and powerful XML editor that uses Perl 5 regular expression syntax. You can access regular expressions through the search and replace dialog, as shown in Figure 1-5, or through its regular expression builder for XML Schema. To use regular expressions with Find/Replace, check the box next to *Regular expression*.

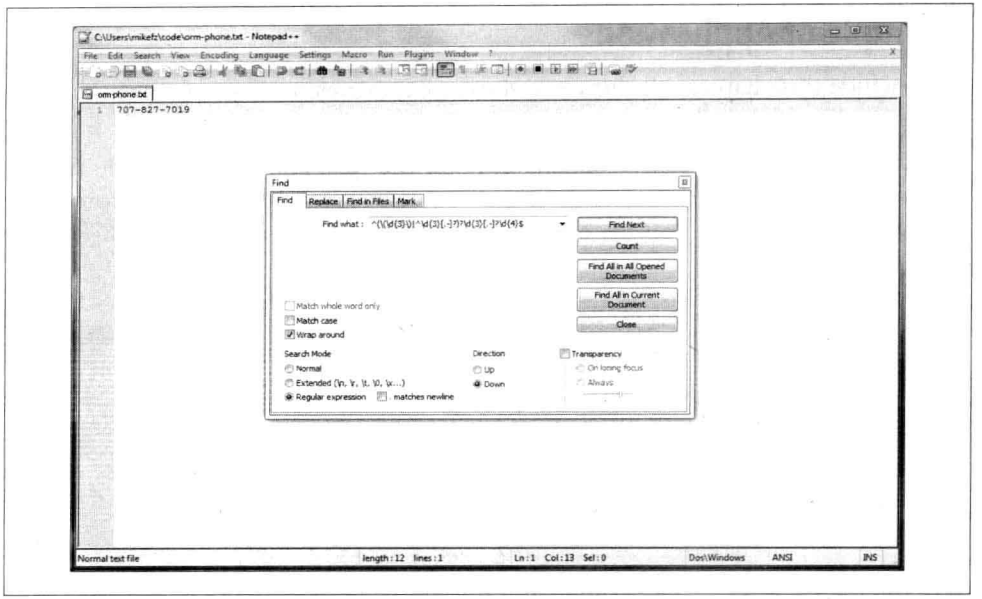

Figure 1-4. Phone number regex in Notepad++

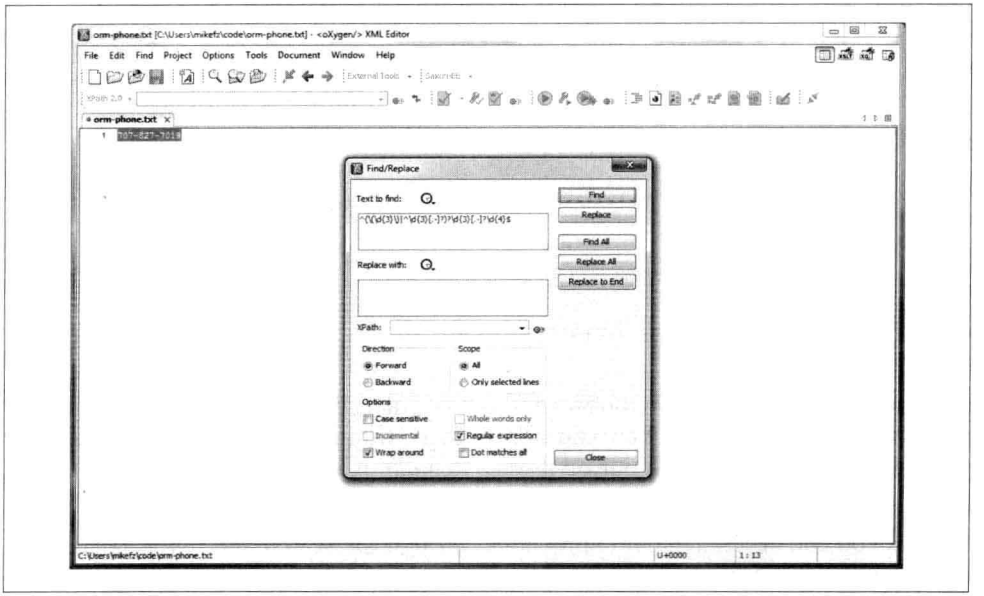

Figure 1-5. Phone number regex in Oxygen

This is where the introduction ends. Congratulations. You've covered a lot of ground in this chapter. In the next chapter, we'll focus on simple pattern matching.