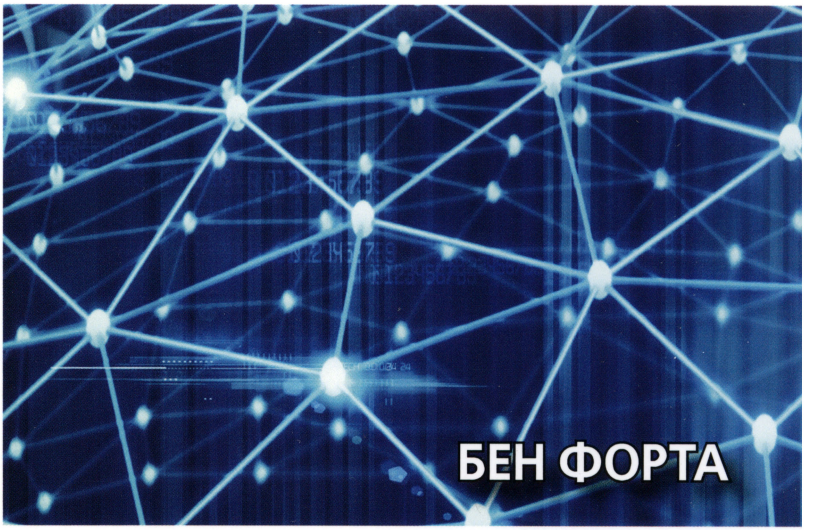
ИЗУЧАЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ



БЕН ФОРТА

ИЗУЧАЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

LEARNING REGULAR EXPRESSIONS

BEN FORTA

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam
Cape Town • Dubai • London • Madrid • Milan • Munich • Paris
Montreal • Toronto • Delhi • Mexico City • São Paulo • Sidney
Hong Kong • Seoul • Singapore • Taipei • Tokyo

ИЗУЧАЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

БЕН ФОРТА

Диалектика

Москва · Санкт-Петербург
2019

ББК 32.973.26-018.2.75
Ф80
УДК 681.3.07

ООО "Диалектика"

Перевод с английского и редакция *И.В. Берштейна*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Форта, Бен.

Ф80 Изучаем регулярные выражения. : Пер. с англ. — СПб. : ООО "Диалектика", 2019. — 192 с. : ил. — Парал. тит. англ.

ISBN 978-5-6041394-2-4 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Бен Форта

Изучаем регулярные выражения

Подписано в печать 17.10.2018. Формат 84x108/32.
Гарнитура Minion Pro.
Усл. печ. л. 6,0. Уч.-изд. л. 5,5.
Тираж 500 экз. Заказ № 9994.

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург,
Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-6041394-2-4 (рус.)
ISBN 978-0-13-475706-3 (англ.)

Оглавление

Содержание

Введение

Регулярные выражения и язык регулярных выражений применяются уже немало лет. Знатоки регулярных выражений давно вооружились этим эффективным средством для выполнения самых разных операций по обработке текста и манипулирования им практически на любом языке и на любой платформе.

Но это лишь одна, приятная, сторона дела. А обратная, неприятная, сторона состоит в том, что регулярные выражения слишком долго оставались исключительным средством только для самых технически грамотных пользователей. Большинство остальных не совсем понимают назначение регулярных выражений и задачи, которые они призваны решать. А те, кто осмелился осваивать регулярные выражения, обнаружили их синтаксис не очень понятным, а порой и запутанным. Но на самом деле регулярные выражения не так сложны, как кажется. Чтобы умело ими пользоваться, достаточно уяснить поставленную задачу и ее наилучшее решение с помощью регулярных выражений.

Отчасти упомянутые выше трудности связаны со скудостью качественной литературы на данную тему. По регулярным выражениям имеется немало книг, а также немало вебсайтов, изобилующих учебными материалами для изучения регулярных выражений, на которых основное внимание уделяется синтаксису и, в частности, назначению и отличиям знака { от знаков + и *. Но на самом деле в этом нет ничего сложного, поскольку в языке регулярных выражений не так уж и много специальных символов. Сложнее понять, как пользоваться ими для решения практических задач.

Эта книга не призвана служить исчерпывающим пособием по регулярным выражениям. Если же вам требуется именно такое пособие, приобретите третье издание книги

Mastering Regular Expressions Джеффри Фридла (Jeffrey Friedl; издательство O'Reilly)[1]. Автор этого исчерпывающего пособия является признанным знатоком регулярных выражений, но, не в обиду ему будь сказано, его книга — не для начинающих и не для случайных пользователей, которым просто нужно решить конкретную задачу, не особенно вдаваясь в принцип действия регулярных выражений. Нельзя сказать, что от этой книги мало проку этим категориям пользователей, но она вряд ли поможет им в решении таких практических задач, как внедрение проверки достоверности данных, вводимых в HTML-формах, или выполнение простых операций замены в синтаксически анализируемом тексте. Чтобы быстро научиться пользоваться регулярными выражениями, прочитайте эту книгу, и вы окажетесь в положении человека, приобретшего нечто среднее между слишком скудными сведениями, чтобы работать продуктивно, и слишком обширными сведениями, чтобы знать, как приступить к делу.

И здесь на помощь вам придет предлагаемая вашему вниманию книга. Читая ее, вы научитесь пользоваться теми регулярными выражениями, которые вам действительно нужно знать, начиная с поиска простых совпадений с заданным текстом и заканчивая более сложными задачами, включая применение обратных ссылок, условные вычисления и обработку с упреждением. Прорабатывая материал каждого урока в данной книге, вы методически, систематически и легко научитесь решать практические задачи, используя регулярные выражения.

---

[1] В русском переводе это издание книги вышло под названием *Регулярные выражения* в издательстве "Символ Плюс", СПб, 2008. — *Примеч. ред.*

Кому адресована эта книга

Данная книга адресована тем, кто:

- только начинает изучать регулярные выражения;
- хочет быстро научиться извлекать наибольшую пользу из языка регулярных выражений;
- желает добиться преимущества, научившись решать практические задачи с помощью одного из самых эффективных (и малопонятных) инструментальных средств;
- стремится разрабатывать веб-приложения с развитой обработкой форм и текста;
- испытывает потребность научиться пользоваться поддержкой регулярных выражений в JavaScript, Java, .NET, PHP, Python, MySQL и многих других языках, СУБД и платформах при разработке приложений;
- стремится работать продуктивно, легко и быстро, разбираясь в регулярных выражениях без посторонней помощи.

Итак, перейдите к первому уроку в данной книге, чтобы сразу же приступить к делу. Вам не потребуется много времени, чтобы научиться пользоваться огромным потенциалом регулярных выражений, а в конечном итоге вам будет невдомек, как это вы раньше умудрялись без них обходиться.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`

WWW: `http://www.dialektika.com`

Урок 1

Введение в регулярные выражения

Из этого урока вы узнаете, что такое регулярные выражения и чем они могут вам помочь.

Потребность в регулярных выражениях

Регулярные выражения (нередко обозначаемые на английском как *RegEx* или *regex*) являются инструментальными средствами, предназначенными для решения конкретных задач. Чтобы лучше всего понять регулярные выражения и их назначение, необходимо уяснить задачу, которую они позволяют решить.

Рассмотрим следующие примеры решаемых задач.

- Найти файл, содержащий текст `car` (без учета регистра букв), но только не в средине слова (например, `scar`, `carry` и `incarcerate`).
- При формировании веб-страницы отобразить текст, извлеченный из базы данных. Этот текст может содержать URL, которые требуется сделать активизируемыми щелчком кнопкой мыши на формируемой странице. Таким образом, вместо самого текста достаточно сформировать достоверную HTML-ссылку `<a href></a>`.

- При создании приложения с формой, в которой у пользователя запрашиваются сведения, включающие адрес электронной почты, проверить, указан ли адрес в верном формате, т.е. достоверен ли он синтаксически.
- При редактировании исходного кода заменить все вхождения слова `size` словом `iSize`, но только как отдельного слова, а не части другого слова.
- При отображении списка всех файлов, присутствующих в файловой системе отдельного компьютера, найти только те файлы, которые содержат текст `Application`.
- Импортировать данные, разделяемые знаками табуляции, в приложение, в котором поддерживаются файлы формата CSV, в котором значения в отдельных строках разделяются запятыми, а возможно, и заключены в круглые скобки.
- Найти в файле конкретный текст, но только в отдельном месте (предположительно в начале строки или в конце предложения).

Решение всех перечисленных выше примеров задач представляет особые трудности программирования, хотя все они могут быть решены практически на любом языке программирования, поддерживающем условные вычисления и манипулирование символьными строками. Но насколько сложным может оказаться решение подобных задач? Для этого, вероятно, придется циклически перебрать слова или отдельные символы, выполнить самые разные виды проверок в условном операторе `if`, отследить немало признаков состояния, чтобы выяснить, что удалось и что не удалось обнаружить, проверить наличие пробелов и специальных символов и т.д. и т.п. И каждый раз все это придется делать вручную.

А с другой стороны, можно воспользоваться регулярными выражениями. Каждую из перечисленных выше задач можно решить с помощью грамотно составленных

операторов в виде очень кратких символьных строк, содержащих текст и специальные инструкции, как в приведенном ниже примере.

```
\b[Cc][Aa][Rr]\b
```

Примечание

Если приведенная выше строка кода вам пока еще непонятна, не отчаивайтесь. Вскоре вы поймете ее назначение.

О применении регулярных выражений

Просмотрите еще раз примеры решаемых задач, перечисленные в предыдущем разделе, обратив внимание на то, что все они могут быть отнесены к одной из следующих категорий: поиск информации или же поиск и замена информации. На самом деле поиск и замена информации — это основное применение регулярных выражений в простейшем случае. Каждое регулярное выражение служит для сопоставления с текстом (т.е. для выполнения операции поиска) или же для сопоставления с текстом и его замены (т.е. для выполнения операции замены).

Операции поиска с помощью регулярных выражений

Регулярные выражения применяются в операциях поиска, когда требуется найти весьма динамичный текст, как в упомянутом ранее примере задачи для поиска слова `car`. Прежде всего, необходимо найти слово `car`, `CAR`, `Car` или даже `CaR`, что сделать совсем нетрудно, поскольку многие инструментальные средства способны выполнять поиск без учета регистра букв. Сложнее убедиться, что слова `scar`, `carry` и `incarcerate` не совпадают с заданным критерием поиска. В одних более развитых редакторах текста имеются варианты поиска наподобие Match Only Whole Word (Найти слово целиком), но во многих других редакторах такая возможность отсутствует, что не позволяет внести подобные изменения в редактируемый документ. В качестве выхода

из этого затруднительного положения вместо одного лишь искомого текста `car` можно воспользоваться регулярным выражением.

> **Совет**
>
> Хотите знать решение упомянутой выше задачи? Оно уже было представлено выше в следующем операторе: **`\b[Cc][Aa][Rr]\b`**.

Следует заметить, что проверка на равенство (например, совпадает ли указанный пользователем адрес электронной почты с данным регулярным выражением) относится к операции поиска. В этой операции осуществляется поиск на совпадение со всей символьной строкой, введенной пользователем, в отличие от поиска подстроки, что обычно имеет место в операциях поиска.

Операции замены с помощью регулярных выражений

Операции поиска с помощью регулярных выражений необыкновенно эффективны, весьма полезны и несложны в освоении, и поэтому многие уроки и примеры в данной книге, по существу, сводятся к обнаружению совпадений. Но истинный потенциал регулярных выражений кроется в операциях замены, как в упоминавшемся ранее примере задачи для замены URL их ссылками, активизируемыми щелчком кнопкой мыши на веб-странице. Для решения этой задачи необходимо сначала найти URL в тексте (предположительно — символьные строки, начинающиеся на `http://` или `https://` и оканчивающиеся точкой, запятой или пробелом). Затем требуется заменить обнаруженный URL двумя вхождениями совпавшей символьной строки со встроенной HTML-разметкой. Так, строку

```
http://www.forta.com/
```

необходимо заменить строкой

```
<a href="http://www.forta.com">http://www.forta.
com/</a>
```

Доступный в большинстве приложений вариант Search and Replace (Найти и заменить) не позволяет выполнить подобного рода операцию, тогда как с помощью регулярных выражений сделать это совсем нетрудно.

Так что же такое регулярное выражение?

Итак, выяснив назначение регулярных выражений, перейдем к их определению. Проще говоря, *регулярные выражения* — это символьные строки, предназначенные для сопоставления с текстом и манипулирования им. Регулярные выражения составляются на языке, специально предназначенном для выполнения описанных выше и прочих операций. Подобно любому другому языку, у языка регулярных выражений имеются свой особый синтаксис и инструкции, которые следует изучить. Именно в этом и поможет вам данная книга.

Язык регулярных выражений нельзя назвать полноценным языком программирования. Он даже не позволяет написать конкретную программу или утилиту, которую можно было бы установить и использовать. Чаще всего регулярные выражения реализуются в виде мини-языков, встраиваемых в другие языки или программные продукты. Приятная сторона дела состоит в том, что регулярные выражения поддерживаются практически во всех приличных языках программирования или инструментальных средствах, а неприятная сторона — сам язык регулярных выражений совсем не похож на тот язык программирования или инструментальное средство, в котором он применяется. Этот язык не настолько самоочевиден и понятен, как остальные языки.

Примечание

Регулярные выражения появились в результате научных исследований, проводившихся в 1950-е годы в области математики. Спустя годы принципы и идеи были перенесены из этих первоначальных исследований в среду операционной системы Unix (в частности, в утилиту **grep**) и языка программирования Perl. Многие годы регулярные выражения, применявшиеся для решения задач, подобных описанным ранее, применялись исключительно в сообществе пользователей Unix. Но это положение изменилось, и теперь регулярные выражения поддерживаются в самых разных формах практически на всех вычислительных платформах.

Итак, подводя итог всему сказанному выше, приведем ряд примеров регулярных выражений (все они достоверны, а их смысл станет вам вскоре понятен).

- `Ben`
- `.`
- `www\.forta\.com`
- `[a-zA-Z0-9_.]*`
- `<[Hh]1>.*</[Hh]1>`
- `\r\n\r\n`
- `\d{3,3}-\d{3,3}-\d{4,4}`

Следует особо подчеркнуть, что в овладении регулярными выражениями проще всего освоить их синтаксис. А самое трудное — научиться применять этот синтаксис, разбивая поставленные задачи на отдельные решения, реализуемые с помощью регулярных выражений. Этому нельзя научиться, прочитав книгу, но, как и в изучении любого языка, мастерство приходит с практикой.

Применение регулярных выражений

Как пояснялось ранее, составить программу из регулярных выражений нельзя. Это не прикладная программа, которую

можно выполнить, и не программное обеспечение, которое можно приобрести или загрузить из Интернета. Напротив, язык регулярных выражений реализован во многих программных продуктах, языках программирования, утилитах и средах разработки.

Порядок применения регулярных выражений и доступность их функциональных возможностей может различаться в разных приложениях. Так, в некоторых приложениях могут предоставляться отдельные пункты меню и диалоговые окна для доступа к регулярным выражениям, тогда как в языках программирования, как правило, предоставляются функции, классы или объекты для раскрытия функциональных возможностей регулярных выражений. Кроме того, не все реализации регулярных выражений одинаковы. Зачастую они едва (а иногда весьма) заметно различаются синтаксисом и доступными средствами.

В приложении к данной книге подробно представлены примечания к инструментальным средствам и языкам программирования, поддерживающим регулярные выражения. Поэтому, прежде чем перейти к следующему уроку, обратитесь за справкой к приложению по поводу того инструментального средства или языка программирования, которым вы собираетесь пользоваться, изучая регулярные выражения.

Чтобы быстрее приступить к делу, ссылки на оперативно доступные инструментальные средства проверки регулярных выражений можно найти на веб-странице, посвященной данной книге и доступной по следующему адресу:

```
http://forta.com/books/0134757068/
```

Эти инструментальные средства зачастую предоставляют самый простой способ для экспериментирования с регулярными выражениями.

Предварительные замечания

Прежде чем двигаться дальше, примите во внимание следующие важные замечания.

- Применяя регулярные выражения, вы непременно обнаружите, что у любой задачи почти всегда имеется несколько решений. Одни решения оказываются более простыми, другие — более быстрыми, третьи — более переносимыми, а четвертые — более пригодными. При написании регулярных выражений редко появляются верные или неверные решения, конечно, при условии, что решение вполне работоспособно.
- Как отмечалось ранее, у разных реализаций регулярных выражений имеются свои особенности. Поэтому примеры и уроки в данной книге применимы в как можно большей степени ко всем главным реализациям регулярных выражений, а там, где это уместно, указываются отличия или несоответствия.
- Для овладения регулярными выражениями, как и любым языком, требуется практика, практика и еще раз практика.

Примечание

Настоятельно рекомендуется опробовать каждый пример из данной книги по мере проработки ее материала.

Резюме

Регулярные выражения относятся к числу самых эффективных инструментальных средств, доступных для манипулирования текстом. Для составления регулярных выражений (а по существу, символьных строк, называемых *регулярными выражениями*) имеется специальный язык. Регулярные выражения служат для выполнения операций поиска и замены.

Урок 2

Обнаружение совпадения с отдельными символами

Из этого урока вы узнаете, как обнаружить совпадение с одним или несколькими символами.

Совпадение с обычным текстом

Имя `Ben` — это регулярное выражение. Это простой текст, и поэтому он может быть непохожим на регулярное выражение. Регулярные выражения могут содержать простой текст — и даже *один лишь* простой текст. Следует, однако, признать, что такое употребление регулярных выражений считается напрасной тратой ресурсов на их обработку, хотя и служит неплохой отправной точкой для изучения регулярных выражений.

Итак, рассмотрим следующий пример регулярного выражения, состоящего только из обычного текста.

Текст

```
Hello, my name is Ben. Please visit
my website at http://www.forta.com/.[1]
```

[1] Здравствуйте, меня зовут Бен. Будьте любезны, посетите мой веб-сайт по адресу http://www.forta.com/.

Регулярное выражение

```
Ben
```

Результат

```
Hello, my name is Ben. Please visit
my website at http://www.forta.com/.
```

Анализ

В качестве регулярного выражения здесь был использован обычный текст, который совпал со словом Ben в исходном тексте данного примера.

> **Примечание**
>
> В примерах из этой книги совпавший текст специально выделяется для большей наглядности как результат применения регулярного выражения.

Рассмотрим еще один пример применения того же самого исходного текста, но для обнаружения совпадения с другим регулярным выражением.

Текст

```
Hello, my name is Ben. Please visit
my website at http://www.forta.com/.
```

Регулярное выражение

```
my
```

Результат

```
Hello, my name is Ben. Please visit
my website at http://www.forta.com/.
```

Анализ

Искомый текст `my` также является статическим, но обратите внимание на то, что в исходном тесте обнаружены два совпадения с ним.

Количество совпадений

По умолчанию большинство механизмов обработки регулярных выражений должны возвращать только первое совпадение. В приведенном выше примере таким совпадением оказывается первое, но не второе слово `my`. Так почему же были обнаружены два совпадения? В большинстве реализаций регулярных выражений предоставляется механизм для получения списка всех совпадений (этот список обычно возвращается в массиве или в какой-то другой специальной форме). Так, если установить в JavaScript дополнительный признак `g`, обозначающий глобальное совпадение, то в конечном счете будет возвращен массив, содержащий все совпадения.

> **Примечание**
>
> Чтобы выяснить, как организовать глобальное совпадение в конкретном языке программирования или инструментальном средстве, обращайтесь за справкой к приложению, приведенному в конце данной книги.

Учет регистра букв

В регулярных выражениях учитывается регистр букв, и поэтому слово `Ben` не совпадет со словом `ben`. Но в большинстве реализаций регулярных выражений поддерживается также обнаружение совпадений без учета регистра букв. Например, программирующие на JavaScript могут установить дополнительный признак `i`, чтобы принудительно организовать поиск без учета регистра букв.

Примечание

Чтобы выяснить, как организовать поиск без учета регистра букв в конкретном языке программирования или инструментальном средстве, обращайтесь за справкой к приложению, приведенному в конце данной книги.

Совпадение с любыми символами

Рассмотренные до сих пор регулярные выражения позволяли обнаруживать совпадение только со статическим текстом, что, конечно, малоинтересно. Поэтому рассмотрим далее совпадение с неизвестными заранее символами. Для обозначения критерия поиска (т.е. того, что требуется найти) в регулярных выражениях употребляются специальные символы (или даже целые их наборы). Так, знак точки обозначает совпадение с любым одиночным символом. Следовательно, поиск по критерию `c.t` приведет к совпадению со словами `cat` и `cot` (и целым рядом других бессмысленных слов).

Ниже приведен пример применения регулярного выражения со знаком точки.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
sales.
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Анализ

В данном примере регулярное выражение `sales.` используется для поиска всех имен файлов, начинающихся со слова `sales`, после которого следует любой другой символ. С этим шаблоном совпали имена трех из девяти файлов.

> **Совет**
>
> Далее в тексте данной книги будет не раз встречаться термин *шаблон*, предназначенный для описания конкретного регулярного выражения.

> **Примечание**
>
> Следует иметь в виду, что в регулярных выражениях осуществляется сопоставление строкового содержимого с заданным шаблоном. В итоге с шаблоном не всегда совпадают целые строки, а только отдельные символы — даже если они составляют лишь часть строки. В приведенном выше примере регулярное выражение совпало не со всем именем файла, а только с его частью. Об этом отличии очень важно не забывать, передавая результаты применения регулярного выражения в какую-нибудь другую часть прикладного кода для последующей обработки.

Точка обозначает совпадение с любым одиночным символом, будь то буква, цифра и даже сам знак точки, как демонстрируется в следующем примере.

Текст

```
sales.xls
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
sales.
```

Результат

```
sales.xls
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Анализ

В данном примере добавлен один дополнительный файл `sales.xls`. Имя этого файла совпало с шаблоном `sales.`, поскольку знак `.` обозначает совпадение с любым одиночным символом, включая и сам знак точки.

В разных местах шаблона можно употреблять несколько экземпляров знака `.` как вместе (т.е. один после другого, например `..` для обнаружения совпадения с любыми двумя соседними символами), так и порознь. Рассмотрим еще один пример применения того же самого исходного текста. На этот раз требуется найти все файлы для Северной

Америки (na) и Южной Африки (sa) независимо количества последующих цифр.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
.a.
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Анализ

С помощью регулярного выражения `.a.` действительно удалось обнаружить файлы na1, na2 и sa1, но, кроме них, есть еще четыре не предполагавшихся совпадения. Почему? А потому, что шаблон в этом регулярном выражении совпадает с тремя любыми символами, при условии, что вторым из них оказывается символ a.

Опробуем другой шаблон, добавив еще один знак точки к шаблону `.a.`, как показано в следующем примере.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
.a..
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Анализ

Шаблон `.a..` оказался ничуть не лучше шаблона `.a.`. Присоединение к нему еще одного знака точки обеспечило лишь совпадение с дополнительным символом, каким бы он ни был. Как же тогда найти знак точки, если он имеет специальное назначение для совпадения с любым символом?

Совпадение со специальными символами

Знак точки имеет специальное назначение в регулярных выражениях. Если этот знак требуется употребить в шаблоне, придется каким-то образом указать, что в регулярном выражении он должен выполнять свое обычное, а не

специальное назначение. Для этого достаточно экранировать, а по существу, предварить знак точки знаком обратной косой черты (\). Этот знак служит в качестве *метасимвола*, т.е. символа со специальным назначением, в отличие от его прямого назначения как знака обратной косой черты. Таким образом, знак . обозначает совпадение с любым символом, а знаки \. — непосредственно знак точки.

Рассмотрим предыдущий пример снова, но на этот раз экранируем знак точки в шаблоне знаком обратной косой черты, как показано ниже.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
.a.\.
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Анализ

Благодаря шаблону .a.\. поставленная цель достигнута. Первый знак точки в этом шаблоне обеспечил совпадение

с буквой n (в первых двух случаях) или с буквой s (в третьем случае), второй знак точки — совпадение с цифрой 1 (в первых двух случаях) или с цифрой 2 (в третьем случае), а последующие знаки \. — совпадение со знаком точки, отделяющим имя файла от его расширения.

Шаблон из рассмотренного выше примера можно усовершенствовать, включив в него расширение файла xls, чтобы исключить совпадение с такими именами файлов, как sa3.doc. Ниже приведен соответствующий пример.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

Регулярное выражение

```
.a.\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
```

В регулярных выражениях знак \ всегда обозначает начало блока, состоящего из одного или нескольких символов, имеющих специальное назначение. В приведенных выше

примерах было продемонстрировано употребление знака \ в блоке символов \., а на последующих уроках будут представлены другие примеры применения этого знака.

Пример

Применение специальных символов более подробно рассматривается в уроке 4.

Пример

Если знак \ требуется употребить в шаблоне непосредственно как знак обратной косой черты, его следует экранировать, указав в итоге два знака \\.

Совет

Как пояснялось выше, знак точки обозначает совпадение с любым одиночным символом. И это действительно так в большинстве реализаций регулярных выражений, кроме знака новой строки.

Резюме

Регулярные выражения, иначе называемые шаблонами, представляют собой строки, состоящие из символов. Эти символы могут быть обычными, образующими текст, или метасимволами, имеющими специальное назначение. Из этого урока вы узнали, как обнаружить совпадение с одиночным символом, используя обычный текст и метасимволы. В частности, знак точки (.) обозначает совпадение с любым одиночным символом, а знак обратной косой черты (\) служит для экранирования символов и обозначает начало последовательности специальных символов.

Урок 3

Совпадение с набором символов

Из этого урока вы узнаете, как обращаться с наборами символов. В отличие от знака точки, обозначающего совпадение с любым одиночным символом, как пояснялось в предыдущем уроке, наборы позволяют обнаруживать совпадения с конкретными символами и их диапазонами.

Совпадение с одним из нескольких символов

Как вам должно быть уже известно из предыдущего урока, знак точки обозначает совпадение с любым одиночным символом, как это происходит с любым указанным обычным символом. В последнем примере из предыдущего урока шаблон `.a` обеспечивал совпадение как с символами `na`, так и с символами `sa`, а знак `.` — совпадение с обоими символами, `n` и `a`. Но что если среди искомых файлов имеется файл `ca1.xls` с данными о продажах в Канаде, но по-прежнему требуется обнаружить совпадение только с файлами, содержащими данные о продажах в Северной Америке (`na`) и Южной Африке (`sa`)? Знак `.` обеспечит совпадение и с символом `c`, а следовательно, будет обнаружен и файл `ca1.xls`, как показано ниже.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Регулярное выражение

```
.a.\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Чтобы найти символ `n` или `s`, совсем не обязательно обеспечить совпадение с *любым* символом. Для этого достаточно и совпадения именно с этими двумя символами. В регулярных выражениях наборы символов определяются с помощью метасимволов **[** и **]**. Эти метасимволы задают пределы для набора символов, чтобы обеспечить совпадение с любым, но не всеми символами из данного набора.

Ниже приведен вариант примера из предыдущего урока, переделанный с целью продемонстрировать совпадение с набором символов.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Регулярное выражение

```
[ns]a.\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Анализ

Регулярное выражение, применяемое в данном примере, начинается с набора символов `[ns]`, обеспечивающего совпадение с буквой `n` или `s`, но не с буквой `c` или любым другим символом. Знаки открывающей (`[`) и закрывающей (`]`) скобок определяют сопоставляемый набор символов, а не любые символы. Символ `a` в шаблоне из данного примера обеспечивает совпадение с буквой `a`, знак `.` — совпадение с любым последующим символом, знаки `\.` — совпадение со знаком точки, а символы `xls` — с расширением файла `xls`. По такому шаблону обнаруживается совпадение только с тремя требующимися именами файлов.

Примечание

На самом деле шаблон **[ns]a.\.xls** не совсем подходит для рассмотренного выше примера. Так, если бы существовал файл **usa1.xls**, он так же совпал бы с данным шаблоном (начальная буква **u** была бы проигнорирована, а остальная часть **usa1.xls** имени файла совпала бы). В качестве выхода из этого положения можно организовать совпадение позиций, более подробно рассматриваемое в уроке 6.

Совет

Как видите, проверить правильность регулярных выражений не так-то просто. Проверить, совпадает ли шаблон именно с тем, что нужно, совсем нетрудно. Но настоящие трудности возникают, когда требуется проверить, не совпадает ли шаблон и с тем, что не нужно.

Наборы символов нередко применяются для поиска всего набора (или отдельных его частей) без учета регистра букв, как демонстрируется в следующем примере.

Текст

```
The phrase "regular expression" is often
abbreviated as RegEx or regex.[1]
```

Регулярное выражение

```
[Rr]eg[Ee]x
```

Результат

```
The phrase "regular expression" is often
abbreviated as RegEx or regex.
```

Анализ

Шаблон, применяемый в данном примере, содержит два набора символов. В частности, набор `[Rr]` обеспечивает

[1] Термин "regular expression" (регулярное выражение) нередко сокращается как RegEx или regex.

совпадение с буквами R и r, а набор [Ee] — совпадение с буквами E и e. Таким образом, в исходном тексте обнаруживается совпадение со словами RegEx и regex. Тем не менее данный шаблон не обеспечивает совпадение со словом REGEX.

> **Совет**
>
> Для совпадения без учета регистра букв приведенный выше метод оказывается излишним. Ведь такого рода совпадение применяется только в операциях поиска с учетом регистра букв, которые частично выполняются без учета регистра букв.

Применение диапазонов в наборах символов

Обратимся снова к примеру со списком файлов. Шаблону [ns]a.\.xls, применявшемуся в последнем примере для поиска файлов, присущ еще один недостаток. Что если в списке имеется файл sam.xls? Он также совпал бы с данным шаблоном, поскольку знак точки обозначает совпадение с любым символом, а не только с цифрой.

И этот недостаток позволяют устранить наборы символов, как показано ниже.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Регулярное выражение

```
[ns]a[0123456789]\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Анализ

В данном примере шаблон был видоизменен таким образом, чтобы первым совпала буква `n` или `s`, второй — буква `a`, а третьей — любая цифра, как указано в наборе `[0123456789]`. Обратите внимание на то, что файл `sam.xls` не совпал с данным шаблоном, поскольку буква `m` не совпадает с перечисленным в нем набором символов, состоящим из десяти цифр.

Работая с регулярными выражениями, вы обнаружите, что вам придется не раз употреблять диапазоны символов (от `0` до `9`, от `A` до `Z` и т.д.). Чтобы упростить работу с диапазонами символов, в регулярных выражениях предоставляется специальный метасимвол `-` (знак дефиса), предназначенный для обозначения конкретного диапазона.

Следующий пример такой же, как и предыдущий, но на этот раз в нем указывается конкретный диапазон символов.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
```

```
na2.xls
sa1.xls
ca1.xls
```

Регулярное выражение

```
[ns]a[0-9]\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Анализ

Шаблон `[0-9]` функционально равнозначен шаблону `[0123456789]`, а следовательно, и полученный результат оказывается таким же, как и в предыдущем примере.

Диапазоны не ограничиваются только цифрами. Ниже перечислены все допустимые диапазоны символов.

- `A-Z` — обеспечивает совпадение со всеми прописными буквами от A до Z.
- `a-z` — обеспечивает совпадение со всеми строчными буквами от a до z.
- `A-F` — обеспечивает совпадение только с прописными буквами от A до F.
- `A-z` — обеспечивает совпадение со всеми символами в коде ASCII от A до z. Этот диапазон не вполне пригоден для составления шаблонов, поскольку включает в себя такие символы, как `[` и `^`, которые находятся в промежутке между буквами `Z` и `a` в таблице символов, представленных в коде ASCII.

Любые два символа в коде ASCII могут быть указаны в качестве начала и конца конкретного диапазона. Но на практике диапазоны обычно составляются из некоторых или всех цифр, а также из некоторых или всех букв.

Совет

Применяя диапазоны символов, будьте внимательны, чтобы не указать конец диапазона меньше его начала, как, например, `[3-1]`. Такой диапазон непригоден для шаблона, поскольку он зачастую препятствует нормальному действию шаблона.

Примечание

Знак дефиса (`-`) — это специальный метасимвол, поскольку он употребляется как метасимвол, когда указывается в наборе, заключаемом между знаками `[` и `]`. А за пределами этого набора знак `-` употребляется как таковой, обеспечивая совпадение только со знаком дефиса. Следовательно, знак `-` совсем необязательно экранировать.

Несколько диапазонов могут быть объединены в единый набор символов. Так, следующий шаблон совпадает с любым буквенно-цифровым символом в верхнем или нижнем регистре, но ни с каким другим символом, который не является ни цифрой, ни буквой:

```
[A-Za-z0-9]
```

Этот шаблон является сокращенной формой приведенного ниже шаблона. Как видите, диапазоны делают намного более ясным синтаксис регулярных выражений.

```
[ABCDEFGHIJKLMNOPQRSTUVWXYZabcde
 fghijklmnopqrstuvwxyz01234567890]
```

Рассмотрим еще один пример, в котором осуществляется поиск значений цвета RGB, обозначающих в шестнадцатеричной форме количество основных (красной, зеленой и синей) составляющих отдельного цвета. На веб-страницах значения цвета RGB указываются как `#000000` (черный),

#ffffff (белый), #ff0000 (красный) и т.д. А поскольку эти значения могут быть указаны как в верхнем, так и в нижнем регистрах букв, значение #FF00ff (пурпурный) вполне допустимо. Приведенный ниже пример взят из файла стилевых таблиц CSS.

Текст

```
body {
    background-color: #fefbd8;
}
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Регулярное выражение

```
#[0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]
[0-9A-Fa-f][0-9A-Fa-f]
```

Результат

```
body {
    background-color: #fefbd8;
}
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Анализ

Шаблон, применяемый в данном примере, состоит из знака # как обычного текстового символа и диапазона

символов [0-9A-Fa-f], повторяющегося шесть раз подряд. Такой шаблон обеспечивает совпадение со знаком # и шестью последующими символами, каждый из которых должен быть цифрой или буквой от A до F (прописной или строчной).

Совпадение с любыми символами, кроме указанных

Наборы символов обычно служат для указания списка символов, любой из которых должен совпасть. Но иногда требуется обратное: список символов, которые не должны совпасть. Иными словами, совпасть должны любые символы, *кроме* указанных в списке.

Вместо того чтобы перечислять каждый требующийся символ, что может быть долго, если требуются все символы, кроме нескольких, наборы символов можно указывать с отрицанием, используя метасимвол ^. Ниже приведен соответствующий пример.

Текст

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Регулярное выражение

```
[ns]a[^0-9]\.xls
```

Результат

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

Анализ

Шаблон, применяемый в данном примере, полностью противоположен шаблону из предыдущего примера. Если шаблон `[0-9]` обеспечивает совпадение со всеми цифрами (и только с ними), то шаблон `[^0-9]` — совпадение с любыми символами, кроме цифр в указанном диапазоне. Таким образом, по шаблону `[ns]a[^0-9]\.xls` обнаруживается файл `sam.xls`, но не файл `na1.xls`, `na2.xls` или `sa1.xls`.

> **Примечание**
>
> Метасимвол `^` обозначает отрицание всех символов или диапазона из набора символов, а не только символа или диапазона, который он предваряет.

Резюме

Метасимволы `[` и `]` служат для обозначения наборов символов, любой из которых должен совпасть (по логической операции ИЛИ, а не И). Можно перечислять наборы символов полностью или указывать лишь их пределы, используя метасимвол `-`. Кроме того, наборы символов можно отрицать, используя метасимвол `^`, чтобы обеспечить совпадение с любыми символами, кроме указанных.

Урок 4

Применение метасимволов

Некоторые метасимволы были представлены в уроке 2. А из этого урока вы узнаете о других метасимволах, применяемых для совпадения с конкретными символами или их типами.

Еще раз об экранировании

Прежде чем перейти непосредственно к описанию метасимволов, целесообразно еще раз упомянуть об экранировании специальных символов. *Метасимволами* называются такие символы, которые имеют специальное назначение в регулярных выражениях. Например, знак точки (`.`) является метасимволом и обозначает совпадение с любым одиночным символом, как пояснялось в уроке 2. Знак открывающей квадратной скобки (`[`) также является метасимволом и обозначает начало набора символов, как пояснялось в уроке 3.

Метасимволы приобретают особое значение, когда они употребляются в регулярных выражениях, и поэтому эти символы нельзя использовать как таковые. Например, знак `[` нельзя использовать для совпадения непосредственно с открывающей квадратной скобкой, а знак `.` — с точкой. Рассмотрим следующий пример, в котором регулярное выражение применяется для того, чтобы обнаружить совпадение с массивом JavaScript, содержащим знаки `[` и `]`.

Текст

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Регулярное выражение

```
myArray[0]
```

Результат

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Анализ

В данном примере блок текста представляет собой фрагмент исходного кода на языке JavaScript. Обычно регулярные выражения применяются в редакторах текста. А в данном примере предполагалось, что регулярное выражение `myArray[0]` буквально совпадет с аналогичным текстом во фрагменте кода, но этого не произошло. Почему? А потому, что метасимволы **[** и **]** обозначают в регулярном выражении набор символов, а не собственно знаки **[** и **]**. А раз так, то шаблон `myArray[0]` совпадет с именем массива `myArray` и одним из членов набора, в котором указан единственный символ `0`. Следовательно, шаблон `myArray[0]` совпадет лишь с текстом `myArray0`.

Как пояснялось в уроке 2, метасимволы можно экранировать, предваряя их знаком обратной косой черты. Так, шаблон `\.` совпадает со знаком точки, а шаблон `\[` — со знаком открывающей квадратной скобки. Каждый метасимвол можно экранировать знаком обратной косой черты, и тогда обеспечивается совпадение с самим символом, а его

специальное назначение как метасимвола игнорируется. Следовательно, для совпадения непосредственно со знаками [и] их необходимо экранировать. Ниже приведен предыдущий пример, но на этот раз метасимволы в шаблоне экранированы.

Текст

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Регулярное выражение

```
myArray\[0\]
```

Результат

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Анализ

На этот раз поиск достиг цели. Шаблон `\[` совпал со знаком `[`, а шаблон `\]` — со знаком `]`. Таким образом, с помощью регулярного выражения `myArray\[0\]` было обнаружено совпадение с элементом массива `myArray[0]`.

Применение регулярного выражения в рассмотренном выше примере оказывается несколько излишним, поскольку для поиска нужного текста было бы достаточно и намного проще указать искомый текст. Но допустим, что требуется обнаружить совпадение не только с элементом массива `myArray[0]`, но и с элементами массива `myArray[1]`, `myArray[2]` и т.д. В таком случае применение регулярного выражения приобрело бы намного больший смысл. Для этого следовало бы экранировать метасимволы [и] и указать

между ними символы, которые должны совпасть. Так, если требуется обнаружить совпадение с элементами массива 0–9, то с этой целью можно воспользоваться следующим регулярным выражением:

```
myArray\[[0-9]\]
```

Совет

Любой метасимвол, а не только упоминавшиеся выше, можно экранировать, предварив его знаком обратной косой черты.

Предупреждение

Метасимволы, указываемые парой (например, знаки `[` и `]`), обязательно должны быть экранированы, если они употребляются не как метасимволы, иначе синтаксический анализатор регулярных выражений может выдать сообщение об ошибке.

Как правило, для экранирования метасимволов применяется знак \. Это означает, что знак \, который также является метасимволом, употребляется для экранирования других символов. Но, как отмечалось в уроке 2, для применения знака \ как такового в регулярном выражении, а не как метасимвола, его также следует экранировать им же самим: \\.

Рассмотрим следующий пример. В качестве исходного текста в данном примере служит путь к файлу в Windows, в котором для этой цели употребляются знаки обратной косой черты. Допустим теперь, что этим путем к файлу требуется воспользоваться в Linux, а следовательно, найти все знаки обратной косой черты и заменить их знаками косой черты.

Текст

```
\home\ben\sales\
```

Регулярное выражение

```
\\
```

Результат

```
\home\ben\sales\
```

Анализ

Шаблон \\ совпадает со знаком \, и поэтому в данном примере обнаружены четыре знака обратной косой черты. Если указать только один знак \ в регулярном выражении, это, вероятнее всего, приведет к ошибке, поскольку синтаксический анализатор регулярных выражений вполне обоснованно предположит, что указанное регулярное выражение не завершено. Ведь после знака \ в регулярном выражении должен следовать еще один символ.

Совпадение с пробельными символами

Как правило, метасимволы делятся на две следующие категории: применяемые для совпадения с текстом (например, знак `.`) и употребляемые как часть синтаксиса регулярных выражений (например, знаки `[` и `]`). Вам предстоит еще открыть немало других метасимволов обеих категорий, начиная с пробельных символов.

Когда поиск выполняется с помощью регулярных выражений, то нередко требуется обнаружить совпадение с непечатаемыми пробельными символами, вводимыми в исходный текст. Например, в исходном тексте требуется найти все знаки табуляции или разрыва строк. Ввести пробельные символы непосредственно в регулярное выражение не так-то просто, поэтому для этой цели можно воспользоваться специальными символами, перечисленными в табл. 4.1.

Таблица 4.1. Пробельные метасимволы

Метасимвол	Описание
`[\b]`	Возврат на один символ ("забой")
`\f`	Перевод страницы
`\n`	Перевод строки

Окончание табл. 4.1

Метасимвол	Описание
`\r`	Возврат каретки
`\t`	Табуляция
`\v`	Вертикальная табуляция

Обратимся к конкретному примеру. Приведенный ниже блок текста содержит ряд записей в формате с разделяющими запятыми, сокращенно обозначаемом как CSV. Прежде чем обрабатывать эти записи, необходимо удалить любые пустые строки в исходных данных, как показано ниже.

Текст

```
"101","Ben","Forta"
"102","Jim","James"


"103","Roberta","Robertson"
"104","Bob","Bobson"
```

Регулярное выражение

```
\r\n\r\n
```

Результат

```
"101","Ben","Forta"
"102","Jim","James"

"103","Roberta","Robertson"
"104","Bob","Bobson"
```

Анализ

Шаблон `\r\n` совпадает с последовательностью знаков возврата каретки и перевода строки, которая в Windows служит в качестве маркера конца строки. Таким образом, поиск по шаблону `\r\n\r\n` приведет к обнаружению совпадения

двух маркеров конца строки, а следовательно, и двух пустых строк, находящихся между двумя записями.

Совет

Как пояснялось выше, последовательность знаков возврата каретки и перевода строки, совпадающая с шаблоном **\r\n**, служит в операционной системе Windows в качестве маркера конца строки. Но в операционных системах Unix, Linux и Mac OSX для этой цели служит лишь знак перевода строки, поэтому в этих системах, вероятнее всего, придется воспользоваться более простым шаблоном, состоящим только из метасимвола **\n**. А в идеальном регулярном выражении, вероятно, должны быть учтены оба варианта обозначения конца строки: необязательный метасимвол **\r** и обязательный метасимвол **\n**. Мы еще вернемся к этому примеру на следующем уроке.

На практике вам, вероятнее всего, придется не раз встречаться с примерами применения метасимволов `\r`, `\n` и `\t`. А остальные пробельные метасимволы применяются нечасто.

Примечание

Выше была продемонстрирована иная разновидность метасимволов. Так, знаки `.` и `[` считаются метасимволами, если они *не* экранированы, а такие буквы, как **f** и **n**, — только в том случае, если они экранированы. Если же эти буквы *не* экранированы, то они употребляются в регулярных выражениях как таковые.

Совпадение с отдельными типами символов

До сих пор было показано, каким образом обнаруживается совпадение с отдельными символами, любыми символами (с помощью знака `.`) и одним из наборов символов (с помощью знаков `[` и `]`) и как отрицаются совпадения (с помощью знака `^`). Наборы символов, в которых обеспечивается совпадение с одним из символов, считаются наиболее употребительной формой совпадения, а вместо часто употребляемых наборов символов могут быть использованы специальные

метасимволы. Такие метасимволы соответствуют *классам* символов. На самом деле метасимволы классов вообще не нужны, поскольку можно всегда перечислить совпадающие символы или указать их диапазоны. Но такие метасимволы, без сомнения, приносят необыкновенную пользу.

> **Примечание**
>
> Упоминаемые далее классы поддерживаются как основные практически во всех реализациях регулярных выражений.

Совпадение с цифровыми и не цифровыми символами

Как вам должно быть уже известно из урока 3, диапазон символов `[0-9]` служит сокращенным обозначением диапазона символов `[0123456789]` и обеспечивает совпадение с любой цифрой. А для совпадения с любым другим символом, кроме цифры, можно воспользоваться отрицательной формой `[^0-9]` такого диапазона символов. Сокращенные формы классов для цифровых и не цифровых символов перечислены в табл. 4.2.

Таблица 4.2. Цифровые и нецифровые метасимволы

Метасимвол	Описание
`\d`	Любая цифра (то же, что и **`[0-9]`**)
`\D`	Любой символ, кроме цифры (то же, что и **`[^0-9]`**)

Чтобы продемонстрировать применение этой разновидности метасимволов, вернемся к рассмотренному ранее примеру.

Текст

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Регулярное выражение

```
myArray\[\d\]
```

Результат

```
var myArray = new Array();
...
if (myArray[0] == 0) {
...
}
```

Анализ

Шаблон `\[` совпадает со знаком `[`, метасимвол `\d` — с любой одиночной цифрой, а шаблон `\]` — со знаком `]`; следовательно, с помощью регулярного выражения `myArray\[\d\]` обнаруживается совпадение с элементом массива `myArray[0]`. Регулярное выражение `myArray\[\d\]` служит сокращенной формой регулярного выражения `myArray\[0-9\]`, а последнее, в свою очередь, — сокращенной формой регулярного выражения `myArray\[0123456789\]`. С помощью этого регулярного выражения обнаруживается также совпадение с элементами массива `myArray[1]`, `myArray[2]` и так далее, но не с элементом массива `myArray[10]`.

Совет

Как видите, определить любое регулярное выражение практически всегда можно несколькими способами. И вы вольны выбрать такой способ, какой вам больше всего подходит.

Предупреждение

В синтаксисе регулярных выражений учитывается регистр букв. В частности, метасимвол **`\d`** обозначает совпадение с цифрами, тогда как метасимвол **`\D`** — наоборот, с нецифровыми символами. То же самое справедливо не только для других рассматриваемых далее метасимволов классов, но даже для совпадения без учета регистра букв. В последнем случае регистр букв *не* будет учитываться в совпадающем тексте, тогда как в специальных символах (например, **`\d`**) он *все же* будет учитываться.

Совпадение с буквенно-цифровыми и не буквенно-цифровыми символами

Еще одним часто употребляемым набором являются буквенно-цифровые символы: буквы от A до Z (как прописные, так и строчные), цифры и знак подчеркивания, нередко применяемый в именах файлов и каталогов, прикладных переменных, объектов базы данных и прочих элементов. Сокращенные формы классов для буквенно-цифровых символов перечислены в табл. 4.3.

Таблица 4.3. Буквенно-цифровые метасимволы

Метасимвол	Описание
\w	Любой буквенно-цифровой символ в верхнем или нижнем регистре, а также знак подчеркивания (то же, что и **[a-zA-Z0-9_]**)
\W	Любой не буквенно-цифровой символ или знак подчеркивания (то же, что и **[^a-zA-Z0-9_]**)

В следующем примере используется фрагмент из базы данных, состоящий из записей с почтовыми индексами США и Канады.

Текст

```
11213
A1C2E3
48075
48237
M1B4F2
90046
H1H2H2
```

Регулярное выражение

```
\w\d\w\d\w\d
```

Результат

```
11213
A1C2E3
48075
48237
M1B4F2
90046
H1H2H2
```

Анализ

В шаблоне из данного примера метасимволы \w и \d применяются для извлечения канадских почтовых индексов.

> **Примечание**
>
> На первый взгляд, приведенный выше пример составлен правильно, но так ли это? Подумайте, почему почтовые индексы США не совпали с шаблоном из данного примера? В том ли дело, что они состоят только из цифр, или для этого имеются иные причины?
>
> Здесь не даются ответы на эти вопросы просто потому, что шаблон из данного примера действует должным образом. Важнее подчеркнуть другое: если регулярные выражения вполне работоспособны, то они редко бывают верными или неверными. Чаще всего они различаются степенью сложности, которая соответствует разной степени строгости сопоставления с шаблоном.

Совпадение с пробельными и не пробельными символами

И наконец, рассмотрим класс пробельных метасимволов. Как пояснялось ранее, для обозначения отдельных пробельных символов имеются соответствующие метасимволы. Сокращенные формы классов для всех пробельных и не пробельных символов перечислены в табл. 4.4.

Таблица 4.4. Пробельные и не пробельные метасимволы

Метасимвол	Описание
`\s`	Любой пробельный символ (то же, что и `[\f\n\r\t\v]`)
`\S`	Любой не пробельный символ (то же, что и `[^\f\n\r\t\v]`)

> **Примечание**
>
> Метасимвол `[\b]`, обозначающий возврат на один символ или "забой", не включается в число пробельных символов, определяемых метасимволом `\s`, не исключается из числа не пробельных символов, определяемых метасимволом `\S`.

Обозначение шестнадцатеричных и восьмеричных значений

Несмотря на то что обозначать отдельные символы их шестнадцатеричными или восьмеричными значениями вряд ли придется на практике, упомянуть о такой возможности все же стоит.

Обозначение шестнадцатеричных значений

Шестнадцатеричные значения (по основанию 16) могут быть указаны с префиксом `\x`. Так, шестнадцатеричное значение `\x0A`, обозначающее ASCII-код `10` знака перевода строки, функционально равнозначно метасимволу `\n`.

Обозначение восьмеричных значений

Восьмеричные (по основанию 8) значения могут быть указаны в виде чисел, состоящих из двух или трех цифр и предваряемых префиксом `\0`. Так, восьмеричное значение `\011`, обозначающее ASCII-код `9` знака табуляции, функционально равнозначно метасимволу `\t`.

Примечание

Во многих реализациях регулярных выражений допускается также обозначать управляющие символы с помощью метасимвола **\c**. Например, шаблон **\cZ** совпадет с комбинацией клавиш <Ctrl+Z>. Но такой синтаксис редко придется применять на практике.

Применение классов символов POSIX

Урок, посвященный метасимволам и сокращенным формам различных наборов символов, был бы неполным без упоминания о классах символов POSIX. В стандарте POSIX определяются классы символов как еще одна сокращенная форма, поддерживаемая во многих, хотя и не во всех реализациях регулярных выражений. Эти классы перечислены в табл. 4.5.

Примечание

Применение классов символов POSIX в регулярных выражениях не поддерживается в языке JavaScript.

Таблица 4.5. Классы символов POSIX

Класс	Описание
[:alnum:]	Любая цифра или буква (то же, что и **[a-zA-Z0-9]**)
[:alpha:]	Любая буква (то же, что и **[a-zA-Z]**)
[:blank:]	Пробел или символ табуляции (то же, что и **[\t]**)
[:cntrl:]	Управляющие символы в коде ASCII (коды ASCII от **0** до **31** и **127**)
[:digit:]	Любая цифра (то же, что и **[0-9]**)
[:graph:]	То же, что и **[:print:]**, кроме пробела
[:lower:]	Любая строчная буква (то же, что и **[a-z]**)
[:print:]	Любой печатаемый символ
[:punct:]	Любой символ, не относящийся ни к буквенно-цифровым (**[:alnum:]**), ни к управляющим символам (**[:cntrl:]**)

Окончание табл. 4.5

Класс	Описание
`[:space:]`	Любой пробельный символ (то же, что и `[\f\n\r\t\v]`)
`[:upper:]`	Любая прописная буква (то же, что и `[A-Z]`)
`[:xdigit:]`	Любая шестнадцатеричная цифра (то же, что и `[a-fA-F0-9]`)

Синтаксис классов символов POSIX заметно отличается от рассматривавшегося до сих пор синтаксиса метасимволов. Чтобы продемонстрировать применение классов POSIX, вернемся к примеру из урока 3, в котором регулярное выражение применялось для обнаружения значений цвета RGB в блоке кода HTML.

Текст

```
body {
    background-color: #fefbd8;
}
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Регулярное выражение

```
#[[:xdigit:]][[:xdigit:]][[:xdigit:]][[:xdigit:]]
[[:xdigit:]][[:xdigit:]]
```

Результат

```
body {
    background-color: #fefbd8;
}
```

```
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Анализ

В шаблоне, применявшемся в примере из урока 3, набор символов `[0-9A-Fa-f]` повторялся шесть раз подряд. Но в шаблоне из данного примера он заменен классом символов `[[:xdigit:]]`. А результат получается такой же самый.

> **Примечание**
>
> Обратите внимание на то, что регулярное выражение, применяемое в приведенном выше примере, начинается со знаков **`[[`** и оканчивается знаками **`]]`**. Это очень важное требование к применению классов символов POSIX. Сами классы символов POSIX заключены между знаками **`[:`** и **`:]`**, поэтому в данном регулярном выражении такой класс обозначен как **`[:xdigit:]`**, а не как **`:xdigit:`**. При этом внешние знаки **`[`** и **`]`** определяют набор символов, а внутренние знаки **`[`** и **`]`** являются частью самого класса символов POSIX.

> **Предупреждение**
>
> Все двенадцать классов символов POSIX, перечисленных в табл. 4.5, обычно поддерживаются в любой реализации регулярных выражений, в которой поддерживается стандарт POSIX. Но в их поддержке могут быть незначительные отличия от описанного выше.

Резюме

Опираясь на основы организации совпадения символов и их наборов, рассмотренные в уроках 2 и 3, в этом уроке были представлены метасимволы, обозначающие

совпадение с конкретными символами (например, со знаками табуляции или перевода строки), целыми наборами или классами символов (например, с цифрами или буквенно-цифровыми символами). Сокращенные формы этих метасимволов и классов символов POSIX могут быть использованы для упрощения шаблонов в регулярных выражениях.

Урок 5

Повторение совпадений

Из предыдущих уроков вы узнали, как обнаруживать совпадения с отдельными символами, используя различные метасимволы и наборы специальных классов символов. А из этого урока вы узнаете, как обнаруживать совпадения с несколькими повторяющимися символами или их наборами.

Количество совпадений

Итак, вы изучили в предыдущих уроках все основы сопоставления с шаблоном в регулярных выражениях, но всем представленным ранее примерам был присущ один очень серьезный недостаток. Рассмотрим в качестве примера, что требуется предпринять для написания регулярного выражения, позволяющего обнаружить совпадение с адресом электронной почты. Основной формат адресов электронной почты выглядит аналогично приведенному ниже.

```
text@text.text
```

Используя метасимволы, описанные в предыдущем уроке, можно составить для этой цели регулярное выражение, подобное следующему:

```
\w@\w\.\w
```

Метасимвол обеспечит совпадение со всеми буквенно-цифровыми символами, а также со знаком подчеркивания, который вполне допустимо указывать в адресе электронной почты. Знак @ экранировать в этом регулярном выражении не нужно, тогда как знак . — нужно.

Это регулярное выражение вполне допустимо, хотя и совершенно бесполезно. Оно обеспечит совпадение с адресом электронной почты наподобие `a@b.c`. И хотя такой адрес синтаксически допустим, он, очевидно, недействителен. Дело в том, что метасимвол `\w` обеспечивает совпадение с единственным символом, но ведь заранее неизвестно, сколько символов следует проверить на совпадение. Так, все приведенные ниже адреса электронной почты действительны, но содержат разное количество символов перед знаком @.

```
b@forta.com
ben@forta.com
bforta@forta.com
```

Следовательно, требуется каким-то образом обеспечить совпадение с несколькими символами. И это можно сделать, используя один из рассматриваемых далее специальных символов.

Совпадение с одним или несколькими символами

Чтобы обнаружить совпадение с одним или несколькими экземплярами отдельного символа (или набора символов), достаточно присоединить к нему знак **+**. Этот знак обозначает совпадение с одним или несколькими символами, т.е. хотя бы с одним символом, а иначе совпадение не происходит. Так, если шаблон `a` обеспечивает совпадение с буквой a, то шаблон `a+` — совпадение с одной или несколькими буквами a. Аналогично шаблон `[0-9]` обеспечивает совпадение с любыми одиночными цифрами, тогда как шаблон `[0-9]+` — с одной или несколькими следующими подряд цифрами.

Совет

Если знак **+** употребляется вместе с наборами символов, он должен быть непременно указан за пределами набора. Следовательно, шаблон **[0-9]+** составлен правильно, а шаблон **[0-9+]** — неправильно. На самом деле шаблон **[0-9+]** является вполне допустимым регулярным выражением, но он не обеспечивает совпадение с одной или несколькими цифрами. Вместо этого он определяет набор символов от **0** до **9** или знак **+**, а следовательно, обеспечивает совпадение с любой одиночной цифрой или со знаком "плюс". И хотя это вполне допустимо, но, вероятно, совсем не то, что нужно.

Вернемся к примеру с адресом электронной почты, но на этот раз воспользуемся знаком **+**, чтобы обнаружить совпадение с одним или несколькими символами, как показано ниже.

Текст

```
Send personal email to ben@forta.com. For questions
about a book use support@forta.com. Feel free to send
unsolicited email to spam@forta.com (wouldn't it be
nice if it were that simple, huh?).[1]
```

Регулярное выражение

```
\w+@\w+\.\w+
```

Результат

```
Send personal email to ben@forta.com. For questions
about a book use support@forta.com. Feel free to send
unsolicited email to spam@forta.com (wouldn't it be
nice if it were that simple, huh?).
```

[1] Отправляйте личную электронную почту по адресу ben@forta.com. С вопросами по книге обращайтесь по адресу support@forta.com. А непрошеную электронную почту можете, не стесняясь, отправлять по адресу spam@forta.com (а как было бы замечательно, если бы это было так просто!).

Анализ

Шаблон, применяемый в данном примере, правильно обнаружил совпадение со всеми тремя адресами электронной почты. Сначала в регулярном выражении обнаруживается совпадение с одним или несколькими символами по шаблону `\w+`, затем — со знаком точки по шаблону `\.`, а далее — с окончанием адреса по еще одному шаблону `\w+`.

Совет

Знак **+** является метасимволом. Чтобы обеспечить совпадение непосредственно со знаком "плюс", его необходимо экранировать: **\+**.

Используя метасимвол **+**, можно также организовать совпадение с одним или несколькими наборами символов. Чтобы продемонстрировать такую возможность, в следующем примере показано применение того же самого регулярного выражения, но к немного другому исходному тексту.

Текст

```
Send personal email to ben@forta.com or
ben.forta@forta.com. For questions about a
book use support@forta.com. If your message
is urgent try ben@urgent.forta.com. Feel
free to send unsolicited email to
spam@forta.com (wouldn't it be nice if
it were that simple, huh?).[2]
```

Регулярное выражение

```
\w+@\w+\.\w+
```

[2] Отправляйте личную электронную почту по адресу ben@forta.com. С вопросами по книге обращайтесь по адресу support@forta.com. А непрошеную электронную почту можете, не стесняясь, отправлять по адресу spam@forta.com (а как было бы замечательно, если бы это было так просто!).

Результат

```
Send personal email to ben@forta.com or
ben.forta@forta.com. For questions about a
book use support@forta.com. If your message
is urgent try ben@urgent.forta.com. Feel
free to send unsolicited email to
spam@forta.com (wouldn't it be nice if
it were that simple, huh?).
```

Анализ

В данном примере регулярное выражение обеспечило совпадение с пятью адресами, но с двумя из них не полностью. Дело в том, что в шаблоне `\w+@\w+\.\w+` не предусмотрены знаки точки перед знаком `@`, а после него допускается только один знак точки, разделяющий две строки. И хотя адрес `ben.forta@forta.com` вполне допустим для электронной почты, данное регулярное выражение обеспечило совпадение только с доменом `forta`, а не с доменом `ben.forta`, поскольку метасимвол `\w` обозначает совпадение с буквенно-цифровыми символами, но не со знаком точки посредине текстовой строки.

В данном случае требуется совпадение с метасимволом `\w` или со знаком `.`, а в терминологии регулярных выражений — с набором символов `[\w\.]`. Ниже приведен исправленный вариант предыдущего примера.

Текст

```
Send personal email to ben@forta.com or
ben.forta@forta.com. For questions about a
book use support@forta.com. If your message
is urgent try ben@urgent.forta.com. Feel
free to send unsolicited email to
spam@forta.com (wouldn't it be nice if
it were that simple, huh?).
```

Регулярное выражение

```
[\w.]+@[\w.]+\.\w+
```

Результат

```
Send personal email to ben@forta.com or
ben.forta@forta.com. For questions about a
book use support@forta.com. If your message
is urgent try ben@urgent.forta.com. Feel
free to send unsolicited email to
spam@forta.com (wouldn't it be nice if
it were that simple, huh?).
```

Анализ

Теперь цель, по-видимому, достигнута. В частности, шаблон `[\w.]+` совпадает с одним или несколькими экземплярами любого буквенно-цифрового символа, знака подчеркивания и знака точки, а следовательно, с доменом `ben.forta`. Этот же шаблон служит и для обнаружения строки после знака `@`, а следовательно, и более глубоко вложенных имен доменов (или сетевых узлов, иначе называемых хостами).

Примечание

Обратите внимание на то, что для совпадения с окончанием адреса электронной почты в регулярном выражении использован шаблон **`\w+`**, а не **`[\w.]+`**. Чтобы выяснить причину такого решения, попробуйте ввести шаблон **`[\w.]`** в конце рассматриваемого здесь регулярного выражения. И тогда вы увидите, что не так со вторым, третьим и четвертым совпадениями.

Примечание

Как видите, знак **`.`** в наборе **`[\w.]`** не был экранирован, но он все равно совпал со знаком точки, поскольку был интерпретирован буквально, а не как метасимвол. В общем, метасимволы наподобие **`.`** и **`+`** считаются элементами обычного текста, когда они употребляются в наборах символов, а следовательно, их не нужно экранировать. Тем не менее экранировать их нисколько не повредит. В частности, шаблон **`[\w.]`** функционально равнозначен шаблону **`[\w\.]`**.

Совпадение с нулевым или большим количеством символов

Как пояснялось выше, метасимвол + обозначает совпадение с одним или несколькими символами, т.е. хотя бы с одним символом, а иначе совпадение не происходит. Но что если требуется обнаружить совпадение с совершенно произвольным количеством символов: от нуля и больше?

Для этой цели служит метасимвол *. Он применяется точно так же, как и метасимвол +, указывается после символа или набора символов и обозначает совпадение с нулевым или большим количеством экземпляров символа или набора символов. Следовательно, шаблон `B.* Forta` обеспечит совпадение с `B Forta`, `B. Forta`, `Ben Forta` и другими сочетаниями имени и фамилии автора данной книги.

Чтобы продемонстрировать применение метасимвола +, рассмотрим следующий видоизмененный вариант примера с адресом электронной почты.

Текст

```
Hello .ben@forta.com is my email address.
```

Регулярное выражение

```
[\w.]+@[\w.]+\.\w+
```

Результат

```
Hello .ben@forta.com is my email address.
```

Анализ

Напомним, что шаблон `[\w.]+` обеспечивает совпадение с одним или несколькими буквенно-цифровыми символами и знаком точки, а следовательно, с доменом `.ben` в приведенном выше адресе электронной почты. Очевидно, в приведенном выше адресе допущена опечатка (лишняя точка в самом его начале), хотя это и не так важно. Важнее другое: несмотря на то что знак . вполне допустим в адресе электронной почты, его нельзя указывать в самом начале этого адреса.

Иными словами, в данном случае требуется обеспечить совпадение с буквенно-цифровым текстом, дополнительно содержащим необязательные символы, как показано в следующем примере.

Текст

```
Hello .ben@forta.com is my email address.
```

Регулярное выражение

```
\w+[\w.]*@[\w.]+\.\w+
```

Результат

```
Hello .ben@forta.com is my email address.
```

Анализ

Приведенный выше шаблон выглядит несколько более сложным. Впрочем, регулярные выражения нередко выглядят намного более сложными, чем они есть на самом деле. Итак, разберем данный шаблон по частям. Сначала шаблон `\w+` обеспечивает совпадение с любым буквенно-цифровым символом, но не со знаком точки, т.е. с теми символами, с которых разрешается начинать адрес электронной почты. После первоначальных достоверных символов могут, конечно, появиться знак точки и дополнительные символы, хотя они могут и отсутствовать. А шаблон `[\w.]*` обеспечивает совпадение с нулевым или большим количеством экземпляров знака точки или буквенно-цифровых символов, что, собственно, и требуется.

Применение

Метасимвол `*` можно рассматривать как обозначающий *нечто необязательное*. В отличие от метасимвола `+`, требующего совпадения хотя бы с одним символом, метасимвол `*` обеспечивает любое количество совпадений, если таковые имеются, хотя и не требует их наличия.

Совет

Знак * является метасимволом. Чтобы обеспечить совпадение непосредственно со знаком "звездочки", его необходимо экранировать: *.

Совпадение с нулевым или единичным количеством символов

К числу весьма полезных относится метасимвол ?. Подобно метасимволу +, метасимвол ? обозначает совпадение с дополнительным текстом, а следовательно, будет обнаружено и нулевое количество совпавших экземпляров. Но, в отличие от метасимвола +, метасимвол ? обозначает совпадение только нулевым или единичным количеством экземпляров символа или набора символов, и не более того. Следовательно, метасимвол ? очень удобен для обнаружения совпадений с конкретными одиночными дополнительными символами в блоке текста.

Рассмотрим следующий пример.

Текст

```
The URL is http://www.forta.com/, to connect
securely use https://www.forta.com/ instead.[3]
```

Регулярное выражение

```
http:\/\/[\w.\/]+
```

Результат

```
The URL is http://www.forta.com/, to connect
securely use https://www.forta.com/ instead.
```

Анализ

В данном примере для обнаружения совпадения с URL применяется шаблон `http:\/\/`, обозначающий обычный

[3] URL — http://www.forta.com/, для безопасного подключения воспользуйтесь адресом https://www.forta.com/.

текст с двумя экранированными знаками косой черты, а следовательно, он совпадает только с самим собой. Далее следует шаблон `[\w./]+`, обеспечивающий совпадение с одним или несколькими экземплярами набора буквенно-цифровых символов, знака точки и косой черты. Этот шаблон позволяет обнаружить совпадение только с первым URL, начинающимся с префикса `http://`, но не со вторым URL, начинающимся с префикса `https://`. Здесь было бы неверно употребить шаблон `s*`, обозначающий совпадение с нулевым или большим количеством экземпляров буквы `s`, поскольку это позволило бы обнаружить совпадение и с префиксом `httpssssss://`, что совершенно неверно.

Как же разрешить данное затруднение? Для этого следует воспользоваться шаблоном `s?`, как демонстрируется в следующем примере.

Текст

```
The URL is http://www.forta.com/, to connect
securely use https://www.forta.com/ instead.
```

Регулярное выражение

```
https?:\/\/[\w.\/]+
```

Результат

```
The URL is http://www.forta.com/, to connect
securely use https://www.forta.com/ instead.
```

Анализ

В данном примере регулярное выражение начинается с шаблона `https?://.`, где знак `?` обозначает, что предшествующий ему символ (в данном случае — буква `s`) должен совпасть как в том случае, если он отсутствует, так и в том случае, если присутствует его единичный экземпляр. Иными словами, шаблон `https?://` обеспечивает совпадение как с префиксом `http://`, так и с префиксом `https://`, но и только.

Между прочим, употребление метасимвола ? позволяет разрешить затруднение, упоминавшееся в предыдущем уроке, в котором рассматривался пример применения шаблона \r\n для обнаружения совпадения с маркером конца линии. Как пояснялось тогда, на платформе Unix или Linux для этой цели необходимо пользоваться шаблоном, состоящим только из метасимвола \n. А в идеальном регулярном выражении, вероятно, должны быть учтены оба варианта обозначения конца строки: необязательный метасимвол \r и обязательный метасимвол \n. Данный пример снова приводится ниже, но на этот раз с несколько видоизмененным регулярным выражением для обнаружения маркера конца строки.

Текст

```
"101","Ben","Forta"
"102","Jim","James"


"103","Roberta","Robertson"
"104","Bob","Bobson"
```

Регулярное выражение

```
[\r]?\n[\r]?\n
```

Результат

```
"101","Ben","Forta"
"102","Jim","James"


"103","Roberta","Robertson"
"104","Bob","Bobson"
```

Анализ

Шаблон [\r]?\n обеспечивает совпадение с дополнительным экземпляром метасимвола \r и обязательным метасимволом \n.

Совет

Как видите, в регулярном выражении из данного примера вместо шаблона `\r?.[\r]` использован шаблон `[\r]?`, в котором определяется набор, состоящий из единственного метасимвола, т.е. единичный набор. Таким образом, шаблон `[\r]?` функционально равнозначен шаблону `\r?`. Знаки `[` и `]` обычно применяются для определения набора символов, хотя некоторые разработчики предпочитают заключать в них даже одиночные символы, чтобы избежать неоднозначности, т.е. ясно указать, к чему именно применяется последующий метасимвол. Если вы пользуетесь знаками `[` и `]` вместе с метасимволом `?`, то непременно укажите метасимвол `?` за пределами набора. Из этого следует, что шаблон `http[s]?://` составлен верно, а шаблон `http[s?]://` — неверно.

Совет

Знак `?` является метасимволом. Чтобы обеспечить совпадение непосредственно с вопросительным знаком, его необходимо экранировать: `\?`.

Применение интервалов

Метасимволы `+`, `*` и `?` служат для решения многих задач с помощью регулярных выражений, но иногда их оказывается недостаточно. Применяя эти метасимволы, необходимо принимать во внимание следующее.

- Метасимволы `+` и `*` обеспечивают совпадение с неограниченным количеством символов. Они не позволяют никоим образом задать максимальное количество совпадающих символов.
- Метасимволы `+`, `*` и `?` поддерживают лишь минимальные количества совпадающих символов: нулевое и единичное. Они не позволяют никоим образом задать вручную минимальное количество совпадений.
- Кроме того, нельзя никоим образом задать требующееся количество совпадений.

Чтобы преодолеть перечисленные выше ограничения и обеспечить в большей степени контроль над повторяющимися совпадениями, в регулярных выражениях допускается применять *интервалы*, которые обычно заключаются в фигурные скобки { и }.

> **Примечание**
>
> Знаки { и } сами являются метасимволами, а следовательно, они должны быть экранированы знаком \, чтобы использовать их как буквальный текст. Следует, однако, иметь в виду, что во многих реализациях регулярных выражений знаки { и } интерпретируются должным образом, даже если они не экранированы, т.е. такие реализации достаточно развиты логически, чтобы различить, когда эти знаки употребляются как таковые, а когда — как метасимволы. Тем не менее на такое поведение лучше не полагаться, экранировав знаки { и }, когда их требуется употребить буквально.

Совпадение с конкретным интервалом символов

Чтобы указать конкретное количество совпадений, достаточно заключить соответствующее число в фигурные скобки { и }. Так, шаблон {3} обозначает три совпадения предыдущего символа или набора символов. Если же совпадений окажется только два, то общего совпадения с таким шаблоном не произойдет.

Чтобы продемонстрировать действие конкретных интервалов символов, обратимся снова к примеру со значениями цвета RGB, рассматривавшемуся в уроках 3 и 4. Напомним, что значения цвета RGB обозначаются тремя рядами шестнадцатеричных чисел, каждое из которых состоит из двух символов. Первый шаблон, применявшийся для сопоставления со значением цвета RGB, выглядел следующим образом:

```
#[0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]
[0-9A-Fa-f]
[0-9A-Fa-f]
```

В уроке 4 этот шаблон был изменен с помощью класса символов POSIX таким образом:

```
#[[:xdigit:]][[:xdigit:]][[:xdigit:]][[:xdigit:]]
[[:xdigit:]][[:xdigit:]]
```

Недостаток обоих приведенных выше шаблонов заключается в том, что конкретный набор (или класс) символов приходится повторять в них шесть раз подряд. Ниже приведен тот же самый пример, но на этот раз в нем применяется совпадение с конкретным интервалом символов.

Текст

```
body {
    background-color: #fefbd8;
}
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Регулярное выражение

```
#[A-Fa-f0-9]{6}
```

Результат

```
body {
    background-color: #fefbd8;
}
h1 {
    background-color: #0000ff;
}
div {
    background-color: #d0f4e6;
}
span {
    background-color: #f08970;
}
```

Анализ

В данном примере шаблон `[A-Fa-f0-9]` совпадает с единственной шестнадцатеричной цифрой, а шаблон `{6}` повторяет это совпадение шесть раз. Этот последний шаблон вполне пригоден и для применения вместе с классами символов POSIX.

Совпадение с интервалом символов в заданных пределах

Интервалы могут быть использованы и для указания конкретного диапазона значений: от минимального до максимального количества совпадающих экземпляров. Диапазоны задаются следующим образом: `{2,4}`, где `2` означает минимум два совпадения, а `4` — максимум четыре совпадения предыдущего символа или набора символов. В приведенном ниже примере демонстрируется регулярное выражение, применяемое для проверки достоверности формата установленных дат.

Текст

```
4/8/17
10-6-2018
2/2/2
01-01-01
```

Регулярное выражение

```
\d{1,2}[-\/]\d{1,2}[-\/]\d{2,4}
```

Результат

```
4/8/17
10-6-2018
2/2/2
01-01-01
```

Анализ

Даты, перечисленные в данном примере, могут быть введены пользователями в поле формы, а следовательно, они должны быть проверены на правильность их соответствия принятому формату дат. В частности, шаблон `\d{1,2}` обеспечивает совпадение с одной или двумя цифрами, что удобно для проверки достоверности указанного дня или месяца. Шаблон `\d{2,4}` обеспечивает совпадение с указанным годом, а шаблон `[-\/]` — совпадение со знаком **-** или же со знаком **/** в качестве разделителя дат. В итоге совпадение в данном примере было обнаружено с тремя датами, но не с датой `2/2/2`, которая не совпала, поскольку год указан в ней слишком коротким.

Совет

В регулярном выражении из данного примера знак / экранирован: `\/`. И хотя во многих реализациях регулярных выражений этого не требуется, тем не менее в некоторых синтаксических анализаторах регулярных выражений такое экранирование считается непременным условием. Поэтому экранировать подобные знаки всегда уместно.

Следует, однако, иметь в виду, что шаблон из приведенного выше примера не проверяет сами даты на достоверность. Например, такие даты, как `54/67/9999`, вполне пройдут проверку по данному шаблону. Ведь этот шаблон проверяет лишь достоверность формата дат, что обычно делается перед проверкой достоверности самих дат.

Примечание

Интервалы могут начинаться с нуля. Так, интервал **`{0,3}`** обозначает от нуля до трех совпадений. Как пояснялось ранее, метасимвол **`?`** обозначает нулевое или единичное количество совпадений предыдущего символа или набора символов. Следовательно, метасимвол **`?`** функционально равнозначен интервалу **`{0,1}`**.

Совпадение хотя бы с заданным интервалом символов

И наконец, интервалы можно применять для указания минимального количества совпадающих экземпляров символов вообще без максимального их количества. Синтаксис такого рода интервала такой же, как и у диапазона, но в нем опущен максимальный предел. Например, шаблон `{3,}` обозначает совпадение хотя бы с тремя экземплярами, иными словами — с тремя или больше экземплярами символов.

Рассмотрим пример, демонстрирующий все, что пояснялось до сих пор в этом уроке. Регулярное выражение применяется в данном примере для обнаружения всех заказов на сумму от 100 и больше долларов США.

Текст

```
1001: $496.80
1002: $1290.69
1003: $26.43
1004: $613.42
1005: $7.61
1006: $414.90
1007: $25.00
```

Регулярное выражение

```
\d+: \$\d{3,}\.\d{2}
```

Результат

```
1001: $496.80
1002: $1290.69
1003: $26.43
1004: $613.42
1005: $7.61
1006: $414.90
1007: $25.00
```

Анализ

В исходном тексте из данного примера приведен отчет с суммами заказов, упорядоченных по номеру заказа.

В регулярном выражении из данного примера сначала применяется шаблон `\d+:`, обеспечивающий совпадение с порядковым номером заказа (если опустить этот шаблон, то с данным регулярным выражением сопоставлялась бы только сумма заказов, а не вся строка заказа в целом, включая и его номер). Шаблон `\$\d{3,}\.\d{2}` обеспечивает совпадение с самой суммой заказа, где `\$` совпадает со знаком `$`, `d{3,}` — с числами, состоящими хотя бы из трех цифр, а следовательно, с суммой не меньше 100 долларов США, `.` — с десятичной точкой, а `\d{2}` — с двумя цифрами после десятичной точки. В итоге регулярное выражение из данного примера правильно обнаруживает совпадение с четырьмя заказами из семи.

Совет

Применяя данную форму интервалов, будьте внимательны. Достаточно опустить в ней запятую (`,`), чтобы проверка изменила свое назначение с минимального на конкретное количество совпадений.

Примечание

Метасимвол `+` функционально равнозначен интервалу `{1,}`.

Предотвращение лишних совпадений

Метасимвол `?` обозначает ограниченное (нулевое или единичное) количество совпадений. То же самое относится и к интервалам, когда в них указываются конкретные величины или диапазоны. Но другие формы повторения, описанные в этом уроке, допускают неограниченное количество совпадений, которое иногда оказывается чрезмерным.

Все представленные до сих пор примеры были тщательно подобраны, чтобы избежать чрезмерных совпадений. Рассмотрим, однако, следующий пример текста с вебстраницы, содержащего встроенные дескрипторы <b> разметки в формате HTML. В регулярном выражении требуется

обнаружить совпадение с любым текстом, заключенным в дескрипторы , возможно, с целью заменить форматирование, как показано ниже.

Текст

```
This offer is not available to customers
living in <b>AK</b> and <b>HI</b>.[4]
```

Регулярное выражение

```
<[Bb]>.*<\/[Bb]>
```

Результат

```
This offer is not available to customers
living in <b>AK</b> and <b>HI</b>.
```

Анализ

В данном примере шаблон <[Bb]> обеспечивает совпадение с открывающим дескриптором разметки , а шаблон </[Bb]> — с закрывающим дескриптором , где буква b может быть как прописной, так и строчной. Но вместо двух отдельных совпадений в данном случае было обнаружено лишь единственное общее совпадение. В частности, шаблон .* обеспечил совпадение с любыми символами после первого дескриптора и вплоть до последнего дескриптора , а следовательно, было обнаружено совпадение с фрагментом разметки AK and HI , включая не только текст, который требовалось найти, но и экземпляры других дескрипторов.

Дело в том, что метасимволы *, . и + *жадны*, т.е. обнаруживают максимально, а не минимально возможное совпадение. В итоге обнаружение совпадения почти всегда начинается не с начала, а с конца текста и продолжается в обратном направлении до тех пор, пока совпадение не будет

[4] Это предложение недоступно клиентам, проживающим на Аляске и Гавайях.

найдено. Это делается преднамеренно, и поэтому подобные кванторы по своему замыслу жадны.

Но что если жадное совпадение не требуется? В таком случае можно воспользоваться ленивыми вариантами упомянутых выше кванторов. Они называются *ленивыми* потому, что обозначают совпадение с минимальным, а не максимальным количеством символов. Ленивые кванторы обозначаются с присоединяемым к ним знаком **?**. У каждого жадного квантора имеется свой ленивый эквивалент, как следует из табл. 5.1.

Таблица 5.1. Жадные и ленивые кванторы

Жадный квантор	Ленивый квантор
`*`	`*?`
`+`	`+?`
`{n,}`	`{n,}?`

А теперь вернем к предыдущему примеру, заменив метасимвол ***** как слишком жадный квантор его ленивым вариантом ***?** в регулярном выражении, как показано ниже.

Текст

```
This offer is not available to customers
living in <b>AK</b> and <b>HI</b>.
```

Регулярное выражение

```
<[Bb]>.*?<\/[Bb]>
```

Результат

```
This offer is not available to customers
living in <b>AK</b> and <b>HI</b>.
```

Анализ

Теперь цель достигнута. Благодаря ленивому квантору ***?** в данном случае удалось обнаружить сначала первое

совпадение с фрагментом разметки AK, а затем второе совпадение с фрагментом разметки HI независимо от первого совпадения.

> **Примечание**
>
> В большинстве примеров из данной книги употребляются жадные кванторы, чтобы сделать шаблоны как можно более простыми. Но вы вольны заменить их ленивыми кванторами по мере надобности.

Резюме

Истинный потенциал регулярных выражений раскрывается в обращении с повторяющимися совпадениями. В этом уроке были представлены метасимволы **+** (обозначает совпадение с единичным или большим количеством символов), ***** (обозначает совпадение с нулевым или большим количеством символов), **?** (обозначает совпадение с нулевым или единичным количеством символов), а также способы обнаружения повторяющихся совпадений. Для более полного контроля могут быть использованы интервалы, обозначающие конкретное, минимальное и максимальное количество совпадений. Упомянутые выше метасимволы являются жадными кванторами, а следовательно, их применение может привести к обнаружению лишних совпадений. Чтобы избежать этого, рекомендуется пользоваться ленивыми аналогами этих кванторов.

Урок 6

Совпадение позиций

Из предыдущих уроков вы узнали, как обнаруживать совпадения с самыми разными символами в различных сочетаниях, повторениях и в любых местах текста. Но иногда требуется обнаружить совпадение в конкретных местах из блока текста, а для данной цели потребуется совпадение позиций, поясняемое в этом уроке.

Назначение границ

Совпадение позиций служит для обозначения в текстовой строке того места, где должно произойти совпадение. Чтобы стала понятнее потребность в совпадении позиций, рассмотрим следующий пример.

Текст

```
The cat scattered his food all over the room.[1]
```

Регулярное выражение

```
cat
```

Результат

```
The cat scattered his food all over the room.
```

[1] Кот разбросал свою еду по всей комнате.

Анализ

Шаблон cat обеспечивает совпадение со всеми вхождениями слова cat, даже если это слово содержится в слове scattered. И хотя это может иногда дать желаемый результат, чаще всего добиться его не удается. Так, если произвести поиск для замены всех вхождений слова cat словом dog, то в конечном счете будет получена следующая бессмыслица:

```
The dog sdogtered his food all over the room.
```

Этот результат приводит к понятию *границ* или специальных метасимволов, обозначающих позицию (т.е. границу) до или после шаблона.

Определение границ слова

Первая (и наиболее употребительная) граница представляет собой границу слова, обозначаемую как \b. Как подразумевает название метасимвола \b, он служит для обозначения совпадения с началом или концом слова. Чтобы продемонстрировать применение метасимвола \b, обратимся снова к предыдущему примеру, задав на этот раз границы слова.

Текст

```
The cat scattered his food all over the room.
```

Регулярное выражение

```
\bcat\b
```

Результат

```
The cat scattered his food all over the room.
```

Анализ

Слово cat отделяется в данном примере пробелами от других слов и поэтому совпадает с шаблоном \bcat\b

(пробел относится к числу символов, применяемых для разделения слов). Но в слове `scattered` совпадение со словом `cat` по данному шаблону не обнаружено, поскольку ему предшествует буква `s`, а после него — буква `t` (ни та, ни другая не совпадает с метасимволом `\b`).

> **Примечание**
>
> Так с чем же конкретно совпадает метасимвол **`\b`**? Механизмы обработки регулярных выражений не понимают ни английского, ни любого другого языка, и поэтому им неизвестны конкретные границы слов. Метасимвол **`\b`** просто обеспечивает совпадение с местоположением в промежутке между символами, которые обычно являются частями слов (буквенно-цифровыми символами и знаком подчеркивания, т.е. текстом, который будет совпадать с метасимволом **`\w`**) и чем-то другим (текстом, который будет совпадать с метасимволом **`\W`**).

В связи с изложенным выше очень важно понять, что для совпадения с целым словом метасимвол `\b` должен быть использован как до, так и после текста, совпадение с которым требуется обнаружить. Рассмотрим следующий пример.

Текст

```
The captain wore his cap and cape proudly as
he sat listening to the recap of how his
crew saved the men from a capsized vessel.[2]
```

Регулярное выражение

```
\bcap
```

Результат

```
The captain wore his cap and cape proudly as
he sat listening to the recap of how his
crew saved the men from a capsized vessel.
```

[2] Капитан гордо восседал в кепке и плаще, слушая краткое изложение рассказа о том, как его команда спасла людей с опрокинутого судна.

Анализ

Шаблон \bcap обеспечивает совпадение с любым словом, начинающимся на cap, и поэтому в данном примере обнаружено совпадение с четырьмя словами, включая три слова, которые, по существу, не являются словом cap.

Ниже приведен тот же самый пример, но только с метасимволом \b в конце регулярного выражения.

Текст

```
The captain wore his cap and cape proudly as
he sat listening to the recap of how his
crew saved the men from a capsized vessel.
```

Регулярное выражение

```
cap\b
```

Результат

```
The captain wore his cap and cape proudly as
he sat listening to the recap of how his
crew saved the men from a capsized vessel.
```

Анализ

Шаблон cap\b обеспечивает совпадение с любым словом, оканчивающимся на cap, и поэтому в данном примере обнаружены два совпадения: одно — само слово cap, а другое — не являющееся этим словом.

Если же требуется обнаружить совпадение только со словом cap, для этой цели следует воспользоваться шаблоном \bcap\b.

> **Примечание**
>
> Метасимвол **\b** обеспечивает совпадение не с конкретным символом, а с определенной позицией. Таким образом, символьная строка, совпадающая с шаблоном **\bcat\b**, будет состоять из трех символов (**c**, **a** и **t**), а не из пяти символов.

Чтобы исключить совпадение на границе слова, следует воспользоваться метасимволом \B. В приведенном выше примере метасимволы \B применяются для того, чтобы обнаружить знаки дефиса с дополнительными пробелами вокруг них.

Текст

```
Please enter the nine-digit id as it
appears on your color - coded pass-key.[3]
```

Регулярное выражение

```
\B-\B
```

Результат

```
Please enter the nine-digit id as it
appears on your color - coded pass-key.
```

Анализ

Шаблон \B-\B обеспечивает совпадение с дефисом, окруженным знаками разрыва слов. Если совпадения с дефисами по данному шаблону в словах `nine-digit` и `pass-key` не происходит, то оно обнаруживается в словосочетании `color - coded`.

Примечание

Если приведенная выше строка кода вам пока еще непонятна, не отчаивайтесь. Вскоре вы поймете ее назначение.

Примечание

В некоторых реализациях регулярных выражений поддерживаются два дополнительных метасимвола. Если метасимвол **\b** обозначает совпадение с началом или концом слова, то метасимвол **\<** — только с началом слова, а метасимвол **\>** —

[3] Введите, пожалуйста, идентификатор цвета из девяти цифр, обозначенный на вашем ключе доступа с цветовой кодировкой.

только с концом слова. И хотя эти метасимволы обеспечивают дополнительный контроль, их поддержка весьма ограничена: они поддерживаются в утилите **egrep**, в отличие от многих других реализаций регулярных выражений.

Определение границ символьных строк

Границы слов служат для обнаружения совпадений, исходя из их позиций в слове (начало слова, конец слова, все слово и т.д.). Аналогичную функцию выполняют границы символьных строк, но они служат для сопоставления с шаблонами в начале или в конце целой строки. Для обозначения границ символьных строк служат следующие метасимволы: **^** — в начале строки, а **$** — в конце строки.

Пример

Из урока 3 вы узнали, что метасимвол ^ служит для того, чтобы обозначить отрицание набора символов. Как же в таком случае воспользоваться им, чтобы обозначить начало символьной строки?

Метасимвол ^ относится к числу тех нескольких метасимволов, которые имеют не одно назначение. Этот метасимвол обозначает отрицание набора символов только в том случае, если в наборе, заключаемом в квадратные скобки [и], он указан первым после открывающей квадратной скобки [. А если метасимвол ^ указан за пределами набора символов и в начале шаблона, то он обозначает совпадение с началом символьной строки.

Чтобы продемонстрировать применение границ символьных строк, рассмотрим следующий пример. Достоверные XML-документы начинаются с дескриптора разметки <?xml>, в котором, вероятнее всего, имеются дополнительные атрибуты (например, номер версии в дескрипторе <xml version="1.0" ?>). Ниже приведена простая проверка на соответствие исходного текста формату XML-документа.

Текст

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

Регулярное выражение

```
<\?xml.*\?>
```

Результат

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

Анализ

Регулярное выражение из данного примера кажется, на первый взгляд, вполне работоспособным. В частности, шаблон `<\?xml` обнаруживает совпадение с началом дескриптора `<?xml`, шаблон `.*` — нулевое или большее количество совпадений с любым другим текстом, а шаблон `\?>` — совпадение с концом дескриптора `?>`. Тем не менее проверка в данном примере весьма неточна.

Рассмотрим еще один пример, в котором то же самое регулярное выражение применяется для обнаружения совпадения с лишним текстом перед дескриптором разметки, открывающим XML-документ.

Текст

```
This is bad, real bad![4]
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

[4] Это никуда не годится!

Регулярное выражение

```
<\?xml.*\?>
```

Результат

```
This is bad, real bad!
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

Анализ

В данном примере по шаблону <\?xml.*\?> было обнаружено совпадение со второй строкой исходного текста. И хотя дескриптор разметки, открывающий XML-документ, действительно находится во второй строке исходного текста, данный пример совершенно неверный, а обработка текста как XML-документа может вызвать всевозможные осложнения.

В данном случае необходимо проверить, действительно ли дескриптор разметки, открывающий XML-документ, находится в первой строке исходного текста. И для этой цели идеально подходит метасимвол ^, как показано ниже.

Текст

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

Регулярное выражение

```
^\s*<\?xml.*\?>
```

Результат

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://tips.cf"
xmlns:impl="http://tips.cf" xmlns:intf="http://tips.cf"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

Анализ

Как упоминалось ранее, метасимвол ^ обозначает совпадение с началом символьной строки, поэтому шаблон `^\s*` обеспечивает совпадение с началом символьной строки, после которой следует от нуля и больше пробельных символов. Следовательно, вполне допустимые знаки пробела, табуляции или разрыва строки правильно обрабатываются перед дескриптором разметки, открывающим XML-документ. Таким образом, все регулярное выражение `^\s*<\?xml.*\?>` в целом позволяет обнаружить открывающий XML-документ дескриптор разметки с любыми атрибутами и правильно обработать пробелы.

Совет

Шаблон **`^\s*<\?xml.*\?>`** оказался вполне работоспособным, но только потому, что XML-документ приведен в рассмотренном выше примере не полностью. Если бы он был приведен полностью, то его обработка послужила бы наглядным примером действия жадного квантора. Кроме того, этот пример наглядно показал бы, когда вместо шаблона **`.*`** следует использовать шаблон **`.*?`**.

Аналогичным образом применяется и метасимвол **$**. С его помощью можно проверить, что после дескриптора разметки </html>, закрывающего веб-страницу, ничего больше не следует, как показано ниже.

Регулярное выражение

```
</[Hh][Tt][Mm][Ll]>\s*$
```

Анализ

Для обнаружения символов H, T, M и L в данном регулярном выражении употребляются наборы соответствующих символов, чтобы можно было обработать любое сочетание прописных и строчных букв, а шаблон `\s*$` обеспечивает совпадение с любыми пробельными символами, которые могут следовать до конца строки.

Примечание

Шаблон `^.*$` является синтаксически правильно составленным регулярным выражением. По данному шаблону практически всегда обнаруживается совпадение, и поэтому он совершенно бесполезен. Попробуйте сами определить, что можно и чего нельзя обнаружить по данному шаблону.

Применение многострочного режима

Как пояснялось ранее, метасимвол **^** обозначает, как правило, совпадение с началом символьной строки, а метасимвол **$** — с ее концом. Но из этого правила имеется исключение или, скорее, способ изменить обычное поведение обоих метасимволов.

Во многих реализациях регулярных выражений поддерживается применение специальных метасимволов, модифицирующих поведение других метасимволов. К их числу относится метасимвол `(?m)`, активизирующий многострочный режим. В многострочном режиме механизм обработки регулярных выражений принудительно интерпретирует разрывы строк как разделители символьных строк. Следовательно, метасимвол **^** обнаруживает совпадение с началом символьной строки как непосредственно, так и после разрыва текущей строки (в начале новой строки), а метасимвол **$** — совпадение с концом символьной строки как непосредственно, так и после разрыва текущей строки.

Если модификатор `(?m)` применяется в регулярном выражении, он указывается в самом начале шаблона, как показано в приведенном ниже примере, в котором с помощью регулярного выражения обнаруживаются все комментарии в блоке кода JavaScript.

Текст

```
<script>
function doSpellCheck(form, field) {
  // убедиться, что значение переменной не пустое
  if (field.value == '') {
    return false;
  }
```

```
  // инициализировать переменную
  var windowName='spellWindow';
  var spellCheckURL=
      'spell.cfm?formname=comment&fieldname='+field.
name;
  ...
  // готово!
  return false;
}
</script>
```

Регулярное выражение

```
(?m)^\s*\/\/.*$
```

Результат

```
<script>
function doSpellCheck(form, field) {
  // убедиться, что значение переменной не пустое
  if (field.value == '') {
    return false;
  }
  // инициализировать переменную
  var windowName='spellWindow';
  var spellCheckURL=
      'spell.cfm?formname=comment&fieldname='+field.
name;
  ...
  // готово!
  return false;
}
</script>
```

Анализ

Шаблон `^\s` в данном примере обеспечивает совпадение с началом символьной строки, а далее — с любыми пробельными символами. По шаблону `\/\/` определяются комментарии в блоке кода JavaScript, после которых следует любой текст до конца строки. Но такой шаблон обеспечил бы совпадение только с первым комментарием, да и то лишь в том случае, если бы он был единственным текстом на странице.

А модификатор (?m), применяемый в регулярном выражении (?m)^\s*\/\/.*$, вынуждает механизм его обработки интерпретировать разрывы строк как разделители символьных строк, и благодаря этому обнаруживается совпадение со всеми комментариями в блоке кода из данного примера.

Предупреждение

Модификатор **(?m)** не поддерживается во многих реализациях регулярных выражений, включая и JavaScript.

Примечание

В некоторых реализациях регулярных выражений поддерживается метасимвол **\A** для обозначения начала символьной строки, а также метасимвол **\Z** для обозначения конца символьной строки. Если эти метасимволы поддерживаются, они функционируют подобно метасимволам **^** и **$** соответственно. Но в отличие от них поведение метасимволов **\A** и **\Z** не видоизменяется модификатором **(?m)**, и поэтому они не будут действовать в многострочном режиме.

Резюме

Регулярные выражения позволяют обнаруживать совпадения с любыми блоками или конкретными местами текста в символьной строке. В частности, метасимвол \b служит для обозначения границы слова, а метасимвол \B делает совершенно противоположное. Метасимволы **^** и **$** обозначают границы символьной строки (ее начало и конец соответственно). Но если они применяются вместе с модификатором (?m), то обозначают также совпадение с символьными строками, начинающимися или оканчивающимися разрывом строки.

Урок 7

Применение подвыражений

Метасимволы и способы организации совпадения с символами составляют основной потенциал регулярных выражений, как было продемонстрировано в предыдущих уроках. А из этого урока вы узнаете, как группировать выражения, используя подвыражения.

Общее представление о подвыражениях

О совпадении с несколькими вхождениями символа речь шла в уроке 5. В частности, шаблон `\d+` обеспечивает совпадение с одним или несколькими цифрами, а шаблон `https?://` — с префиксом `http://` или `https://`. В обоих случаях (и, конечно, во всех представленных до сих пор примерах) метасимволы повторения (например, `?`, `*` или `{2}`) применяются к предыдущему символу или метасимволу.

Разработчики HTML-разметки документов нередко размещают неразрывные пробелы (с помощью специального кода разметки ` `) между словами, чтобы текст между этими словами не был автоматически перенесен на новую строку. Допустим, что требуется обнаружить все неразрывные пробелы, повторяющиеся в HTML-разметке документа, чтобы заменить их чем-то другим, как показано в следующем примере.

Текст

```
Hello, my name is Ben Forta, and I am
the author of multiple books on SQL (including
MySQL, Oracle PL/SQL, and SQL Server T-SQL),
Regular  Expressions, and other subjects.[1]
```

Регулярное выражение

```
 {2,}
```

Результат

```
Hello, my name is Ben Forta, and I am
the author of multiple books on SQL (including
MySQL, Oracle PL/SQL, and SQL Server T-SQL),
Regular  Expressions, and other subjects.
```

Анализ

Специальный код разметки ` ` обозначает ссылку на сущность для неразрывных пробелов в HTML-документе. Шаблон ` {2,}` должен был обеспечить совпадение с двумя или больше экземплярами ссылки на сущность ` `, но этого так и не произошло. Почему? А потому что интервал `{2,}` обозначает количество совпадений с тем, что ему непосредственно предшествует (в данном случае — со знаком точки с запятой). Следовательно, он должен обеспечить совпадение с фрагментом текста ` ;;;;`, а не `  `. Данное затруднение приводит к понятию подвыражений, из которых состоят более крупные выражения.

Группирование подвыражений

Подвыражения как составные части более крупных выражений могут быть сгруппированы, чтобы интерпретировать

[1] Здравствуйте! Меня зовут Бен Форта, я — автор ряда книг по SQL, включая MySQL, Oracle PL/SQL и SQL Server T-SQL, регулярным выражениям и другим предметам.

их как единое целое. С этой целью подвыражения заключаются в круглые скобки (и).

Совет

Круглые скобки (и) являются метасимволами. Чтобы обеспечить совпадение непосредственно со знаками круглых скобок, необходимо экранировать их соответственно: \(и \).

Обратимся снова к предыдущему примеру, чтобы продемонстрировать применение подвыражений.

Текст

```
Hello, my name is Ben Forta, and I am
the author of multiple books on SQL (including
MySQL, Oracle PL/SQL, and SQL Server T-SQL),
Regular  Expressions, and other subjects.
```

Регулярное выражение

```
( ){2,}
```

Результат

```
Hello, my name is Ben Forta, and I am
the author of multiple books on SQL (including
MySQL, Oracle PL/SQL, and SQL Server T-SQL),
Regular  Expressions, and other subjects.
```

Анализ

В регулярном выражении из данного примера ( ) обозначает подвыражение, интерпретируемое как единое целое. Таким образом, следующий после него интервал {2,} применяется ко всему подвыражению, а не только к знаку точки с запятой. Тем самым достигается поставленная ранее цель.

Рассмотрим еще один пример, в котором на этот раз регулярное выражение применяется для обнаружения IP-адресов. Такие адреса обычно представлены в формате из четырех наборов чисел, разделяемых точками, как,

например 12.159.46.200. Каждое из чисел может состоять из одной, двух или трех цифр, и поэтому шаблон для сопоставления с каждым числом в IP-адресе может быть выражен как \d{1,3}, что и демонстрируется в следующем примере.

Текст

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:[2]
```

Регулярное выражение

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Результат

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Анализ

Каждый экземпляр шаблона \d{1,3} обеспечивает в данном примере совпадение с одним из чисел в IP-адресе. Четыре числа этого адреса разделяются знаками точки, которые соответственно экранируются: \..

Шаблон \d{1,3}\., обеспечивающий совпадение с цифрами в количестве от одной до трех, а также со знаком точки, повторяется в приведенном выше регулярном выражении три раза, а следовательно, он может быть также выражен как повторение. Ниже приведен альтернативный вариант предыдущего примера.

Текст

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

[2] Эхо-тестирование веб-сайта по адресу hog.forta.com [12.159.46.200] пакетом данных из 32 байтов:

Регулярное выражение

```
(\d{1,3}\.){3}\d{1,3}
```

Результат

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Анализ

Шаблон в данном примере действует таким же образом, как и в предыдущем примере, но синтаксис его иной. В частности, выражение `\d{1,3}\.` заключено в круглые скобки `(` и `)`, чтобы превратить его в подвыражение. Таким образом, шаблон `(\d{1,3}\.){3}` обеспечивает троекратное повторение подвыражения для сопоставления с первыми тремя числами в IP-адресе, а следующий далее шаблон `\d{1,3}` — совпадение с последним числом этого адреса.

Примечание

Шаблон **`(\d{1,3}\.){4}`** нельзя считать равноценной альтернативой для рассмотренного выше шаблона. Попробуйте сами определить, почему этот шаблон не годится для данного примера.

Совет

Некоторые пользователи предпочитают заключать в круглые скобки отдельные составляющие выражений, выделяя их как подвыражения, чтобы повысить удобочитаемость регулярных выражений. Так, рассмотренный выше пример можно выразить следующим образом: **`(\d{1,3}\.){3}(\d{1,3})`**. Такая норма практики вполне допустима и не оказывает никакого влияния на фактическое поведение выражения в частности и производительность в целом, хотя это зависит от применяемой реализации регулярных выражений.

Группирование подвыражений оказывается настолько важным, что стоит рассмотреть еще один пример, но на этот раз вообще без повторений. В данном примере

предпринимаются попытки обнаружить совпадение с годом записи о пользователе.

Текст

```
ID: 042
SEX: M
DOB: 1967-08-17
Status: Active[3]
```

Регулярное выражение

```
19|20\d{2}
```

Результат

```
ID: 042
SEX: M
DOB: 1967-08-17
Status: Active
```

Анализ

В данном примере шаблон служит для того, чтобы обнаружить совпадение с годом, состоящим из четырех цифр, но для большей точности первые две цифры года явно перечислены как `19` или `20`, а расположенный между ними знак `|` обозначает логическую операцию ИЛИ. Таким образом, шаблон `19|20` обеспечивает совпадение с цифрами `19` или `20`, а следовательно, шаблон `19|20\d2` должен был обеспечить совпадение с любым числом, состоящим из четырех цифр, начиная с цифр `19` или `20` и еще двух цифр. Очевидно, что это не сработало. Дело в том, что при выполнении логической операции `|` анализируется ее левый и правый операнды, и поэтому шаблон `19|20\d{2}` воспринимается как операнд `19` или операнд `20\d{2}`, если считать шаблон

3 Идентификатор: 042
ПОЛ: M
Дата рождения:17.08.1967
Состояние: Активный

\d{2} частью выражения, начинающегося с цифр 20. Иными словами, совпадение произойдет с цифрами 19 или с любым годом, состоящим из четырех цифр и начинающимся с цифр 20. Поэтому и было обнаружено совпадение только с цифрами 19 в годе из даты рождения пользователя.

Чтобы разрешить возникшее затруднение, следует сгруппировать шаблон 19|20 в подвыражение, как показано ниже.

Текст

```
ID: 042
SEX: M
DOB: 1967-08-17
Status: Active
```

Регулярное выражение

```
(19|20)\d{2}
```

Результат

```
ID: 042
SEX: M
DOB: 1967-08-17
Status: Active
```

Анализ

Если все возможные варианты выбора сгруппированы в подвыражение, то при выполнении логической операции | станет известно, что требуется выбрать один из этих вариантов. Таким образом, шаблон (19|20)\d{2} правильно обеспечивает совпадение с годом 1967, т.е. с любыми четырьмя цифрами года, начиная с цифр 19 или 20. Для обработки более поздних дат (например, через сотню лет после текущей даты) достаточно внести соответствующие коррективы в шаблон. Так, чтобы обеспечить совпадение с годом, начинающимся с цифр 21, потребуется шаблон (19|20|21)\d{2}.

Примечание

Несмотря на то что в данном уроке рассматривается применение подвыражений для группирования, им находится еще одно очень важное применение. Подробнее об этом речь пойдет в уроке 8.

Вложение подвыражений

Подвыражения допускают вложение. На самом деле одни подвыражения могут быть вложены в другие, а те — в третьи и т.д. Возможность вложения подвыражений позволяет составлять невероятно эффективные выражения, но в то же время они могут стать запутанными, неудобочитаемыми, непонятными и в какой-то степени отпугивающими. Но в действительности вложенные подвыражения редко бывают настолько сложными, насколько они таковыми выглядят.

Обратимся снова к примеру с IP-адресами, чтобы продемонстрировать применение вложенных подвыражений. Ниже приведено регулярное выражение, применявшееся в предыдущем примере. Подвыражение повторяется в нем три раза, а затем следует шаблон для сопоставления с последним числом в IP-адресе.

Регулярное выражение

```
(\d{1,3}\.){3}\d{1,3}
```

Что же не так в этом регулярном выражении? Синтаксически оно составлено верно. IP-адрес действительно состоит из четырех чисел, а каждое из них содержит от одной до трех цифр и отделяется от других чисел точкой. Следовательно, шаблон составлен правильно и обеспечивает совпадение с любым достоверным IP-адресом. Но это еще не все. Ведь по данному шаблону совпадут и недостоверные IP-адреса.

Обычный IP-адрес состоит из четырех байтов, как, например, представленный ранее IP-адрес `12.159.46.200`. Следовательно, четыре числа в IP-адресе могут быть представлены значениями в пределах от `0` до `255`, т.е. одного

байта. Это означает, что ни одно из чисел в IP-адресе не может быть больше 255. А по упомянутому выше шаблону будет обнаружено совпадение и с такими числами, как, например, 345, 700 и 999, которые недопустимы в IP-адресе.

Примечание

Из сказанного выше можно извлечь очень важный урок. Написать регулярные выражения для совпадения с тем, что требуется и предполагается, нетрудно. Но намного труднее написать такие регулярные выражения, в которых предусмотрены все возможные случаи, чтобы не обнаруживать совпадение с тем, что не требуется.

Было бы, конечно, неплохо указать в рассматриваемом здесь примере диапазон достоверных значений, но ведь регулярные выражения обнаруживают совпадение с символами, не имея ни малейшего представления, чем эти символы являются на самом деле. Следовательно, математические вычисления в данном случае не годятся.

Есть ли из этого затруднения выход? Возможно, есть. Чтобы составить регулярное выражение, необходимо ясно определить, что именно требуется обнаружить и чего не следует обнаруживать при сопоставлении с шаблоном. Ниже приведены правила, определяющие достоверные сочетания каждого из чисел в IP-адресе.

- Любое число, состоящее из одной или двух цифр.
- Любое число, состоящее из трех цифр, начиная с 1.
- Любое число, состоящее из трех цифр, начиная с 2, если вторая цифра оказывается в пределах от 0 до 4.
- Любое число, состоящее из трех цифр, начиная с 25, если третья цифра оказывается в пределах от 0 до 5.

Если последовательно разобрать поставленную задачу подобным образом, то станет ясно, что на самом деле имеется вполне работоспособный шаблон. Ниже приведен соответствующий пример.

Текст

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Регулярное выражение

```
(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2}))\.){3}
(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2})))
```

Результат

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Анализ

Очевидно, что регулярное выражение в данном примере вполне работоспособно, тем не менее оно требует дополнительных пояснений. Работоспособность данного регулярного выражения обеспечивает последовательный ряд вложенных подвыражений. Начнем его анализ с шаблона `(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2}))\.)`, состоящего из четырех вложенных подвыражений, рассмотрев их в обратном порядке. В частности, подвыражение `(\d{1,2})` обеспечивает совпадение с любым числом, состоящим из одной или двух цифр, т.е. с числами от 0 до 99. Подвыражение `(1\d{2})` обеспечивает совпадение с любым числом, состоящим из трех цифр, начиная с 1 (т.е. из 1 и любых двух последующих цифр), а следовательно, с числами от 100 до 199. Подвыражение `(2[0-4]\d)` обеспечивает совпадение с числами от 200 до 249, а подвыражение `(25[0-5])` — с числами от 250 до 255. Каждое из этих подвыражений вложено в другое подвыражение и разделяется знаком | логической операции ИЛИ, и поэтому совпадение будет обнаружено не со всеми, а только с одним из четырех подвыражений. После четырех подвыражений, определяющих диапазон допустимых чисел в IP-адресе, следует шаблон `\.`, обеспечивающий совпадение со знаком точки, а далее — вся последовательность описанных выше

подвыражений вложена вместе с шаблоном \. в еще одно подвыражение и повторяется три раза подряд с помощью интервала {3}. И наконец, весь диапазон допустимых чисел повторяется снова, но на этот раз без завершающего шаблона \., чтобы обеспечить совпадение с последним числом в IP-адресе. Ограничивая каждое из четырех чисел значениями в пределах от 0 до 255, данное регулярное выражение действительно позволяет обнаружить совпадение с достоверными IP-адресами, отвергнув недостоверные адреса.

Нужно, однако, иметь в виду, что более логичный порядок следования четырех подвыражений, чем описанный выше, не годится. Чтобы убедиться в этом, рассмотрим следующий пример.

Текст

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Регулярное выражение

```
(((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))\.){3}
((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5]))
```

Результат

```
Pinging hog.forta.com [12.159.46.200]
with 32 bytes of data:
```

Анализ

Обратите внимание, что на этот раз последняя цифра 0 в IP-адресе не совпала с шаблоном. Дело в том, что подвыражения вычисляются слева направо, и если имеется четыре подвыражения, каждое из которых может обнаружить совпадение, то сначала проверяется первое из них, затем — второе и т.д. И как только обнаруживается совпадение с шаблоном в любом подвыражении, остальные даже не проверяются. В данном примере подвыражение (\d{1,2}) обнаруживает совпадение с цифрами 20 в последнем числе

200, и поэтому остальные подвыражения, в том числе подвыражение (25[0-5]), которое в данном случае обязательно, вообще не вычисляются.

> **Совет**
>
> Подобные регулярные выражения могут выглядеть непомерно сложными. Ключ к их пониманию лежит в разбиении их на части и последовательном анализе отдельных подвыражений. Начиная изнутри регулярного выражения, следует продвигаться наружу, пытаясь разобрать его посимвольно от начала и до конца. И сделать это не так сложно, как кажется на первый взгляд.

Резюме

Подвыражения применяются для группирования отдельных частей выражения и определяются с помощью круглых скобок (и). К типичным примерам применения подвыражений относится возможность точно контролировать повторяемость шаблонов с помощью метасимволов повторения, а также правильно определять условия в логической операции ИЛИ. Подвыражения допускают вложение, если в этом есть потребность.

Урок 8

Применение обратных ссылок

В предыдущем уроке были представлены подвыражения как средство для группирования символов в наборы. Основное назначение такого рода группирования состоит в возможности правильно определить последовательный ряд повторяющихся сопоставлений с шаблонами, как было продемонстрировано в предыдущем уроке. А в этом уроке будет рассмотрено еще одно важное применение подвыражений для манипулирования обратными ссылками.

Общее представление об обратных ссылках

Пояснить потребность в обратных ссылках лучше всего на конкретном примере. Разработчики HTML-документов пользуются дескрипторами разметки заголовков (<h1>–<h6> с соответствующими конечными дескрипторами) для того, чтобы определить формат текста заголовков на веб-страницах. Допустим, что требуется найти текст всех заголовков независимо от их уровня, как демонстрируется в следующем примере.

Текст

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
```

```
<h2>RegEx</h2>
Information about Regular Expressions.
</body>[1]
```

Регулярное выражение

```
<[hH]1>.*<\/[hH]1>
```

Результат

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
</body>
```

Анализ

В данном примере по шаблону `<[hH]1>.*<\/[hH]1>` было обнаружено совпадение с первым заголовком, заключенным между дескрипторами разметки `<h1>` и `</h1>`. Кроме того, может быть обнаружено совпадение с дескриптором разметки `<H1>`, поскольку в формате HTML регистр букв не учитывается. Но какой шаблон потребуется, чтобы обнаружить совпадение с *любым* заголовком, расположенным на одном из шести допустимых уровней? Одно из возможных решений этой задачи может состоять в применении простого диапазона вместо 1, как показано ниже.

---

[1]
```
<body>
<h1>Добро пожаловать на мою начальную страницу</h1>
Ее содержимое разделено на два раздела:<br/>
<h2>SQL</h2>
Сведения об языке SQL.
<h2>RegEx</h2>
Сведения о регулярных выражениях.
</body>
```

Текст

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
</body>
```

Регулярное выражение

```
<[hH][1-6]>.*?<\/[hH][1-6]>
```

Результат

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
</body>
```

Анализ

Цель, по-видимому, достигнута. Шаблон `<[hH][1-6]>` обеспечивает совпадение с начальным дескриптором разметки любого заголовка (в данном случае — `<h1>` и `<h2>`), а шаблон `<\/[hH][1-6]>` — с конечным дескриптором разметки любого заголовка (в данном случае — `</h1>` и `</h2>`).

> **Примечание**
>
> Обратите внимание на то, что в данном примере был применен ленивый квантор **`*?`** вместо жадного квантора **`*`** в шаблоне **`.*?`**. Как пояснялось в уроке 5, такие кванторы, как **`*`**, являются жадными, а следовательно, по шаблону **`<[hH][1-6]>.*<\/[hH][1-6]>`** могло быть обнаружено все содержимое, начиная с открывающего дескриптора разметки **`<h1>`** во второй строке и заканчивая закрывающим дескриптором раз

метки **</h2>** в шестой строке. Этого недостатка удалось избежать благодаря применению ленивого квантора * в шаблоне **.*?**.

Упомянутое выше излишнее совпадение *могло*, но не *должно* было быть обнаружено, поскольку в данном конкретном примере регулярное выражение, вероятно, оказалось бы вполне работоспособным и с жадным квантором. Как правило, метасимвол . не обозначает совпадение с разрывами строк, а в данном примере каждый заголовок находится в отдельной строке. Но в применении здесь ленивого квантора нет ничего плохого, поскольку лучше перестраховаться на всякий пожарный случай.

Итак, цель в данном примере можно считать достигнутой? Не совсем. Чтобы убедиться в этом, рассмотрим следующий пример с тем же самым регулярным выражением.

Текст

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>[2]
</body>
```

Регулярное выражение

```
<[hH][1-6]>.*?<\/[hH][1-6]>
```

Результат

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
```

[2] <h2>Это неверная HTML-разметка</h3>

```
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>
</body>
```

Анализ

Разметка последнего заголовка начинается с дескриптора <h2> и заканчивается дескриптором </h3>, а следовательно, она неверна. Тем не менее она совпала с применяемым в данном примере шаблоном.

Дело в том, что вторая часть шаблона, в которой осуществляется сопоставление с конечным дескриптором разметки, никак не связана с первой его частью, в которой осуществляется сопоставление с начальным дескриптором разметки. Именно здесь и приходят на помощь обратные ссылки.

Совпадение с обратными ссылками

Мы еще вернемся к упомянутому выше недостатку обнаружения заголовков на веб-странице. А пока рассмотрим более простой пример, в котором нельзя никак разрешить возникающий недостаток без применения обратных ссылок.

Допустим, что имеется блок текста и требуется обнаружить все повторяющиеся слова (опечатки, где одно и то же слово ошибочно набрано дважды). Очевидно, что при обнаружении второго вхождения слова оно должно быть уже известно. Обратные ссылки позволяют обращаться к предыдущим совпадениям из шаблонов регулярных выражений (в данном случае — к совпавшему ранее слову).

Пояснить механизм действия обратных ссылок лучше всего на конкретном примере, поэтому ниже приведен исходный текст, содержащий три ряда повторяющихся слов, все из которых требуется обнаружить.

Текст

```
This is a block of of text,
several words here are are
```

```
repeated, and and they
should not be.[3]
```

Регулярное выражение

```
[ ]+(\w+)[ ]+\1
```

Результат

```
This is a block of of text,
several words here are are
repeated, and and they
should not be.
```

Анализ

Очевидно, что регулярное выражение в данном примере сработало исправно, но как? Шаблон `[ ]+` обеспечивает совпадение с одним или больше пробелами, шаблон `\w+` — с одним или больше буквенно-цифровыми символами, а шаблон `[ ]+` — с завершающими пробелами. Но обратите внимание на то, что шаблон `\w+` заключен в круглые скобки, а следовательно, превращен в подвыражение. Это подвыражение служит не для обнаружения повторяющихся совпадений, поскольку такие совпадения в данном примере отсутствуют, а просто для группирования выражения с целью обозначить его как таковое для дальнейшего распознавания и применения. И завершает данное регулярное выражение обратная ссылка `\1`, обеспечивающая совпадение с тем же самым текстом, что и в первой совпавшей группе. Так, если подвыражение `(\w+)` обнаружит совпадение со словом `of`, то оно будет обнаружено и по обратной ссылке `\1`. А если подвыражение `(\w+)` обнаружит совпадение со словом `and`, то оно будет обнаружено и по обратной ссылке `\1`.

[3] Это блок текста, в котором несколько слов (of, are и and) повторяются, хотя и не должны повторяться.

Примечание

Термин *обратная ссылка* обозначает то обстоятельство, что одни сущности ссылаются *обратно* на другие в предыдущем выражении.

Что же конкретно обозначает обратная ссылка `\1`? Она обозначает совпадение с первым подвыражением в шаблоне, обратная ссылка `\2` — совпадение со вторым подвыражением, обратная ссылка `\3` — совпадение с третьим подвыражением и т.д. Таким образом, шаблон `[ ]+(\w+) [ ]+\1` обеспечивает совпадение сначала с любым словом, а затем с тем же самым словом снова, что и было продемонстрировано в приведенном выше примере.

Предупреждение

К сожалению, синтаксис обратных ссылок заметно отличается в разных реализациях регулярных выражений. Так, в JavaScript, как, впрочем, и в текстовом редакторе **vi**, знак **** служит для обозначения обратной ссылки, за исключением операций замены, в которых применяется знак **$**. А в языке Perl для этой же цели применяется знак **$**, и поэтому вместо **\1** обратная ссылка на первое выражение в шаблоне обозначается как **$1**. На платформе .NET регулярные выражения поддерживаются таким образом, чтобы возвращать объект со свойством **Groups**, содержащим обнаруженные совпадения, и поэтому обратная ссылка **match.Groups[1]** делается на первое совпадение, обнаруженное на языке C#, а обратная ссылка **match.Groups(1)** — на то же самое совпадение, обнаруженное на языке Visual Basic. Аналогичная информация в PHP возвращается в массиве **$matches**, и поэтому обратная ссылка **$matches[1]** делается на первое обнаруженное совпадение, хотя такое поведение можно изменить, установив соответствующие признаки. И наконец, в Java и Python возвращается объект совпадения, содержащий массив **group**. Особенности отдельных реализаций регулярных выражений перечислены в приложении к данной книге.

Совет

Обратные ссылки можно рассматривать как подобные переменным.

Итак, рассмотрев особенности применения обратных ссылок, вернемся снова к примеру с заголовками в формате HTML. Используя обратные ссылки, можно составить шаблон, обеспечивающий совпадение с любым начальным и конечным дескрипторами разметки заголовка, игнорируя любые пары несовпадений, как демонстрируется в приведенном ниже примере.

Текст

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>
</body>
```

Регулярное выражение

```
<[hH]([1-6])>.*?<\/[hH][1-6]\1>
```

Результат

```
<body>
<h1>Welcome to my Homepage</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>
</body>
```

Анализ

И в данном примере обнаружены три совпадения: одно — с парой дескрипторов разметки <h1> и два — с парами дескрипторов разметки <h2>. Как и прежде, шаблон <[hH]([1-6])> обеспечивает совпадение с любым начальным

дескриптором разметки заголовка. Но, в отличие от прежнего, интервал `[1-6]` заключен в круглые скобки, а следовательно, он превращен в подвыражение. Таким образом, шаблон для сопоставления с конечным дескриптором разметки заголовка может ссылаться на данное подвыражение, как это делается по обратной ссылке `\1` в шаблоне `<\/[hH]\1>`. При этом подвыражение `([1-6])` обеспечивает совпадение с цифрами от 1 до 6, а обратная ссылка `\1` — только с той же самой цифрой. Именно поэтому в данном примере и не было обнаружено совпадение с неверной разметкой заголовка `<h2>This is not valid HTML</h3>`.

Предупреждение

Обратные ссылки будут действовать только в том случае, если они делаются на подвыражение, заключенное в круглые скобки.

Совет

Совпадения обычно обозначаются, начиная с **1**. Но во многих реализациях регулярных выражений обратно ссылаться на подвыражение можно, начиная с **0**.

Примечание

Как было показано в приведенных выше примерах, обратные ссылки на подвыражения делаются по их относительным позициям: **\1** — на первое подвыражение, **\5** — на пятое подвыражение и т.д. И хотя такой синтаксис обратных ссылок находит всеобщую поддержку, ему присущ следующий серьезный недостаток: переместив или поправив подвыражения, а следовательно, изменив их порядок расположения, можно в конечном счете нарушить весь шаблон. Еще труднее вводить и удалять подвыражения.

Для устранения подобного недостатка в ряде новых реализаций регулярных выражений поддерживается *именованный захват* — средство, позволяющее присвоить каждому подвыражению однозначное имя, по которому можно в дальнейшем обращаться к подвыражению вместо относительной его позиции в шаблоне. Именованный захват в данной книге не рассматривается потому, что он по-прежнему не находит широкой

поддержки, а его синтаксис заметно отличается в тех реализациях регулярных выражений, в которых он поддерживается. Но если в той реализации, которой вы пользуетесь, именованный захват поддерживается (например, на платформе .NET), непременно воспользуйтесь преимуществами его функциональных возможностей.

Выполнение операций замены

Каждое из рассмотренных до сих пор регулярных выражений применялось для поиска, т.е. для обнаружения конкретного текста в более крупном блоке текста. Разумеется, большинство регулярных выражений пишутся именно для целей поиска нужного текста. Но этим применение регулярных выражений не отграничивается. Их можно использовать и для выполнения эффективных операций замены текста.

Для простой замены текста регулярные выражения не требуются. Они, например, не нужны для того, чтобы заменить все экземпляры сокращения `CA` названием штата `California` или `MI` названием штата `Michigan`. И хотя такую операцию вполне допустимо реализовать с помощью регулярного выражения, ее ценность сомнительна, а на самом деле проще и быстрее воспользоваться для той же самой цели имеющимися в наличии обычными функциями манипулирования символьными строками.

Операции замены с помощью регулярных выражений стали привлекательными с тех пор, когда появилась возможность пользоваться обратными ссылками. Обратимся снова к примеру из урока 5.

Текст

```
Hello, ben@forta.com is my email address.
```

Регулярное выражение

```
\w+[\w\.]*@[\w\.]+\.\w+
```

Результат

```
Hello, ben@forta.com is my email address.
```

Анализ

По шаблону из данного примера в блоке исходного текста обнаруживаются адреса электронной почты, как пояснялось в уроке 5.

Но что если требуется сделать любые адреса электронной почты доступными по гиперссылке? В формате HTML можно, например, воспользоваться разметкой `<a href="mailto:user@address.com">user@address.com</a>`, чтобы создать адрес электронной почты, активизируемый щелчком кнопкой мыши. А можно ли преобразовать исходный адрес электронной почты в подобную его форму, используя регулярное выражение? Можно и очень просто, если, конечно, воспользоваться обратными ссылками, как показано ниже.

Текст

```
Hello, ben@forta.com is my email address.
```

Регулярное выражение

```
(\w+[\w\.]*@[\w\.]+\.\w+)
```

Замена

```
<a href="mailto:$1">$1</a>
```

Результат

```
Hello, <a href="mailto:$1">$1</a> is my email address.
```

Анализ

В операциях замены применяются два регулярных выражения: одно — для обозначения шаблона поиска, а второе — для обозначения того текста, который должен быть

заменен совпавшим текстом. Обратные ссылки могут соединять вместе шаблоны, и поэтому результат совпадения с подвыражением в первом шаблоне может быть использован во втором шаблоне. В данном примере используется тот же самый шаблон для обнаружения адреса электронной почты, что и в предыдущем примере, но на этот раз он указан в виде подвыражения (`\w+[\w\.]*@[\w\.]+\.\w+`). Таким образом, текст, совпавший по данному шаблону поиска, может быть использован в шаблоне замены. Так, в шаблоне замены `<a href="mailto:$1">$1</a>` результат совпадения с подвыражением из шаблона поиска используется дважды: один раз — в атрибуте `href` для определения ссылки `mailto:`, а другой раз — в виде текста, активизируемого щелчком кнопкой мыши. Таким образом, исходный адрес электронной почты `ben@forta.com` превращается в гиперссылку `<a href="mailto:ben@forta.com">ben@forta.com</a>`, что, собственно, и требовалось в данном примере.

Предупреждение

Как отмечалось ранее, обозначение обратной ссылки должно быть изменено в соответствии с применяемой реализацией регулярных выражений. Например, программирующим на JavaScript придется воспользоваться знаком **$** вместо знака **** в приведенном выше примере.

Совет

Как демонстрировалось в предыдущем примере, к подвыражению можно обращаться неоднократно, просто используя обратную ссылку по мере надобности.

Рассмотрим еще один пример. Сведения о пользователях обычно хранятся в базе данных, а номера их телефонов — в формате `313-555-1234`. Но допустим, что этот формат требуется изменить на `(313) 555-1234`. Ниже показано, как это делается с помощью регулярных выражений.

Текст

```
313-555-1234
248-555-9999
810-555-9000
```

Регулярное выражение

```
(\d{3})(-)(\d{3})(-)(\d{4})
```

Замена

```
($1) $3-$5
```

Результат

```
(313) 555-1234
(248) 555-9999
(810) 555-9000
```

Анализ

И в данном примере применяются два регулярных выражения. Первое из них выглядит намного более сложным, чем оно есть на самом деле, и поэтому рассмотрим его подробнее. Шаблон `(\d{3})(-)(\d{3})(-)(\d{4})` обеспечивает совпадение с номером телефона, но разбит на пять подвыражений, чтобы разделить номер телефона на отдельные части. В частности, первое подвыражение `(\d{3})` обнаруживает совпадение с первыми тремя цифрами номера телефона, второе подвыражение `(-)` — со знаком дефиса и т.д. В конечном счете номер телефона разделяется на пять частей, каждая из которых представлена отдельным подвыражением: код зоны, дефис, первые три цифры номера телефона, еще один дефис и четыре последние цифры номера телефона. Эти пять частей могут быть использованы по отдельности и по мере надобности, поэтому второй шаблон `($1) $3-$5` просто переформатирует номер телефона, используя лишь три подвыражения и пренебрегая двумя другими, благодаря чему исходный номер телефона `313-555-1234` превращается в номер `(313) 555-1234`.

Совет

При манипулировании текстом для его переформатирования нередко оказывается полезно разбить исходный текст на целый ряд небольших подвыражений, чтобы получить над ним больший контроль.

Смена регистра букв

В некоторых реализациях регулярных выражений поддерживается ряд операций смены регистра букв с помощью метасимволов, перечисленных в табл. 8.1.

Таблица 8.1. Метасимволы для смены регистра букв

Метасимвол	Описание
\E	Завершить операцию смены регистра букв **\L** или **\U**
\l	Привести следующий символ к нижнему регистру букв
\L	Привести к нижнему регистру букв все символы вплоть до метасимвола **\E**
\u	Привести следующий символ к верхнему регистру букв
\U	Привести к верхнему регистру букв все символы вплоть до метасимвола **\E**

Метасимволы `\l` и `\u` указываются перед символом (или выражением), чтобы привести следующий символ (или выражение) к нижнему или верхнему регистру букв соответственно. А метасимволы `\L` и `\U` сменяют регистр букв всех символов вплоть до завершающего метасимвола `\E`.

Ниже демонстрируется простой пример приведения текста, заключенного между парой дескрипторов разметки `<h1>`, к верхнему регистру букв.

Текст

```
<body>
<h1>Welcome to my Homepage</h1>
```

```
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>
</body>
```

Регулярное выражение

```
(<[Hh]1>)(.*?)(<\/[Hh]1>)
```

Замена

```
$1\U$2\E$3
```

Результат

```
<body>
h1>WELCOME TO MY HOMEPAGE</h1>
Content is divided into two sections:<br/>
<h2>SQL</h2>
Information about SQL.
<h2>RegEx</h2>
Information about Regular Expressions.
<h2>This is not valid HTML</h3>
</body>
```

Анализ

Первый шаблон `(<[Hh]1>)(.*?)(<\/[Hh]1>)` в данном примере разбит на три подвыражения для сопоставления с заголовком и последовательного обнаружения открывающего дескриптора разметки, текста заголовка и закрывающего дескриптора разметки. А второй шаблон служит для составления заголовка из отдельных частей. В частности, переменная `$1` содержит начальный дескриптор разметки, шаблон `\U$2\E` приводит результат совпадения со вторым подвыражением в первом шаблоне (т.е. текст заголовка) к верхнему регистру букв, а переменная `$3` содержит конечный дескриптор разметки.

Резюме

Подвыражения служат для определения наборов символов или выражений. Они не только применяются для обнаружения повторяющихся совпадений, как демонстрировалось в предыдущем уроке, но и допускают обращение к ним в шаблонах. Такого рода обращение называется обратной ссылкой, но, к сожалению, ее синтаксис в разных реализациях регулярных выражений заметно различается. Обратные ссылки удобны для применения в операциях совпадения и замены текста.

Урок 9

Просмотр вперед и назад

Все рассматривавшиеся до сих пор регулярные выражения служили для обнаружения совпадений с текстом, но иногда требуются выражения для отметки позиции в совпавшем тексте, в отличие от самого текста. Для данной цели служит *позиционный просмотр* — возможность выполнять просмотр вперед и назад, поясняемая в этом уроке.

Общее представление о позиционном просмотре

Начнем, как всегда, с конкретного примера. Допустим, что требуется извлечь заглавие из веб-страницы. Заглавия HTML-страниц размещаются между дескрипторами `<title>` и `</title>` в разделе `<head>` кода HTML-разметки. В следующем примере демонстрируется решение поставленной задачи.

Текст

```
<head>
<title>Ben Forta's Homepage</title>
</head>
```

Регулярное выражение

```
<[tT][iI][tT][lL][eE]>.*<\/[tT][iI][tT][lL][eE]>
```

Результат

```
<head>
<title>Ben Forta's Homepage</title>
</head>
```

Анализ

Шаблон `<[tT][iI][tT][lL][eE]>.*<\/[tT][iI][tT][lL][eE]>` обеспечивает совпадение с открывающим дескриптором разметки `<title>` (прописными, строчным и теми и другими буквами), закрывающим дескриптором разметки `</title>` и любым расположенным между ними текстом.

Таким образом, цель можно считать достигнутой или все же нельзя? В данном примере требовалось извлечь текст заглавия веб-страницы, но полученный результат содержит также открывающий и закрывающий дескрипторы разметки `<title>` и `</title>`. А можно ли возвратить только текст заглавия?

В качестве одного из решений этой задачи можно было бы воспользоваться подвыражениями, как пояснялось в уроке 7. Это дало бы возможность извлечь сначала совпавший текст тремя частями: открывающий дескриптор разметки, текст заглавия и закрывающий дескриптор разметки, после чего не составило бы особого труда извлечь именно ту часть, которая требуется.

Но вряд ли стоит прилагать столько усилий, чтобы извлекать то, что не нужно, лишь для того, чтобы затем удалять его вручную. На самом деле в данном случае требуется составить шаблон таким образом, чтобы он содержал совпадения, которые не возвращаются, но используются для поиска нужного места совпадения, а не как составные части основного совпадения. Иными словами, требуется *позиционный просмотр*.

Примечание

В этом уроке рассматривается просмотр как *вперед*, так и *назад*. Первая разновидность просмотра поддерживается во всех основных реализациях регулярных выражений, а вторая — не так широко. В частности, просмотр назад поддерживается в Java, .NET, PHP, Python и Perl, хотя и с некоторыми ограничениями, а в JavaScript он не поддерживается.

Просмотр вперед

Для просмотра вперед указывается шаблон, по которому обнаруживается, но не возвращается совпадение. По существу, просмотр вперед определяет подвыражение, и поэтому он форматируется соответствующим образом. Синтаксически шаблон для просмотра вперед состоит из подвыражения, предваряемого знаками **?=**, а после знака **=** следует сопоставляемый текст.

Обратимся к следующему примеру. Допустим, что имеется текст, содержащий список URL. Из каждого URL требуется извлечь часть, определяющую сетевой протокол. Для этого желательно, конечно, знать, каким образом обрабатываются URL.

Текст

```
http://www.forta.com/
https://mail.forta.com/
ftp://ftp.forta.com/
```

Регулярное выражение

```
.+(?=:)
```

Результат

```
http://www.forta.com/
https://mail.forta.com/
ftp://ftp.forta.com/
```

Анализ

В перечисленных выше URL сетевой протокол отделяется от имени хоста (т.е. сетевого узла) знаком :. Шаблон .+ обеспечивает совпадение с любым текстом (сначала с ним совпадает название сетевого протокола http), а подвыражение (?=:) — совпадение со знаком :. Обратите, однако, внимание на то, что фактического совпадения со знаком : не произошло. Метасимволы ?= предписывают механизму обработки регулярных выражений обнаружить совпадение со знаком :, но произвести просмотр вперед, а не употребить этот знак.

Чтобы стало понятнее назначение метасимволов ?=, рассмотрим снова тот же самый пример, но на этот раз без этих метасимволов просмотра вперед.

Текст

```
http://www.forta.com/
https://mail.forta.com/
ftp://ftp.forta.com/
```

Регулярное выражение

```
.+(:)
```

Результат

```
http://www.forta.com/
http://mail.forta.com/
ftp://ftp.forta.com/
```

Анализ

Подвыражение (:) исправно обнаруживает совпадение со знаком :, но этот совпавший текст употребляется и возвращается как часть основного совпадения.

Оба представленных выше примера отличаются тем, что для совпадения со знаком : в первом из них применяется шаблон (?=:), а во втором — шаблон (:). Оба эти шаблона обеспечивают совпадение с той же самой частью исходного

текста, т.е. со знаком :, следующим после названия сетевого протокола. А отличаются они фактическим включением знака : в совпавший текст. При просмотре вперед синтаксический анализатор регулярных выражений выполняет поиск вперед, чтобы обработать совпадение со знаком :, но не обрабатывает его как часть основного поиска. Так, по шаблону .+(:) обнаруживается текст вплоть до знака :, включая и сам знак, а по шаблону (?=:) — тот же самый текст, но исключая знак :.

Примечание

При просмотре вперед и назад на самом деле возвращаются результаты совпадений, но эти результаты всегда оказываются длиной в *нуль* символов. Именно поэтому операции просмотра вперед и назад (т.е. позиционного просмотра) иногда еще называются операциями *нулевой ширины*.

Совет

Любое подвыражение может быть превращено в выражение для просмотра вперед, для чего достаточно предварить искомый текст метасимволами **?=**. В шаблоне поиска можно использовать несколько выражений для просмотра вперед, причем в любом его месте, а не только вначале, как демонстрировалось в приведенных выше примерах.

Просмотр назад

Как было показано ранее, метасимволы ?= обозначают просмотр вперед, т.е. поиск того, что следует после совпавшего и возвращаемого текста, но то, что обнаруживается в конечном итоге, на самом деле не употребляется. Таким образом, метасимволы ?= обозначают операцию *просмотра вперед*. Помимо просмотра вперед, во многих реализациях регулярных выражений поддерживается просмотр назад, т.е. поиск того, что находится прежде совпавшего и возвращаемого текста. Операция просмотра назад обозначается метасимволами ?<=.

Совет

Чтобы различить метасимволы **?=** и **?<=** и обозначаемые ими операции, достаточно запомнить следующее простое правило: те метасимволы, которые содержат стрелку, указывающую назад (**<=**), обозначают операцию просмотра назад.

Метасимволы ?<= употребляются таким же образом, как и метасимволы ?=, в подвыражениях и после текста, сопоставляемого с шаблоном. В следующем примере из сведений о видах продукции, найденных в базе данных, требуется извлечь только цены.

Текст

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4[1]
```

Регулярное выражение

```
\$[0-9.]+
```

Результат

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4
```

Анализ

В данном примере шаблон \$ обеспечивает совпадение со знаком денежной единицы $, а шаблон [0-9.]+ — с ценой.

Итак, поставленная цель в данном примере достигнута. Но что если наличие знаков **$** в совпавшем тексте не требуется? Можно ли просто исключить шаблон \$ из

[1] Всего найдено товаров: 4

приведенного выше регулярного выражения? Ниже показано, что из этого выйдет.

Текст

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4
```

Регулярное выражение

```
[0-9.]+
```

Результат

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4
```

Анализ

Очевидно, что такой способ не годится. Шаблон `\$` все же требуется, чтобы определить совпадающий текст. Но в то же время знак **$** не должен возвращаться в результате поиска на совпадение.

Как же выйти из этого затруднительного положения? Для этого придется организовать совпадение при просмотре назад, как демонстрируется в следующем примере.

Текст

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4
```

Регулярное выражение

```
(?<=\$)[0-9.]+
```

Результат

```
ABC01: $23.45
HGG42: $5.31
CFMX1: $899.00
XTC99: $69.96
Total items found: 4
```

Анализ

Теперь поставленная цель достигнута. В частности, подвыражение `(?<=\$)` обеспечивает совпадение со знаком **$**, но он не употребляется, и поэтому возвращаются только цены (без предваряющего их знака денежной единицы **$**).

Сравним первое и последнее регулярные выражения, использованные в приведенных выше примерах. В частности, с помощью регулярного выражения `\$[0-9.]+` было обнаружено совпадение со знаком денежной единицы **$** и цены в долларах США. То же самое было обнаружено и с помощью регулярного выражения `(?<=\$)[0-9.]+`. А различаются эти регулярные выражения не тем, что было обнаружено в результате поиска, а тем, что было включено в полученные результаты. В первом случае знак денежной единицы **$** был обнаружен и включен в результаты совпадения, а во втором случае он был исключен из них.

> **Предупреждение**
>
> Шаблоны для просмотра вперед могут иметь переменную длину. Они могут, например, содержать метасимволы . и +, чтобы стать в высшей степени динамичными. А шаблоны для просмотра назад должны, как правило, иметь фиксированную длину. Такое ограничение накладывается практически во всех реализациях регулярных выражений.

Сочетание просмотра вперед и назад

Операции просмотра вперед и назад можно сочетать, как демонстрируется в следующем примере, в котором решается задача, поставленная в начале этого урока.

Текст

```
<head>
<title>Ben Forta's Homepage</title>
</head>
```

Регулярное выражение

```
(?<=<[tT][iI][tT][lL][eE]>).*(?=</[tT][iI][tT][lL][eE]>)
```

Результат

```
<head>
<title>Ben Forta's Homepage</title>
</head>
```

Анализ

Регулярное выражение в данном примере сработало исправно. В частности, подвыражение `(?<=<[tT][iI][tT][lL][eE]>)` обозначает операцию поиска назад, обнаруживающую совпадение, но употребляющую открывающий дескриптор разметки `<title>`. Аналогично подвыражение `(?=</[tT][iI][tT][lL][eE]>)` обозначает операцию поиска вперед, обнаруживающую совпадение, но употребляющую закрывающий дескриптор разметки `</title>`. В итоге возвращается только текст заглавия веб-страницы, т.е. именно то, что и было фактически употреблено.

> **Совет**
>
> В приведенном выше примере, возможно, имело смысл экранировать первый сопоставляемый знак **<**, чтобы исключить неоднозначность. Следовательно, вместо выражения **(?<=<** лучше употребить выражение **(?<=\<**.

Отрицание позиционного просмотра

Как было показано до сих пор, просмотр вперед и назад обычно применяется для организации совпадения с текстом, чтобы, по существу, указать конкретное местоположение в возвращаемом тексте (т.е. текст до или после требующегося совпадения). Это так называемый *положительный* позиционный просмотр (*вперед* и *назад*). Термином *положительный* здесь обозначается то обстоятельство, что в операциях позиционного просмотра осуществляется поиск совпадения.

Менее употребительной формой позиционного просмотра является *отрицательный* позиционный просмотр (*вперед* и *назад*). В частности, при отрицательном просмотре вперед осуществляется поиск в прямом направлении текста, *не* совпадающего с указанным шаблоном, а при отрицательном просмотре назад — поиск в обратном направлении текста, *не* совпадающего с указанным шаблоном.

В упомянутых выше операциях можно было бы ожидать употребления знака `^` для отрицания позиционного просмотра, но нет, их синтаксис несколько иной. Операции позиционного просмотра отрицаются с помощью знака `!`, заменяющего знак `=`. Все операции позиционного просмотра перечислены в табл. 9.1.

Таблица 9.1. Операции позиционного просмотра

Класс операции	Описание
`(?=)`	Положительный просмотр вперед
`(?!)`	Отрицательный просмотр вперед
`(?<=)`	Положительный просмотр назад
`(?<!)`	Отрицательный просмотр назад

Совет

В любых реализациях регулярных выражений, в которых поддерживается просмотр вперед, как правило, поддерживаются как положительный, так и отрицательный просмотры вперед. Аналогично в тех реализациях регулярных выражений, в которых поддерживается просмотр назад, также поддерживаются как положительный, так и отрицательный просмотры назад.

Чтобы продемонстрировать отличия положительного просмотра назад от отрицательного, обратимся к конкретному примеру. В приведенном ниже блоке текста содержатся числа, обозначающие как цены, так и количества фруктов. Получим сначала цены на фрукты, как показано ниже.

Текст

```
I paid $30 for 100 apples,
50 oranges, and 60 pears.
I saved $5 on this order.[2]
```

Регулярное выражение

```
\b(?<!\$)\d+\b
```

Результат

```
I paid $30 for 100 apples,
50 oranges, and 60 pears.
I saved $5 on this order.
```

Анализ

И в данном примере шаблон `\d+` обеспечивает совпадение с числами, но на этот раз с ним совпали количества фруктов, но не цены на них. Подвыражение `(?<!\$)` обеспечивает отрицательный просмотр назад, при котором будет обнаружено совпадение только в том случае, если то, что предшествует числам, не является знаком **$**. Чтобы

[2] Я заплатил 30 долларов США за 100 яблок, 50 апельсин и 30 груш, сэкономив на этом заказе 5 долларов США.

изменить просмотр назад с положительного на отрицательный, достаточно заменить знак ! знаком =.

В ходе анализа приведенного выше примера отрицательного просмотра назад может возникнуть следующий вопрос: почему в шаблоне определяются границы слов с помощью метасимвола \b? Чтобы стала понятнее причина, по которой они нужны в данном шаблоне, рассмотрим тот же самый пример, но на этот раз без установленных границ слов.

Текст

```
I paid $30 for 100 apples,
50 oranges, and 60 pears.
I saved $5 on this order.
```

Регулярное выражение

```
(?<!\$)\d+
```

Результат

```
I paid $30 for 100 apples,
50 oranges, and 60 pears.
I saved $5 on this order.
```

Анализ

Без установленных границ совпала также цифра 0 в цене $30. Дело в том, что перед этой ценой стоит знак денежной единицы **$**. Этот недостаток можно устранить, заключив весь шаблон в границы слов.

Резюме

Позиционный просмотр (вперед и назад) обеспечивает более полный контроль над тем, что возвращается при сопоставлении с шаблоном. В операциях позиционного просмотра допускается применять подвыражения, чтобы указать конкретную позицию в сопоставляемом по шаблону

тексте. Но обнаруженное на этой позиции совпадение не употребляется, т.е. не включается в возвращаемый совпавший текст. Положительный просмотр вперед обозначается с помощью подвыражения `(?=)`, а отрицательный просмотр вперед — с помощью подвыражения `(?!)`. В некоторых реализациях регулярных выражений поддерживается также положительный и отрицательный просмотры назад, обозначаемые с помощью подвыражений `(?<=)` и `(?<!)` соответственно.

Урок 10

Встраивание условий

Возможность встраивать условную обработку в регулярные выражения весьма эффективна, хотя и применяется нечасто. Поэтому данному предмету посвящен этот урок.

Причины для встраивания условий

Для представления номеров телефонов в Северной Америке употребляются форматы `(123)456-7890` и `123-456-7890`. Номера телефонов `1234567890`, `(123)-456-7890` и `(123-456-7890` содержат нужное количество цифр, но неудачно отформатированы. Как же написать регулярное выражение для совпадения только с *приемлемыми* и ни с какими другими форматами номеров телефонов?

Это непростая задача. Рассмотрим следующее ее очевидное решение.

Текст

```
123-456-7890
(123)456-7890
(123)-456-7890
(123-456-7890
1234567890
123 456 7890
```

Регулярное выражение

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

Результат

```
123-456-7890
(123)456-7890
(123)-456-7890
(123-456-7890
1234567890
123 456 7890
```

Анализ

В данном примере шаблон `\(?` обеспечивает совпадение с открывающей скобкой (обратите внимание: что знак `(` должен быть непременно экранирован), шаблон `\d{3}` — с первыми тремя цифрами номера телефона, шаблон `\)?` — с дополнительной закрывающей скобкой, шаблон `-?` — с дополнительным знаком дефиса, а шаблон `\d{3}-\d{4}` — с остальными семью цифрами номера телефона, разделяемыми дефисом. Рассматриваемое здесь регулярное выражение совершенно верно не обнаружило совпадение с двумя последними строками в исходном тексте из данного примера, но в то же время оно обнаружило совпадение с третьей и четвертой строками, отформатированными неверно, причем в третьей строке содержатся оба знака, **)** и **-**, а в четвертой строке — непарная скобка.

Если заменить шаблон `\)?-?` шаблоном `[\)-]?`, из совпавшего текста будет исключена третья строка, поскольку в данном случае допускается наличие в номере телефона только знаков **)** или **-**, но не и тех, и других вместе. Но эта замена не разрешает затруднение, связанное с четвертой строкой. На самом деле регулярное выражение должно обнаруживать совпадение с закрывающей скобкой только в том случае, если имеется открывающая скобка, а иначе — совпадение с дефисом. Но такого рода регулярное выражение нельзя реализовать без условной обработки.

Предупреждение

Условная обработка поддерживается не во всех реализациях регулярных выражений.

Применение условий

Условия в регулярных выражениях определяются с помощью метасимвола **?**. На самом деле в представленных ранее примерах уже демонстрировалась пара совершенно особых условий.

- Условие ? обеспечивает совпадение с предыдущим символом или выражением, *если* этот символ или выражение существует.
- Условие ?= и ?<= обеспечивает совпадение с текстом вперед или назад, *если* этот текст существует.

В синтаксисе встраиваемых условий употребляется также метасимвол **?**, что не удивительно, если учесть, что условия встраиваются и обрабатываются на основании обратной ссылки и позиционного просмотра.

Условия в обратных ссылках

Условие в обратной ссылке позволяет использовать выражение только в том случае, если поиск предыдущего подвыражения завершается удачно. Чтобы стало понятнее, обратимся к конкретному примеру, в котором требуется найти все дескрипторы разметки <img> в исходном тексте. А если любые дескрипторы <img> размечают ссылки, заключаемые в дескрипторы <a> и , то придется организовать совпадение и со всеми дескрипторами разметки ссылок.

Синтаксис такого рода условия следующий:

```
(?(обратная ссылка)true)
```

Условие начинается с метасимвола **?**, затем в круглых скобках указывается обратная ссылка, а далее следует выражение, которое вычисляется лишь в том случае, если обратная ссылка присутствует.

Текст

```
<!-- Навигационная панель -->
<div>
<a href="/home"><img src="/images/home.gif"></a>
<img src="/images/spacer.gif">
<a href="/search"><img src="/images/search.gif"></a>
<img src="/images/spacer.gif">
<a href="/help"><img src="/images/help.gif"></a>
</div>
```

Регулярное выражение

```
(<[Aa]\s+[^>]+>\s*)?<[Ii][Mm][Gg]\
s+[^>]+>(?(1)\s*<\/[Aa]>)
```

Результат

```
<!-- Навигационная панель -->
<div>
<a href="/home"><img src="/images/home.gif"></a>
<img src="/images/spacer.gif">
<a href="/search"><img src="/images/search.gif"></a>
<img src="/images/spacer.gif">
<a href="/help"><img src="/images/help.gif"></a>
</div>
```

Анализ

Регулярное выражение в данном примере требует дополнительного пояснения. В частности, шаблон `(<[Aa]\s+[^>]+>\s*)?` обеспечивает совпадение с открывающим дескриптором разметки <A> или <a> (с любыми атрибутами, которые могут в нем присутствовать). Затем шаблон `<[Ii][Mm][Gg]\s+[^>]+>` обеспечивает совпадение с дескриптором разметки <img> (независимо от регистра букв) с любыми его атрибутами. А шаблон `(?(1)\s*</[Aa]>)` начинается с условия `?(1)`, которое означает выполнение только того, что следует дальше, если существует обратная ссылка `1` (на открывающий дескриптор разметки <A>), т.е. только в том случае, если сопоставление с первым дескриптором разметки <A> завершится удачно. Если же обратная

ссылка (1) существует, то шаблон \s*</[Aa]> обеспечит совпадение с любыми конечными пробельными символами, следующими после закрывающего дескриптора </A>.

Примечание

По условию **?(1)** проверяется, имеется ли обратная ссылка **1**. Номер обратной ссылки (в данном случае — **1**) совсем необязательно экранировать в условиях. Следовательно, условие **?(1)** составлено правильно, а условие **?(\1)** — неправильно, хотя оно, как правило, оказывается вполне пригодным.

В рассмотренном выше примере указанное выражение выполняется лишь в том случае, если заданное условие выполняется. В условиях могут также присутствовать выражения *else*, которые выполняются лишь в том случае, если обратная ссылка отсутствует (т.е. условие не выполняется). Синтаксис для такой формы условия следующий:

```
(?(обратная ссылка)true|false)
```

В этом синтаксисе предусмотрены условие, а также два выражения, которые должны быть вычислены в зависимости от того, выполняется условие или не выполняется. Благодаря именно такому синтаксису удается найти решение упоминавшейся ранее задачи выявления номеров телефонов, как показано ниже.

Текст

```
123-456-7890
(123)456-7890
(123)-456-7890
(123-456-7890
1234567890
123 456 7890
```

Регулярное выражение

```
(\()?\d{3}(?(1)\)|-)\d{3}-\d{4}
```

Результат

```
123-456-7890
(123)456-7890
(123)-456-7890
(123-456-7890
1234567890
123 456 7890
```

Анализ

Регулярное выражение в данном примере, по-видимому, действует исправно, но почему? Как и прежде, по шаблону `(\()?` проверяется начало пары скобок, но на этот раз получаемые результаты заключаются в круглые скобки, образуя в итоге подвыражение. Шаблон `\d{3}` обеспечивает совпадение с кодом зоны, состоящим из трех цифр, а шаблон `(?(1)\)|-)` —с закрывающей скобкой или дефисом в зависимости от того, выполняется ли заданное условие. Если существует обратная ссылка `(1)`, обозначающая обнаружение открывающей скобки, то должно произойти совпадение с закрывающей скобкой (`\)`), а иначе — с дефисом (`-`). Таким образом, круглые скобки должны быть всегда парными, а сопоставление с дефисом, отделяющим код зоны от номера телефона, осуществляется лишь в том случае, если круглые скобки не употребляются в номере телефона. Почему же тогда обнаружено совпадение с четвертой строкой? А потому, что у открывающей скобки в этой строке отсутствует парная закрывающая скобка, и поэтому она полностью проигнорирована как не относящийся к поиску текст.

> **Совет**
>
> Шаблоны могут стать довольно сложными в результате встраивания выражений, что сильно затрудняет поиск в них ошибок. Поэтому рекомендуется составлять и проверять их сначала небольшими частями, а затем объединять эти части.

Условия при позиционном просмотре

Условие при позиционном просмотре позволяет выполнять выражения в зависимости от удачного завершения операции просмотра вперед или назад. Синтаксис условий при позиционном просмотре очень похож на синтаксис в обратных ссылках, за исключением того что обратная ссылка, обозначаемая числом в круглых скобках, заменяется в нем целым выражением для позиционного просмотра.

> **Примечание**
>
> Подробнее об обработке результатов позиционного просмотра читайте в уроке 9.

В качестве примера рассмотрим почтовые индексы США, которые могут состоять из пяти цифр в формате `12345` или же в формате `12345-6789` с четырьмя дополнительными цифрами. Дефис употребляется лишь в том случае, если в почтовом индексе присутствуют четыре дополнительные цифры. Ниже приведено одно из решений задачи выявления почтовых индексов.

Текст

```
11111
22222
33333-
44444-4444
```

Регулярное выражение

```
\d{5}(-\d{4})?
```

Результат

```
11111
22222
33333-
44444-4444
```

Анализ

В данном примере шаблон `\d{5}` обеспечивает совпадение с первыми пятью цифрами почтового индекса, а шаблон `(-\d{4})?` — с четырьмя дополнительными цифрами, если они существуют.

Но что если требуется исключить совпадение с любыми неверно отформатированными почтовыми индексами? Так, третья строка в приведенном выше примере завершается дефисом, которого в конце почтового индекса, по-видимому, быть не должно. Регулярное выражение в приведенном выше примере позволяет обнаруживать совпадение с цифрами без дефиса, но как исключить совпадение с целым почтовым индексом, если он отформатирован неверно?

Приведенный ниже пример несколько надуман, тем не менее он наглядно демонстрирует применение условий при позиционном просмотре.

Текст

```
11111
22222
33333-
44444-4444
```

Регулярное выражение

```
\d{5}(?(?=-)-\d{4})
```

Результат

```
11111
22222
33333-
44444-4444
```

Анализ

И в данном примере шаблон `\d{5}` обеспечивает совпадение с первыми пятью цифрами почтового индекса. Условие применяется при просмотре вперед `?=-`, чтобы

обнаружить совпадение, но не употребить дефис. И если данное условие выполняется, т.е. дефис существует, то шаблон `-\d{4}` обеспечит совпадение с дефисом и четырьмя последующими цифрами. Таким образом, совпадение со строкой `33333-` не произошло, поскольку в ней отсутствуют четыре дополнительные цифры, несмотря на наличие дефиса и выполнение данного условия.

Просмотр вперед и назад (как положительный, так и отрицательный), а также дополнительное выражение *else* можно задавать в качестве условия, используя тот же самый синтаксис, что и прежде:

```
| выражение
```

> **Совет**
>
> Условия нечасто применяются при позиционном просмотре, поскольку аналогичных результатов можно, как правило, добиться более простыми средствами.

Резюме

Условия можно встраивать в шаблоны регулярных выражений, чтобы выполнять выражения в зависимости от того, выполняется ли заданное условие. В качестве условия может быть задана обратная ссылка, и тогда в условии проверяется ее наличие или же операция позиционного просмотра.

Урок 11

Решение типичных задач с помощью регулярных выражений

В этом уроке рассматривается ряд регулярных выражений и подробно поясняется их назначение и применение. Цель этого урока — подытожить все, что было вами усвоено в предыдущих уроках в данной книге, а также продемонстрировать на реальных примерах применение регулярных выражений для решения типичных практических задач.

Примечание

Приведенные далее примеры применения регулярных выражений не следует считать окончательными решениями рассматриваемых в этом уроке типичных задач. В настоящий момент вам должно быть ясно, что окончательное решение редко удается найти с помощью регулярных выражений. Чаще всего существует несколько решений, допускающих в разной степени непредсказуемые результаты. Кроме того, всегда существует компромисс между производительностью шаблона и его пригодностью на все случаи жизни. Принимая все это во внимание, можете свободно пользоваться представленными далее регулярными выражениями и даже приспосабливать их к конкретным обстоятельствам по мере надобности.

Номера телефонов в Северной Америке

Североамериканский план нумерации (NANP — North American Numbering Plan) определяет порядок форматирования номеров телефонов. Согласно этому плану номера телефонов (в Соединенных Штатах, Канаде, большей части стран Карибского бассейна и ряде других регионов) состоят из кода зоны длиной три цифры, формально называемого *зоной по плану нумерации* (или сокращенно — "NPA") и самого номера длиной семь цифр, который форматируется в виде *префикса* длиной три цифры, дефиса и *номера абонентской линии* длиной четыре цифры. В номере телефона допускается употреблять любые цифры, за двумя следующими исключениями: первой цифрой кода зоны и префикса не должен быть 0 или 1. Зачастую код зоны заключается в круглые скобки и отделяется дефисом от самого номера телефона. Организовать совпадение с одной из форм `(555) 555-5555`, `(555)555-5555` и `555-555-5555` номера телефона нетрудно, но труднее — с любой из этих форм, как показано ниже.

Текст

```
J. Doe: 248-555-1234
B. Smith: (313) 555-1234
A. Lee: (810)555-1234
```

Регулярное выражение

```
\(?[2-9]\d\d\)?[ -]?[2-9]\d\d-\d{4}
```

Результат

```
J. Doe: 248-555-1234
B. Smith: (313) 555-1234
A. Lee: (810)555-1234
```

Анализ

В данном примере регулярное выражение начинается с необычного вида шаблона `\(?`. Круглые скобки для

обозначения кода зоны необязательны, и потому шаблон `\(?` обеспечивает совпадение с нулевыми или единичным (метасимвол **?**) количеством открывающих скобок (знаки **\(**). Шаблон `[2-9]\d\d` обеспечивает совпадение с кодом зоны, состоящим из трех цифр, причем первая его цифра должна находиться в пределах от 2 до 9. Шаблон `\)?` обеспечивает совпадение с необязательной закрывающей скобкой, а шаблон `[ -]?` — с единственным пробелом или дефисом, если он присутствует в номере телефона. И наконец, шаблон `[2-9]\d\d-\d{4}` обеспечивает совпадение непосредственно с номером телефона, состоящим из префикса длиной три цифры, причем первая его цифра должна находиться в пределах от 2 до 9, дефиса и еще четырех цифр.

Рассмотренное выше регулярное выражение нетрудно приспособить под другие форматы представления номеров телефонов. Ниже приведен пример для формата `555.555.5555`.

Текст

```
J. Doe: 248-555-1234
B. Smith: (313) 555-1234
A. Lee: (810)555-1234
M. Jones: 734.555.9999
```

Регулярное выражение

```
[\(.]?[2-9]\d\d[\).]?[ -]?[2-9]\d\d[-.]\d{4}
```

Результат

```
J. Doe: 248-555-1234
B. Smith: (313) 555-1234
A. Lee: (810)555-1234
M. Jones: 734.555.9999
```

Анализ

В регулярном выражении из данного примера сначала проверяется наличие открывающей скобки или точки

в качестве необязательного знака с помощью шаблона [\(.]?. Подобным образом, наличие закрывающей скобки или точки также в качестве необязательного знака проверяется с помощью шаблона [\).]?, а наличие дефиса или точки — с помощью шаблона [-.]. Аналогично в регулярное выражение можно добавить другие форматы номеров телефонов.

Почтовые индексы США

В Соединенных Штатах почтовые индексы были внедрены в 1963 году. Они называются иначе ZIP-кодами и происходят от названия *Zone Improvement Plan* (Зональный план улучшения). Все почтовые индексы, а их в США насчитывается более 40 тысяч, состоят из цифр. Первая цифра присваивается от Восточного до Западного побережья, причем 0 обозначает Восточное побережье, а 9 — Западное побережье. В 1983 году американская почтовая служба начала пользоваться расширенными почтовыми индексами в формате *ZIP+4*. Четыре дополнительные цифры в почтовом индексе обеспечивают большую степень детализации (зачастую до уровня городского квартала, а иногда и отдельного здания), что, в свою очередь, повышает надежность почтовых услуг. И хотя пользоваться расширенным почтовым индексом необязательно, проверку достоверности почтовых индексов необходимо приспособить как под пятизначный формат, так и под десятизначный формат, в котором употребляется дефис для отделения четырех дополнительных цифр почтового индекса от первых пяти. В приведенном ниже примере показано, каким образом решается эта задача.

Текст

```
999 1st Avenue, Bigtown, NY, 11222
123 High Street, Any City, MI 48034-1234
```

Регулярное выражение

```
\d{5}(-\d{4})?
```

Результат

```
999 1st Avenue, Bigtown, NY, 11222
123 High Street, Any City, MI 48034-1234
```

Анализ

В данном примере шаблон `\d{5}` обеспечивает совпадение с любыми пятью цифрами почтового индекса, а шаблон `-\d{4}` — с дефисом и еще четырьмя цифрами. А поскольку четыре последние цифры, по существу, необязательны, то шаблон `-\d{4}` заключен в круглые скобки, т.е. превращен в подвыражение, после которого следует метасимвол **?**, допускающий наличие одного экземпляра данного подвыражения.

Канадские почтовые индексы

Почтовые индексы в Канаде состоят из шести символов с чередованием букв и цифр. Первый ряд из трех букв и цифр обозначает *первую зону сортировки* (FSA), а второй ряд из букв и цифр — *местное отделение доставки почты* (LDU). Первая буква в части FSA обозначает провинцию, территорию или регион (на этой позиции допускается 18 букв, причем буква `A` обозначает провинции Ньюфаундленд и Лабрадор, буква `B` — провинцию Новая Шотландия, буквы `K`, `L`, `N` и `P` — провинцию Онтарио, кроме города Торонто, для обозначения которого используется буква `M`, и т.д.). Таким образом, при проверке канадских почтовых индексов было бы идеально убедиться в достоверности первой буквы. Как правило, канадские почтовые индексы форматируются с помощью пробелов, отделяющих первую часть FSA от второй части LDU. В приведенном ниже примере показано, каким образом решается задача обработки канадских почтовых индексов.

Текст

```
123 4th Street, Toronto, Ontario, M1A 1A1
567 8th Avenue, Montreal, Quebec, H9Z 9Z9
```

Регулярное выражение

```
[ABCEGHJKLMNPRSTVXY]\d[A-Z] \d[A-Z]\d
```

Результат

```
123 4th Street, Toronto, Ontario, M1A 1A1
567 8th Avenue, Montreal, Quebec, H9Z 9Z9
```

Анализ

В данном примере шаблон `[ABCEGHJKLMNPRSTVXY]` обеспечивает совпадение с любой из 18 допустимых букв, шаблон `\d[A-Z]` — с цифрой и любой буквой, а следовательно, с первой частью FSA почтового индекса. И наконец, шаблон `\d[A-Z]\d` обеспечивает совпадение со второй частью LDU почтового индекса, где после цифры следуют любая буква и еще одна цифра.

> **Примечание**
>
> В рассмотренном выше регулярном выражении совсем необязательно учитывать регистр букв.

Почтовые коды Великобритании

Британские почтовые индексы состоят из пяти, шести или семи букв и цифр, определяемых Королевской почтой. Они делятся на две части: *внешний* и *внутренний* почтовый индексы. Внешний почтовый индекс состоит из одной или двух букв, одной или двух цифр или же одной или двух букв, после которых следует цифра и буква. А внутренний почтовый индекс всегда состоит из одной цифры и любых двух букв, кроме букв `C`, `I`, `K`, `M`, `O` и `V`, которые вообще не употребляются в британских почтовых индексах. Внешний

почтовый индекс отделяется от внутреннего пробелом. В приведенном ниже примере показано, каким образом решается задача обработки британских почтовых индексов.

Текст

```
171 Kyverdale Road, London N16 6PS
33 Main Street, Portsmouth, P01 3AX
18 High Street, London NW11 8AB
```

Регулярное выражение

```
[A-Z]{1,2}\d[A-Z\d]? \d[ABD-HJLNP-UW-Z]{2}
```

Результат

```
171 Kyverdale Road, London N16 6PS
33 Main Street, Portsmouth, P01 3AX
18 High Street, London NW11 8AB
```

Анализ

По шаблону `[A-Z] {1,2}\d. [A-Z\d]?` в данном примере обнаруживается совпадение с одной или двумя буквами и последующей цифрой во внутреннем почтовом индексе, а по шаблону `[A-Z\d]?` — с дополнительным буквенно-цифровым символом, если он присутствует в данном индексе. Следовательно, шаблон `[A-Z] {1,2}\d[A-Z\d]?` обеспечивает совпадение с любой допустимой комбинацией буквенно-цифровых символов во внутреннем почтовом индексе. А для сопоставления с внешним почтовым индексом служит шаблон `\d[ABD-HJLNP-UW-Z]{2}`. Он обеспечивает совпадение с одной цифрой и двумя последующими допустимыми буквами (от `A`, `B`, `D` до `H`, `J`, `L`, `N`, `P` и до `U`, а также от `W` до `Z`).

> **Примечание**
>
> В рассмотренном выше регулярном выражении совсем необязательно учитывать регистр букв.

Номера карточек социального страхования в США

В Соединенных Штатах номера карточек социального страхования, обычно обозначаемые сокращенно как "SSN", состоят из трех рядов цифр, разделяемых дефисами. Первый ряд состоит из трех цифр, второй ряд — из двух цифр, а третий ряд — из четырех цифр. Начиная с 1972 года первый ряд из трех цифр присваивается на основании адреса, указываемого в заявлении лица, претендующего на социальное страхование. В приведенном ниже примере показано, каким образом решается задача обработки номеров социального страхования в США.

Текст

```
John Smith: 123-45-6789
```

Регулярное выражение

```
\d{3}-\d{2}-\d{4}
```

Результат

```
John Smith: 123-45-6789
```

Анализ

В данном примере шаблон `\d{3}-\d{2}-\d{4}` обеспечивает совпадение с любыми тремя цифрами, последующим дефисом, двумя цифрами, еще одним дефисом и любыми четырьмя оставшимися цифрами.

> **Примечание**
>
> Практически любая комбинация цифр может оказаться достоверным номером карточки социального страхования, поэтому для проверки достоверности подобных номеров можно при необходимости применить пару правил. В частности, номера карточек социального страхования не должны содержать поля, состоящие из одних только нулей, а первый ряд цифр

(до настоящего времени) должен обозначать номер не больше 728, поскольку номера больше этого пока еще не присвоены, хотя такое вполне возможно в будущем. Но написать такое регулярное выражение будет совсем непросто, и поэтому обычно применяется более простой шаблон **`\d{3}-\d{2}-\d{4}`**.

IP-адреса

Обычные IP-адреса состоят из четырех байтов, каждый из которых находится в пределах от 0 до 255. Как правило, IP-адреса форматируются в виде четырех рядов цифр, состоящих из трех цифр и разделяемых точками. В приведенном ниже примере показано, каким образом решается задача обработки IP-адресов.

Текст

```
localhost is 127.0.0.1.
```

Регулярное выражение

```
(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2}))\.){3}
(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2})))
```

Результат

```
localhost is 127.0.0.1.
```

Анализ

В регулярном выражении из данного примера применяется ряд вложенных подвыражений. Первая его часть состоит из четырех вложенных подвыражений `(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2}))\.)`. В частности, подвыражение `(\d{1,2})` обеспечивает совпадение с любым числом, состоящим из одной или двух цифр, т.е. с числом от 0 до 99; подвыражение `(1\d{2})` — с любым числом, состоящим из трех цифр, начиная с 1, и еще двух любых цифр, т.е. с числом от 100 до 199; подвыражение `(2[0-4]\d)` — с числом от 200 до 249; а подвыражение

(25[0-5]) — с числом от 250 до 255. Каждое из этих подвыражений вложено в еще одно подвыражение и отделяется от других подвыражений знаком операции |, обозначающей сопоставление с одним из четырех подвыражений, а не со всеми сразу. После этого следуют знаки \., обозначающие совпадение со знаком точки, а затем вся эта последовательность заключается в еще одно подвыражение и повторяется три раза с помощью интервала {3}. Далее описанный выше ряд подвыражений повторяется, но на этот раз без завершающих знаков \., чтобы обеспечить совпадение с последним числом в IP-адресе. Ограничивая каждое из четырех чисел в IP-адресе пределами от 0 до 255, рассматриваемое здесь регулярное выражение действительно обнаруживает совпадение с достоверными IP-адресами, отвергая недостоверные адреса.

Примечание

В рассмотренном выше регулярном выражении совсем необязательно учитывать регистр букв.

URL

Обнаружить совпадение с URL — непростая задача, которая может еще больше усложниться в зависимости от того, насколько гибким должно быть совпадение. При сопоставлении с URL должно быть как минимум обнаружено совпадение с сетевым протоколом (вероятнее всего, http и https), именем хоста (сетевого узла), необязательным номером порта и путем к искомому веб-ресурсу. В приведенном ниже примере показано, каким образом решается задача обработки URL.

Текст

```
http://www.forta.com/blog
https://www.forta.com:80/blog/index.cfm
http://www.forta.com
http://ben:password@www.forta.com/
http://localhost/index.php?ab=1&c=2
http://localhost:8500/
```

Регулярное выражение

```
https?:\/\/[-\w.]+(:\d+)?(\/([\w\/_.]*)?)?
```

Результат

```
http://www.forta.com/blog
https://www.forta.com:80/blog/index.cfm
http://www.forta.com
http://ben:password@www.forta.com/
http://localhost/index.php?ab=1&c=2
http://localhost:8500/
```

Анализ

В данном примере шаблон `https?://` обеспечивает совпадение с префиксом сетевого протокола `http://` или `https://`, причем метасимвол **?** обозначает совпадение с дополнительной буквой **s**; шаблон `[-\w.]+` — с именем хоста; шаблон `(:\d+)?` — с дополнительным портом, присутствующим во второй и шестой строках исходного текста из данного примера; шаблон `(/([\w/_.]*)?)?` — с путем к искомому веб-ресурсу; внешнее подвыражение — со знаком **/**, если он присутствует в заданном пути, а внутреннее подвыражение — с самим путем. Как видите, данное регулярное выражение не в состоянии обрабатывать строки запросов, а также правильно читать встроенные в URL пары "имя пользователя–пароль". Но для проверки большинства URL такое регулярное выражение вполне пригодно, обнаруживая совпадение с именами хостов, номерами портов и путями к веб-ресурсам.

Примечание

В рассмотренном выше регулярном выражении совсем необязательно учитывать регистр букв.

Совет

Чтобы обработать URL по сетевому протоколу **ftp**, достаточно заменить шаблон **https?** шаблоном **(http|https|ftp)**. То же самое можно сделать и для обработки других типов URL.

Полные URL

Более полное, но и медленнее действующее регулярное выражение способно обнаружить совпадение со строками запросов в URL, т.е. с переменной информацией, передаваемой URL и отделяемой от самого URL знаком **?**. Такое регулярное выражение позволяет также обнаружить совпадение с регистрационной информацией, если она указана в URL. В приведенном ниже примере показано, каким образом решается задача обработки полных URL.

Текст

```
http://www.forta.com/blog
https://www.forta.com:80/blog/index.cfm
http://www.forta.com
http://ben:password@www.forta.com/
http://localhost/index.php?ab=1&c=2
http://localhost:8500/
```

Регулярное выражение

```
https?:\/\/(\w*:\w*@)?[-\w.]+(:\d+)?(\/
([\w\/_.]*(\?\S+)?)?)?
```

Результат

```
http://www.forta.com/blog
https://www.forta.com:80/blog/index.cfm
http://www.forta.com
http://ben:password@www.forta.com/
http://localhost/index.php?ab=1&c=2
http://localhost:8500/
```

Анализ

Регулярное выражение в данном примере основано на предыдущем примере. Теперь после шаблона `https?://` следует шаблон `(\w*:\w*@)?`. В данном регулярном выражении проверяется наличие в URL встроенного имени пользователя и пароля, разделяемых двоеточием, после чего следует знак `@`, как показано в четвертой строке исходного

текста из данного примера. Кроме того, шаблон `(\?\S+)?` обеспечивает совпадение со строкой запроса, указываемой после пути к веб-ресурсу, начиная со знака **?**, а также следующим далее дополнительным текстом благодаря применению метасимвола **?**.

Примечание

В рассмотренном выше регулярном выражении совсем необязательно учитывать регистр букв.

Совет

А почему бы не воспользоваться регулярным выражением из данного примера вместо регулярного выражения из предыдущего примера? С точки зрения производительности это несколько более сложное регулярное выражение, и поэтому оно будет выполняться медленнее. Если не требуются дополнительные функциональные возможности, то пользоваться им не рекомендуется.

Адреса электронной почты

Регулярные выражения нередко применяются для проверки достоверности адресов электронной почты, хотя проверить простой адрес электронной почты не составляет большого труда. В приведенном ниже примере показано, каким образом решается задача проверки достоверности адресов электронной почты.

Текст

```
My name is Ben Forta, and my
email address is ben@forta.com.
```

Регулярное выражение

```
(\w+\.)*\w+@(\w+\.)+[A-Za-z]+
```

Результат

```
My name is Ben Forta, and my
email address is ben@forta.com.
```

Анализ

В данном примере шаблон `(\w+\.)*\w+` обеспечивает совпадение с той частью адреса электронной почты, в которой указывается имя адресата, т.е. со всем, что предшествует знаку `@`. Часть `(\w+\.)*` этого шаблона обнаруживает совпадение с нулевым или большим количеством символов и последующей точкой; а другая его часть `\w+` — с требующимся текстом (в данном случае — как с именем `ben`, так и с именем `ben.forta`). Шаблон `@` обеспечивает совпадение со знаком `@`, следующий далее шаблон `(\w+\.)+` — хотя бы с одним экземпляром текста и последующей точкой, а шаблон `[A-Za-z]+` — с доменом верхнего уровня (`com`, `edu`, `us` или `uk`).

Правила, определяющие форматирование достоверных адресов электронной почты, довольно сложны. Рассмотренное выше регулярное выражение не в состоянии проверить достоверность всех возможных адресов электронной почты. Например, с его помощью будет обнаружено совпадение с недостоверным адресом наподобие `ben..forta@forta.com`, но не с IP-адресами, которые обозначают имена хостов и вполне допустимы. Тем не менее данного регулярного выражения будет достаточно для проверки достоверности большинства адресов электронной почты, а следовательно, оно может пригодиться и вам.

> **Примечание**
>
> В регулярных выражениях, применяемых для проверки достоверности адресов электронной почты, совсем необязательно учитывать регистр букв.

Комментарии к HTML-разметке

Комментарии на HTML-страницах размещаются между дескрипторами разметки `<!--` и `-->`, где допускается указывать от двух и больше дефисов. Обнаруживать комментарии удобно при просмотре (и отладке) веб-страниц. В приведенном ниже примере показано, каким образом решается задача обработки комментариев к HTML-разметке.

Текст

```
<!-- Начало страницы -->
<html>
<!-- Начало заголовка -->
<head>
<tile>My Title</title> <!-- Заглавие страницы -->
</head>
<!-- Тело страницы -->
<body>
```

Регулярное выражение

```
<!-{2,}.*?-{2,}>
```

Результат

```
<!-- Начало страницы -->
<html>
<!-- Начало заголовка -->
<head>
<tile>My Title</title> <!-- Заглавие страницы -->
</head>
<!-- Тело страницы -->
<body>
```

Анализ

В данном примере шаблон `<!-{2,}` обеспечивает совпадение с началом комментария, т.е. со знаками `<!` и двумя или больше дефисами; шаблон `.*?` — (не жадно) с телом комментария; а шаблон `-{2,}>` — с концом комментария.

> **Примечание**
>
> Рассмотренное выше регулярное выражение позволяет обнаружить совпадение с двумя или больше дефисами, а следовательно, выявить комментарии к CFML-разметке, обозначаемые тремя дефисами. Тем не менее в этом регулярном выражении не предпринимается попытка обнаружить парность дефисов в начале и конце комментариев, что было бы полезно для выявления комментариев с непарным числом дефисов.

Комментарии к сценариям JavaScript

В языке JavaScript и других языках сценариев, включая ActionScript и прочие производные стандарта ECMAScript, комментарии предваряются знаками //. Как и в предыдущем примере, выявить сразу все комментарии на вебстранице очень удобно при ее просмотре и отладке. В приведенном ниже примере показано, каким образом решается задача обработки комментариев к сценариям JavaScript.

Текст

```
<script language="JavaScript">
// скрыть поля, применяемые только для замены
function hideReplaceFields() {
  document.getElementById('RegExReplace').
disabled=true;
  document.getElementById('replaceheader').
disabled=true;
}
// показать поля, применяемые только для замены
function showReplaceFields() {
  document.getElementById('RegExReplace').
disabled=false;
  document.getElementById('replaceheader').
disabled=false;
}
```

Регулярное выражение

```
\/\/.*
```

Результат

```
<script language="JavaScript">
// скрыть поля, применяемые только для замены
function hideReplaceFields() {
  document.getElementById('RegExReplace').
disabled=true;
  document.getElementById('replaceheader').
disabled=true;
}
// показать поля, применяемые только для замены
```

```
function showReplaceFields() {
  document.getElementById('RegExReplace').
disabled=false;
  document.getElementById('replaceheader').
disabled=false;
}
```

Анализ

Регулярное выражение в данном примере очень простое. В частности, шаблон `\/\/.*` обеспечивает совпадение со знаками // и последующим телом комментария.

Номера кредитных карточек

На самом деле номера кредитных карточек нельзя по-настоящему проверить на достоверность с помощью регулярных выражений. Для окончательной их проверки всегда требуется некоторое взаимодействие с организацией, обрабатывающей кредитные карточки. Тем не менее номера кредитных карточек могут быть проверены с помощью регулярных выражений на наличие таких опечаток при их вводе, как, например, чрезмерное или недостаточное количество цифр, прежде чем передавать их куда-то еще на последующую обработку.

> **Примечание**
>
> Во всех применяемых далее шаблонах предполагается, что любые вставляемые пробелы или дефисы удалены. Удалять любые не цифровые символы из номеров кредитных карточек, прежде чем обрабатывать их с помощью регулярных выражений, обычно считается надлежащей нормой практики.

Нумерация всех кредитных карточек осуществляется по следующей основной схеме: после начальной последовательности цифр следует указанное количество цифр. В приведенном ниже примере проверка номеров кредитных карточек начинается с карточки MasterCard.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
5[1-5]\d{14}
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

Номера всех кредитных карточек MasterCard состоят из 16 цифр. Первой всегда является цифра 5, а второй следует цифра от 1 до 5, поэтому шаблон `5[1-5]` обеспечивает совпадение с первыми двумя цифрами кредитной карточки, а шаблон `\d{14}` — с остальными четырнадцатью цифрами.

Проверить номера кредитных карточек Visa немного сложнее, как показано ниже.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
4\d{12}(\d{3})?
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

Номера всех кредитных карточек Visa начинаются с цифры 4 и состоят из 13 или 16 цифр, а не 14 или 15 цифр, и поэтому для их проверки требуется иной интервал. В данном примере символ 4 совпадает с начальной цифрой 4 номера кредитной карточки, шаблон \d{12} — со следующими двенадцатью цифрами, а шаблон (\d{3})? — с тремя дополнительными цифрами, если они присутствуют в номере кредитной карточки.

Для проверки номеров кредитных карточек American Express требуется намного более простое регулярное выражение, как показано ниже.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
3[47]\d{13}
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

Номера кредитных карточек American Express состоят из 15 цифр и начинаются с цифр 34 или 37. В данном примере шаблон `3[47]` совпадает с первыми двумя цифрами номера кредитной карточки, а шаблон `\d{13}` — с остальными тринадцатью цифрами.

Для проверки номеров кредитных карточек Discover Express также требуется простое регулярное выражение, как показано ниже.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
6011\d{14}
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

Номера всех кредитных карточек Discover состоят из 16 цифр и начинаются с цифр 6011. Поэтому цель их проверки достигается с помощью шаблона `6011\d{14}`.

Проверить номер кредитной карточки Diners немного сложнее, как показано ниже.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
(30[0-5]|36\d|38\d)\d{11}
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

Номера кредитных карточек Diners Club состоят из 14 цифр и начинаются с цифр 300–305, 36 или 38. Если номер кредитной карточки начинается с цифр 300–305, то требуются еще одиннадцать дополнительных цифр, а если он начинается с цифр 36 или 38, — то двенадцать дополнительных цифр. Чтобы упростить проверку, регулярное выражение в данном примере начинается с сопоставления с первыми тремя цифрами, какими бы они ни были. Поэтому подвыражение `(30[0-5]|36\d|38\d)` состоит из трех шаблонов, любой из которых должен совпасть в тремя начальными цифрами номера кредитной карточки. В частности, шаблон `30[0-5]` обеспечивает совпадение с цифрами 300–305, шаблон `36\d` — с любыми тремя цифрами, начиная с 36, а шаблон `38\d` — с любыми тремя цифрами, начиная с 38. А по шаблону `\d{11}` обеспечивается совпадение с оставшимися одиннадцатью цифрами.

Теперь остается лишь организовать проверку на достоверность номера любого из пяти описанных выше типов кредитных карточек, как показано ниже.

Текст

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Регулярное выражение

```
(5[1-5]\d{14})|(4\d{12}(\d{3})?)|(3[47]\d{13})|
(6011\d{14})|((30[0-5]|36\d|38\d)\d{11})
```

Результат

```
MasterCard: 5212345678901234
Visa 1: 4123456789012
Visa 2: 4123456789012345
Amex: 371234567890123
Discover: 601112345678901234
Diners Club: 38812345678901
```

Анализ

В регулярном выражении из данного примера применяется чередование альтернативных шаблонов, объединяемых с помощью логической операции ИЛИ, обозначаемой знаком |. В итоге организуется простая проверка достоверности номеров всех основных типов кредитных карточек.

> **Примечание**
>
> Шаблоны, применяемые в приведенном выше примере, позволяют проверить правильность ввода первых цифр и длины номеров кредитных карточек. Но не каждый номер, состоящий из тринадцати цифр и начинающийся с цифры 4, является достоверным номером кредитной карточки Visa. Чтобы проверить подлинность цифр в номерах всех упомянутых выше типов

кредитных карточек, можно воспользоваться известным алгоритмом *Mod 10*. И хотя этот алгоритм является важной составляющей реализации процедуры обработки кредитных карточек, его нельзя применить в регулярных выражениях, поскольку он подразумевает математические расчеты.

Резюме

В этом уроке были продемонстрированы практические примеры применения многих понятий и принципов составления регулярных выражений, представленных в предыдущих уроках. Можете свободно пользоваться этими примерами, приспосабливая их к своим потребностям. А вместе с тем добро пожаловать в увлекательный и плодотворный мир регулярных выражений!

Приложение

Регулярные выражения в распространенных приложениях и языках

Несмотря на то что в разных реализациях синтаксис регулярных выражений в основном согласован, этого нельзя определенно сказать об их применении. В тех языках программирования и приложениях, в которых поддерживаются регулярные выражения, применяются собственные методы вызова этих выражений, которые имеют незначительные, а порой и значительные, отличия и особенности. В этом приложении описывается применение регулярных выражений в наиболее распространенных приложениях и языках программирования с соответствующими примечаниями там, где это уместно.

Примечание

Сведения, представленные в этом приложении, служат для справки и оказания помощи тем, кто приступает к изучению регулярных выражений. Конкретные примеры применения и примечания к каждой реализации регулярных выражений выходят за рамки данной книги. Поэтому за дополнительными сведениями обращайтесь к документации, сопровождающей соответствующее приложение или язык программирования.

Утилита grep

Утилита grep служит в ОС Unix для поиска текста в файлах или в стандартном потоке ввода. В ней поддерживаются основные, расширенные или применяемые в языке Perl регулярные выражения в зависимости от параметра, указанного в командной строке, как поясняется ниже.

- **-E** — обозначает расширенные регулярные выражения.
- **-G** — обозначает основные регулярные выражения.
- **-P** — обозначает регулярные выражения, применяемые в языке Perl.

> **Совет**
>
> Конкретные свойства и функциональные возможности утилиты **grep** зависят от параметра, указанного в командной строке. Большинство пользователей предпочитают регулярные выражения, применяемые в языке Perl и описываемые далее, поскольку они в наибольшей степени стандартизированы.

Применяя регулярные выражения в утилите grep, необходимо иметь в виду следующее.

- По умолчанию утилита grep отображает полностью любые строки, содержащие совпадения. Чтобы отобразить только совпадение, следует указать параметр **-o**.
- Чтобы отобразить только не совпавшие строки, т.е. отрицать совпадение, следует указать параметр **-v**.
- Чтобы отобразить подсчитанное количество совпадений вместо подробных сведений о них, следует указать параметр **-c**.
- Чтобы организовать совпадение без учета регистра букв, следует указать параметр **-i**.
- Утилита grep служит для выполнения операций поиска, но не замены, поэтому функциональные возможности замены в ней не поддерживаются.

Язык Java

В языке Java для обнаружения совпадений с помощью регулярных выражений служит класс `java.util.rgex.Matcher`, содержащий следующие методы.

- `find()` — обнаруживает вхождение заданного шаблона в символьной строке.
- `lookingAt()` — пытается обнаружить совпадение в начале символьной строки по заданному шаблону.
- `matches()` — пытается обнаружить совпадение со всей символьной строкой по заданному шаблону.
- `replaceAll()` — выполняет операцию замены всех обнаруженных совпадений.
- `replaceFirst()` — выполняет операцию замены первого обнаруженного совпадения.

Дополнительные методы из упомянутого выше класса обеспечивают более полный контроль над конкретными операциями. Кроме того, в классе `java.util.regex.Pattern` предоставляются следующие методы-оболочки.

- `compile()` — компилирует выражение в шаблон.
- `flags()` — возвращает признаки сопоставления с шаблоном.
- `matches()` — функционально равнозначен описанному ранее методу `matches()`.
- `pattern()` — возвращает регулярное выражение, из которого был составлен шаблон.
- `split()` — разбивает символьную строку на подстроки.

Поддержка регулярных выражений в языке Java основывается на их реализации в языке Perl. Необходимо, однако, иметь в виду следующее.

- Чтобы воспользоваться регулярными выражениями, необходимо импортировать соответствующий пакет с помощью оператора `import java.util.regex.*`. (Этот оператор импортирует весь пакет. Если же требуются только отдельные подпакеты, вместо знака * в этом операторе следует указать их имена.)
- Встроенные условия не поддерживаются.
- Смена регистра букв с помощью метасимволов `\E`, `\l`, `\L`, `\u` и `\U` не поддерживается.
- Возврат на одну позицию назад с помощью метасимвола `\b` не поддерживается.
- Метасимвол `\z` не поддерживается.

Язык JavaScript

В языке JavaScript обработка регулярных выражений реализуется в объектах типа `String` и `RegEx` через следующие методы.

- `exec()` — метод из объекта типа `RegEx`, предназначенный для обнаружения совпадений.
- `match()` — метод из объекта типа `String`, предназначенный для обнаружения совпадений с символьными строками.
- `replace()` — метод из объекта типа `String`, предназначенный для выполнения операций замены.
- `search()` — метод из объекта типа `String`, предназначенный для проверки символьных строк на совпадение.
- `split()` — метод из объекта типа `String`, предназначенный для разбиения символьной строки на несколько подстрок.
- `test()` — метод из объекта типа `RegEx`, предназначенный для проверки символьных строк на совпадение.

Поддержка регулярных выражений в языке JavaScript основывается на модели из языка Perl. Необходимо, однако, иметь в виду следующие особенности.

- Для организации глобального поиска с учетом регистра букв в JavaScript применяются следующие признаки:
  - **g** — активизирует глобальный поиск;
  - **i** — обнаруживает совпадения без учета регистра букв.
- Оба эти признака могут употребляться вместе: **gi**.
- Начиная с версии 4 в браузерах поддерживаются следующие дополнительные модификаторы:
  - **m** — служит для поддержки многострочного текста;
  - **s** — служит для поддержки однострочного текста;
  - **x** — игнорирует пробельные символы в шаблоне.
- Если используются обратные ссылки, то по шаблону `$`` возвращается все, что предшествует совпавшей символьной строке, по шаблону `$'` — все, что следует после совпавшей символьной строки, по шаблону `$+` — последнее совпавшее подвыражение, а по шаблону `$&` — все, что совпало.
- В языке JavaScript имеется глобальный объект типа `RegExp`, доступный для получения сведений о ходе дальнейшего выполнения после того, как будет выполнено регулярное выражение.
- Классы символов по стандарту POSIX не поддерживаются.
- Метасимволы `\A` и `\Z` не поддерживаются.

Платформа Microsoft .NET

На платформе Microsoft .NET предоставляются эффективные и гибкие средства для обработки регулярных

выражений, входящие в состав библиотеки базовых классов. Поэтому регулярные выражения доступны для применения в любом языке программирования или инструментальном средстве, поддерживаемом на платформе .NET, включая ASP.NET, C# и Visual Studio .NET.

Поддержка регулярных выражений на платформе .NET предоставляется средствами класса `Regex`, а также дополнительных поддерживающих классов. В состав класса `Regex` входят следующие методы.

- `IsMatch()` — проверяет, обнаружено ли совпадение в указанной символьной строке.
- `Match()` — обнаруживает совпадение, возвращаемое в виде объекта типа `Match`.
- `Matches()` — обнаруживает все совпадения, возвращаемые в виде объекта типа `MatchCollection`.
- `Replace()` — выполняет операцию замены в указанной символьной строке.
- `Split()` — разбивает символьную строку на массив символьных строк.

Кроме того, регулярное выражение можно выполнить через следующие методы-оболочки, не получая экземпляр класса `Regex` и оперируя им.

- `Regex.IsMatch()` — функционально равнозначен описанному выше методу `IsMatch()`.
- `Regex.Match()` — функционально равнозначен описанному выше методу `Match()`.
- `Regex.Matches()` — функционально равнозначен описанному выше методу `Matches()`.
- `Regex.Replace()` — функционально равнозначен описанному выше методу `Replace()`.
- `Regex.Split()` — функционально равнозначен описанному выше методу `Split()`.

Ниже перечислен ряд важных особенностей, касающихся поддержки регулярных выражений на платформе .NET.

- Чтобы воспользоваться регулярными выражениями, необходимо импортировать их объекты с помощью оператора `Imports System.Text.RegularExpressions`.
- Для быстрой обработки регулярных выражений, встраиваемых в строки кода, идеально подходят методы-оболочки.
- Параметры регулярных выражений указываются с помощью свойства `Regex.Options` — перечисления `RegexOption`, в котором могут быть установлены такие члены, как `IgnoreCase`, `Multiline`, `Singleline` и пр.
- На платформе .NET поддерживается именованный захват, т.е. возможность именовать подвыражения, чтобы обращаться к ним по имени, а не по номеру. Для именования подвыражений служит синтаксис `?<`*имя*`>`, для обратных ссылок на них — синтаксис `\k<`*имя*`>`, а для ссылки на шаблон замены — синтаксис `${`*имя*`}`.
- Если применяются обратные ссылки, то по шаблону `$`` возвращается все, что предшествует совпавшей символьной строке, по шаблону `$'` — все, что следует после совпавшей символьной строки, по шаблону `$+` — последнее совпавшее подвыражение, по шаблону `$_` — вся исходная символьная строка, а по шаблону `$&` — вся совпавшая символьная строка.
- Смена регистра букв с помощью метасимволов `\E`, `\l`, `\L`, `\u` и `\U` не поддерживается.
- Классы символов по стандарту POSIX не поддерживаются.

Язык Microsoft SQL Server T-SQL

На самом сервере базы данных Microsoft SQL Server регулярные выражения не поддерживаются. Тем не менее в операторах языка SQL Server T-SQL можно применять среду Microsoft CLR (Common Language Runtime — Общеязыковая исполняющая среда), через которую становятся доступными функциональные возможности регулярных выражений. Описание среды CLR выходит за рамки данной книги, хотя документацию на нее можно найти на официальном веб-сайте корпорации Microsoft.

Microsoft Visual Studio .NET

Поддержка регулярных выражений в Visual Studio .NET обеспечивается средствами .NET Framework. Подробнее о платформе .NET читайте в соответствующем разделе ранее в этом приложении.

Чтобы воспользоваться регулярными выражениями в Visual Studio .NET, необходимо выполнить следующие действия.

- Выбрать команду меню Edit⇨Find and Replace (Правка⇨Найти и заменить).
- Выбрать команду Replace⇨Find in Files (Заменить⇨Найти в файлах) или Replace in Files (Заменить в файлах).
- Установить флажок Use (Использовать) и выбрать регулярные выражения из раскрывающегося списка.

Применяя регулярные выражения, необходимо также иметь в виду следующие особенности.

- Вместо знаков `*?` употребляется знак `@`.
- Вместо знака `#` употребляются знаки `+?`.
- Вместо знаков `^n` употребляются знаки `{n}`.
- В операциях замены обратные ссылки могут быть соответственно дополнены для выравнивания по

левому краю с помощью выражения \(w,n), где w — ширина, а n — обратная ссылка. А для выравнивания по правому краю служит выражение \(-w,n).

- В Visual Studio .NET употребляются следующие специальные метасимволы и знаки:
  - **:a** — для обозначения интервала [a-zA-Z0-9];
  - **:c** — для обозначения интервала [a-zA-Z];
  - **:d** — для обозначения метасимвола \d;
  - **:h** — для обозначения интервала [a-fA-F0-9] (шестнадцатеричных чисел);
  - **:i** — для обозначения достоверных идентификаторов [a-zA-Z_$][a-zA-Z_0-9$]*;
  - **:q** — для обозначения символьных строк в кавычках;
  - **:w** — для обозначения шаблона [a-zA-Z]+;
  - **:z** — для обозначения шаблона \d+.
- Знаками \n обозначается разрыв строки независимо от используемой платформы. Они служат для вставки знаков новой строки в операциях замены.
- В Visual Studio .NET поддерживаются следующие специальные символы для сопоставления с отдельными буквами:
  - **:Lu** — обозначает совпадение с любой прописной буквой;
  - **:Ll** — обозначает совпадение с любой строчной буквой;
  - **:Lt** — обозначает совпадение с заглавной буквой;
  - **:Lm** — обозначает совпадение со знаками препинания.
- Для сопоставления с разными числами служат следующие специальные знаки:
  - **:Nd** — обозначает совпадение с шаблоном [0-9]+;
  - **:Nl** — обозначает совпадение с римскими цифрами.

- Для сопоставления с разными знаками препинания служат следующие специальные знаки:
  - `:Ps` — обозначает совпадение с открывающими знаками препинания;
  - `:Pe` — обозначает совпадение с закрывающими знаками препинания;
  - `:Pi` — обозначает совпадение со знаками двойных кавычек;
  - `:Pf` — обозначает совпадение со знаками одиночных кавычек;
  - `:Pd` — обозначает совпадение со знаком дефиса;
  - `:Pc` — обозначает совпадение со знаком подчеркивания;
  - `:Po` — обозначает совпадение с прочими знаками препинания.
- Для сопоставления с разными символами препинания служат следующие специальные знаки:
  - `:Sm` — обозначает совпадение с математическими символами;
  - `:Sc` — обозначает совпадение со знаками денежных единиц;
  - `:Sk` — обозначает совпадение с модификаторами ударений;
  - `:So` — обозначает совпадение с другими специальными символами.
- В Visual Studio .NET поддерживаются и другие специальные символы. Подробнее об этом читайте в документации к Visual Studio .NET.

База данных MySQL

MySQL является весьма распространенной реляционной базой с открытым кодом, в которой обеспечивается такая поддержка регулярных выражений для выполнения

операций поиска, какую не решаются предоставлять поставщики других баз данных. Эта поддержка доступна в предложениях WHERE из запросов к базе данных MySQL в следующей форме:

```
REGEXP "выражение"
```

> **Примечание**
>
> Полная форма запроса к базе данных MySQL, в котором применяется регулярное выражение, синтаксически выглядит следующим образом:
>
> ```
> : SELECT * FROM table WHERE REGEXP "шаблон"
> ```

Поддержка регулярных выражений в MySQL удобна и эффективна, хотя не может считаться полноценной реализацией регулярных выражений. Ниже перечислены особенности этой поддержки.

- Обеспечивается поддержка только операций поиска, но не замены.
- Операции поиска выполняются без учета регистра букв. Чтобы выполнить их с учетом регистра букв, следует вставить слово BINARY между ключевым словом REGEXP и самим регулярным выражением.
- Для сопоставления с началом слова служит шаблон `[[:<:]]`, а для сопоставления с концом слова — шаблон `[[:>:]]`.
- Позиционный просмотр не поддерживается.
- Встраиваемые выражения не поддерживаются.
- Поиск восьмеричных чисел не поддерживается.
- Метасимволы \a, \b, \e, \f и \v не поддерживаются.
- Обратные ссылки не поддерживаются.

Язык Oracle PL/SQL

Язык PL/SQL является разновидностью языка SQL в реляционных базах данных Oracle. В языке PL/SQL регулярные выражения поддерживаются в условии `REGEXP_LIKE` вместо условия `SQL LIKE`.

Ниже перечислены некоторые из наиболее примечательных особенностей, касающихся применения регулярных выражений в языке PL/SQL.

- Условие `REGEXP_LIKE` служит для сопоставления с типами данных `VARCHAR2`, `CHAR`, `NVARCHAR2`, `NCHAR`, `CLOB` и `NCLOB`.
- Для сопоставления с учетом регистра букв в условии `REGEXP_LIKE` указывается параметр **`c`**.
- Для сопоставления без учета регистра букв в условии `REGEXP_LIKE` указывается параметр **`i`**.
- Для сопоставления со знаком новой строки в условии `REGEXP_LIKE` указывается параметр **`n`**.
- Для игнорирования пробельных символов в условии `REGEXP_LIKE` указывается параметр **`x`**.
- Для обозначения логической операции ИЛИ служит знак конвейеризации `|`.

Язык Perl

Язык Perl является патриархом всех реализаций регулярных выражений, и в большинстве из них предпринимается попытка обеспечить совместимость с Perl.

Поддержка регулярных выражений составляет основную часть языка Perl и обозначается в виде выполняемой операции и шаблона одним из следующих средств.

- `m/`*`шаблон`*`/` — обозначает совпадение с указанным шаблоном.
- `s/`*`шаблон`*`/`*`шаблон`*`/` — обозначает операцию замены.

- `qr/шаблон/` — обозначает возврат объекта типа `Regex`, который может быть использован в дальнейшем.
- `split()` — разбивает символьную строку на подстроки.

Ниже приведены некоторые из полезных примечаний относительно применения регулярных выражений в языке Perl.

- Модификаторы могут быть указаны после шаблона. В частности, модификатор `/i` служит для поиска без учета регистра букв, а модификатор `/g` — для глобального поиска (с целью обнаружить все совпадения).
- Если используются обратные ссылки, то по шаблону `$`` возвращается все, что предшествует совпавшей символьной строке, по шаблону `$'` — все, что следует после совпавшей символьной строки, по шаблону `$+` — последнее совпавшее подвыражение, а по шаблону `$&` — вся совпавшая символьная строка.

Язык PHP

В языке PHP обеспечивается совместимая с языком Perl поддержка регулярных выражений через пакет PCRE (Perl Compatible Regular Expressions — Регулярные выражения, совместимые с Perl).

В пакете PCRE предоставляются следующие функции для поддержки регулярных выражений.

- `preg_grep()` — выполняет поиск и возвращает массив совпадений.
- `preg_match()` — выполняет поиск первого совпадения с помощью регулярных выражений.
- `preg_match_all()` — выполняет глобальный поиск с помощью регулярных выражений.

- `preg_quote()` — принимает шаблон и возвращает его экранированную версию.
- `preg_replace()` — выполняет операцию поиска и замены.
- `preg_replace_callback()` — выполняет операцию поиска и замены, но использует отдельную функцию для конкретной замены.
- `preg_split()` — разбивает символьную строку на подстроки.

Применяя регулярные выражения в языке PHP, необходимо, однако, иметь в виду следующее.

- Для организации совпадения без учета регистра букв служит модификатор **`i`**.
- С помощью модификатора **`m`** можно активизировать режим обработки многострочного текста.
- Замещающие строки допускается вычислять в виде кода PHP. Для активизации такого режима служит модификатор **`e`**.
- Во всех функциях `preg_replace()`, `preg_replace_callback()` и `preg_split()` поддерживается дополнительный параметр, обозначающий ограничение на максимальное количество выполняемых замен или разбиений.
- Обратные ссылки могут обозначаться с помощью знака **$** (например, `$1`) начиная с версии 4.0.4 языка Perl, а в более ранних версиях вместо знака **$** применяются знаки ****.
- Метасимволы `\E`, `\l`, `\L`, `\u` и `\U` не поддерживаются.

Язык Python

В языке Python поддержка регулярных выражений обеспечивается через модуль `re`. Регулярные выражения

поддерживаются с помощью следующих функций языка Python.

- `preg_grep()` — выполняет поиск и возвращает массив совпадений.
- `findall()` — обнаруживает все подстроки и возвращает их в виде списка.
- `finditer()` — обнаруживает все подстроки и возвращает их в виде итератора.
- `match()` — выполняет поиск в начале символьной строки с помощью регулярных выражений.
- `search()` — выполняет поиск всех совпадений в символьной строке.
- `split()` — преобразует символьную строку в список, разбивая ее на те части, которые совпадают с заданным шаблоном.
- `sub()` — заменяет обнаруженные совпадения указанной подстрокой.
- `subn()` — возвращает символьную строку, в которой совпавшие части заменяются указанными подстроками.

Применяя регулярные выражения в языке Python, необходимо, однако, иметь в виду следующее.

- Регулярные выражения компилируются перед применением в объекты с помощью функции `re.compile()`.
- Функция `re.compile()` принимает дополнительные признаки наподобие `re.IGNORECASE` для поиска без учета регистра букв.
- Для отладки регулярного выражения служит признак `re.VERBOSE`.
- Функции `match()` и `search()` возвращают значение `None`, если совпадения не обнаружены.

Предметный указатель

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ: ОСНОВЫ

Майкл Фицджеральд

www.williamspublishing.com

Если вы программист, не имеющий опыта работы с регулярными выражениями, то данная книга — как раз то, что нужно для первого знакомства с ними. Многочисленные примеры, приведенные в книге, помогут вам не только освоить основы регулярных выражений, но и научиться применять их для поиска, извлечения и преобразования фрагментов текста посредством их сопоставления с определенными символами, словами и шаблонами. Прочитав книгу, вы будете знать синтаксис наиболее распространенных диалектов регулярных выражений и понимать, в каких случаях они смогут обеспечить вам значительную экономию времени.

- Узнайте о том, чем отличаются регулярные выражения, применяемые в утилитах командной строки и в различных языках программирования.
- Освойте простые способы нахождения определенных образцов текста, включая цифры, буквы, символы Unicode и строковые литералы.
- Научитесь работать с группами, обратными ссылками, классами символов и квантификаторами.

ISBN 978-5-8459-1953-3

в продаже

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ ЗА 10 МИНУТ

Бен Форта

www.williamspublishing.com

В данной книге представлены все наиболее важные сведения о регулярных выражениях: основные понятия и концепции, наборы символов, метасимволы, повторители, поиск позиции, подвыражения, ссылки назад, контекстный поиск (просмотр вперед и назад), условная обработка, реализация регулярных выражений в популярных приложениях и языках (grep, JavaScript, Macromedia ColdFusion, Macromedia Dreamweaver, Microsoft ASP, Microsoft ASP. NET, Microsoft C#, Microsoft .NET, Microsoft Visual Studio .NET, MySQL, Perl, PHP и Java). Особое внимание в книге уделяется технологии решения задач с помощью регулярных выражений. Подробно рассматриваются все этапы подготовки и тестирования регулярных выражений. Все теоретические положения детально демонстрируются на содержательных примерах, которые часто встречаются на практике. Большое внимание уделяется прагматическому подходу к решению практических задач.

ISBN 978-5-8459-2133-8

в продаже

ШАБЛОНЫ C++
СПРАВОЧНИК РАЗРАБОТЧИКА
ВТОРОЕ ИЗДАНИЕ

Дэвид Вандевурд
Николаи М. Джосаттис
Дуглас Грегор

www.williamspublishing.com

Второе издание книги отражает изменения, происшедшие в первую очередь с самим языком программирования C++ за 15 лет со времени первого издания, который стал еще более мощным орудием в руках программиста, позволяющим создавать существенно более производительные программы. Принятые со времени первого издания стандарты C++11, C++14 и C++17 сделали шаблоны еще более выразительным средством программирования, позволяющем создавать еще более эффективные (и эффектные с точки зрения эстетики программирования) программы. По сути, почти вся стандартная библиотека C++ построена на применении шаблонов. Книга описывает работу с шаблонами C++ от азов до самых "темных закутков", будет одинаково полезна как программисту, желающему повысить свой начальный уровень умений и понимания шаблонов, так и профессиональному разработчику сложнейших шаблонных библиотек. Книга рекомендуется в первую очередь для профессиональных программистов в качестве справочного пособия, но она будет полезна программистам на C++ любого уровня.

ISBN 978-5-9500296-8-4

в продаже

JAVA ДЛЯ ЧАЙНИКОВ
7-Е ИЗДАНИЕ

Барри Берд

www.dialektika.com

Перед вами бестселлер для начинающих, посвященный Java 9 — новой версии самого мощного объектно-ориентированного языка программирования. Программа, написанная на Java, будет выполняться практически на любом компьютере, ноутбуке или портативном устройстве. Освоив Java, вы сможете создавать мультимедийные приложения, предназначенные для любой платформы. Теперь пришел ваш черед! Независимо от того, на каком языке вы программировали раньше (и даже если вы никогда прежде не программировали), вы быстро научитесь создавать современные кроссплатформенные приложения, используя возможности Java 9.

Основные темы книги:

- ключевые концепции Java;
- грамматика языка;
- повторное использование кода;
- циклы и условные конструкции;
- основы объектно-ориентированного программирования;
- обработка исключений;
- использование ссылочных типов данных.

ISBN 978-5-9500296-1-5 **в продаже**

Научитесь пользоваться самыми эффективными средствами из всех доступных!

Знатоки регулярных выражений уже давно включили их в свой арсенал средств, чтобы выполнять самые разные, изощренные виды обработки текста и манипулирования практически на любом языке программирования и на любой вычислительной платформе. Но это лишь одна, приятная, сторона дела. А обратная, неприятная, сторона состоит в том, что регулярные выражения слишком долго оставались исключительным средством только для самых технически грамотных пользователей. Но так было до сих пор.

Эта книга поможет вам научиться пользоваться теми регулярными выражениями, которые действительно нужно знать, начиная с поиска простых совпадений с заданным текстом и заканчивая более сложными задачами, включая применение обратных ссылок, условные вычисления и обработку с упреждением. Прорабатывая материал каждого урока в данной книге, вы методически, систематически и легко научитесь решать практические задачи, используя регулярные выражения. Регулярные выражения не так сложны, как кажется. Чтобы умело ими пользоваться, достаточно уяснить поставленную задачу и ее наилучшее решение с помощью регулярных выражений.

>> Ясные, практические примеры, реализованные на разных языках

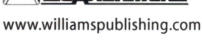
Прорабатывая уроки в данной книге, вы научитесь следующему.

- Читать и понимать регулярные выражения
- Пользоваться обычным текстом и метасимволами для построения эффективных шаблонов поиска
- Извлекать пользу из расширенных возможностей регулярных выражений, включая позиционный поиск и обратные ссылки
- Эффективно выполнять операции поиска и замены во всех профессиональных инструментальных средствах редактирования исходного текста
- Внедрять логически развитые формы обработки текста в веб-приложения
- Находить файлы, используя такие утилиты командной строки, как `grep` и `egrep`
- Применять регулярные выражения в таких языках программирования, как JavaScript, Java, PHP, Python Microsoft .NET и C#, а также в СУБД, включая MySQL и Oracle
- Обрабатывать номера телефонов, почтовые индексы, номера карточек социального страхования, IP-адреса, URL, адреса электронной почты и номера кредитных карточек

Бен Форта занимает пост старшего управляющего в отделении образовательных инициатив компании Adobe Systems, имея за плечами более чем 30-летний опыт разработки, поддержки, обучения и сбыта программных продуктов. Он является автором целого ряда книг из серии *Освой... за 10 минут* на самые разные темы, включая язык SQL, регулярные выражения и разработку приложений для мобильных устройств на платформах Adobe ColdFusion, Java и Windows.

Категория: программирование
Предмет рассмотрения: регулярные выражения
Уровень: начальный — средний

ISBN 978-5-6041394-2-4



www.williamspublishing.com