20 ГЛАВА

Сеть

Как известно читателям, Java — практически синоним программирования для Internet. К тому есть множество причин, и не последние из них — способность генерировать безопасный, межплатформенный и переносимый код. Однако одна из наиболее важных причин того, что Java является великолепным языком для сетевого программирования, кроется в классах, определенных в пакете `java.net`. Они обеспечивают легкие в использовании средства, с помощью которых программисты всех уровней квалификации могут обращаться к сетевым ресурсам.

Эта глава посвящена пакету `java.net`. Важно подчеркнуть, что сети — очень обширная и сложная тема. В настоящей книге невозможно полностью охватить все средства, содержащиеся в `java.net`. Поэтому в данной главе мы сосредоточим внимание лишь на некоторых основополагающих классах и интерфейсах.

Основы работы с сетью

Прежде чем начать, полезно будет получить представление о ключевых концепциях и терминах, связанных с сетями. В основе сетевой поддержки Java лежит концепция *сокета* (socket). Сокет идентифицирует конечную точку сети. Парадигма сокета появилась в версии 4.2BSD Berkley UNIX в самом начале 80-х гг. По этой причине также используется термин *сокет Беркли*. Сокеты — основа современных сетей, поскольку сокет позволяет отдельному компьютеру обслуживать одновременно как множество разных клиентов, так и множество различных типов информации. Это достигается за счет использования *порта* (port) — нумерованного сокета на определенной машине. Говорят, что серверный процесс "слушает" порт до тех пор, пока клиент не соединится с ним. Сервер в состоянии принять множество клиентов, подключенных к одному и тому же номеру порта, хотя каждый сеанс является уникальным. Чтобы обработать множество клиентских соединений, серверный процесс должен быть многопоточным либо обладать какими-то другими средствами обработки одновременного ввода-вывода.

Сокетные коммуникации происходят по определенному протоколу. *Internet-протокол* (Internet Protocol — IP) — это низкоуровневый маршрутизирующий протокол, который разбивает данные на небольшие пакеты и посылает их через сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу. *Протокол управления*

передачей (Transmission Control Protocol — TCP) является протоколом более высокого уровня, обеспечивающий надежную сборку этих пакетов, сортировку и повторную передачу, необходимую для надежной доставки данных. Третий протокол — *протокол пользовательских дейтаграмм* (User Datagram Protocol — UDP), стоящий непосредственно за TCP, может быть использован непосредственно для поддержки быстрой, не требующей постоянного соединения и ненадежной транспортировки пакетов.

Как только соединение установлено, применяется высокоуровневый протокол, зависящий от используемого порта. TCP/IP резервирует первые 1024 порта для специфических протоколов. Многие из них покажутся вам знакомыми, если вы хоть какое-то время потратили на путешествия в океане Internet. Порт номер 21 — для FTP, 23 — для Telnet, 25 — для электронной почты, 43 — для whois; 79 — для finger, 80 — для HTTP, 119 — для netnews; список можно продолжать. На каждый из протоколов возлагается определение того, как клиент должен взаимодействовать с портом.

Например, HTTP — это протокол, используемый серверами и Web-браузерами для передачи гипертекста и графических изображений. Это довольно простой протокол для базового постраничного просмотра информации, предоставляемой Web-серверами. Посмотрим, как оно работает. Когда клиент запрашивает файл с сервера HTTP, это действие известно как *попадание* (hit) и состоит в простой отправке имени файла в определенном формате на предопределенный порт с последующим чтением содержимого этого файла. Сервер также сообщает код состояния, чтобы известить клиент о том, был ли запрос обработан или нет, и по какой причине.

Ключевым компонентом Internet является *адрес*. Каждый компьютер в Internet обладает собственным адресом. Адрес Internet представляет собой число, уникально идентифицирующее каждый компьютер в Internet. Изначально все Internet-адреса состояли из 32-битных значений, организованных в четыре 8-битных значения. Адрес такого типа определен IPv4 (Internet-протокол версии 4). Однако в последнее время на сцену выступает новая схема адресации, называемая IPv6, которая предназначена для того, чтобы поддержать гораздо большее адресное пространство, чем IPv4.

Для обеспечения обратной совместимости с IPv4 младшие 32 бита адреса IPv6 могут содержать в себе корректный адрес IPv4. Таким образом, адресация IPv4 совместима снизу вверх с IPv6. К счастью, имея дело с Java, вам обычно не придется беспокоиться о том, используется адрес IPv4 или IPv6, поскольку Java позаботится обо всех деталях.

Точно так же, как IP-адрес описывает сетевую иерархию, имя адреса Internet, называемое *доменным именем*, описывает местонахождение машины в пространстве имен. Например, `www.osborne.com` относится к домену `com` (зарезервированному для коммерческих сайтов США), имеет имя `osborne` (по названию компании), а `www` идентифицирует сервер, обрабатывающий Web-запросы. Доменное имя Internet отображается на IP-адрес посредством *службы доменных имен* (Domain Name Service — DNS). Это позволяет пользователям работать с доменными именами, в то время как Internet оперирует IP-адресами.

Сетевые классы и интерфейсы

Java поддерживает TCP/IP как за счет расширения уже имеющихся интерфейсов потокового ввода-вывода, представленных в главе 19, так и за счет добавления средств, необходимых для построения объектов ввода-вывода в сети. Java поддерживает семейства протоколов как TCP , так и UDP. TCP применяется для надежного потокового ввода-вывода по сети. UDP поддерживает более простую, а потому быструю модель передачи дейтаграмм от точки к точке. Классы, содержащиеся в пакете `java.net`, перечислены ниже.

Authenticator	Inet6Address	ServerSocket
CacheRequest	InetAddress	Socket
CacheResponse	InetSocketAddress	SocketAddress
ContentHandler	InterfaceAddress (добавлен в Java SE 6)	SocketImpl
CookieHandler	JarURLConnection	SocketPermission
CookieManager (добавлен в Java SE 6)	MulticastSocket	URI
DatagramPacket	NetPermission	URL
DatagramSocket	NetworkInterface	URLClassLoader
DatagramSocketImpl	PasswordAuthentication	URLConnection
HttpCookie (добавлен в Java SE 6)	Proxy	URLDecoder
HttpURLConnection	ProxySelector	URLEncoder
IDN (добавлен в Java SE 6)	ResponseCache	URLStreamHandler
Inet4Address	SecureCacheResponse	

Интерфейсы пакета `java.net` перечислены далее:

ContentHandlerFactory	DatagramSocketImplFactory	SocketOptions
CookiePolicy (добавлен в Java SE 6)	FileNameMap	
CookieStore (добавлен в Java SE 6)	SocketImplFactory	

В следующем разделе мы рассмотрим основные сетевые классы и продемонстрируем несколько примеров их применения. Как только вы поймете устройство сетевых классов, то сможете строить собственные на их основе.

InetAddress

Класс `InetAddress` используется для инкапсуляции как числового IP-адреса, так и доменного имени для этого адреса. Вы взаимодействуете с классом, используя имя IP-хоста, что намного удобнее и понятнее, чем IP-адрес. Класс `InetAddress` скрывает внутри себя число. Он может работать как с адресами IPv4, так и с IPv6.

Методы-фабрики

Класс `InetAddress` не имеет видимых конструкторов. Чтобы создать объект `InetAddress`, вы должны использовать один из доступных методов-фабрик. *Методы-фабрики* (factory method) — это просто соглашение, в соответствии с которым статические методы класса возвращают экземпляр этого класса. Это делается вместо перегрузки конструктора с различными списками параметров, когда наличие уникальных имен методов делает результат более ясным. Ниже приведены три часто используемых метода-фабрики `InetAddress`.

```
static InetAddress getLocalHost( )
   throws UnknownHostException
static InetAddress getByName(String hostName)
   throws UnknownHostException
static InetAddress[ ] getAllByName(String hostName)
   throws UnknownHostException
```

Метод `getLocalHost()` просто возвращает объект `InetAddress`, представляющий локальный хост. Метод `getByName()` возвращает `InetAddress` хоста, чье имя ему передано. Если эти методы оказываются не в состоянии получить имя хоста, они возбуждают исключение `UnknownHostException`.

Когда одно имя используется для представления нескольких машин в Internet — это обычное явление. В мире Web-серверов это единственный путь предоставления некоторой степени масштабируемости. Метод-фабрика `getAllByName()` возвращает массив `InetAddress`, представляющий все адреса, в которые преобразуется конкретное имя. Он также возбуждает исключение `UnknownHostException` в случае, если не может преобразовать имя в хотя бы один адрес.

`InetAddress` также включает фабричный метод `getDyAddress()`, который принимает IP-адрес и возвращает объект `InetAddress`. Причем могут использоваться как адреса IPv4, так и IPv6.

В следующем примере распечатываются адреса и имена локальной машины, а также двух широко известных Internet-сайтов.

```
// Демонстрация применения InetAddress.
import java.net.*;
class InetAddressTest
{
public static void main(String args[]) throws UnknownHostException {
  InetAddress Address = InetAddress.getLocalHost();
  System.out.println(Address);
  Address = InetAddress.getByName("osborne.com");
  System.out.println(Address);
  InetAddress SW[] = InetAddress.getAllByName("www.nba.com");
  for (int i=0; i<SW.length; i++)
     System.out.println(SW[i]);
}
}
```

Ниже показан вывод, сгенерированный этой программой (конечно, код, который вы увидите на своей машине, может несколько отличаться).

```
default/206.148.209.138
osborne.com/198.45.24.162
www.nba.com/64.5.96.214
www.nba.com/64.5.96.216
```

Методы экземпляра

В классе `InetAddress` также имеется несколько других методов, которые могут быть использованы с объектами, возвращенными методами, о которых мы говорили только что. Некоторые из наиболее часто применяемых методов перечислены в табл. 20.1.

Поиск Internet-адресов осуществляется в серии иерархических кэшированных служб. Это значит, что ваш локальный компьютер может получить определенное отображение имени на IP-адрес автоматически, как для себя, так и для ближайших серверов. Для всех прочих имен он может обращаться к DNS-серверам, откуда получит информацию об IP-адресах. Если такой сервер не имеет информации об определенном адресе, он может обратиться к следующему удаленному сайту и запросить эту информацию у него. Это может продолжаться вплоть до корневого сервера, и упомянутый процесс может потребовать длительного времени, так что разумно построить структуру вашего кода таким образом, чтобы информация об IP-адресах локально кэшировалась, и ее не приходилось искать каждый раз заново.

Таблица 20.1. Часто используемые методы класса `InetAddress`

Метод	Описание
`boolean equals(Object other)`	Возвращает `true`, если объект имеет тот же адрес Internet, что и *other*.
`byte[] getAddress()`	Возвращает байтовый массив, представляющий IP-адрес в порядке байт сети.
`String getHostAddress()`	Возвращает строку, представляющую адрес хоста, ассоциированного с объектом `InetAddress`.
`String getHostName()`	Возвращает строку, представляющую имя хоста, ассоциированного с объектом `InetAddress`.
`boolean isMulticastAddress()`	Возвращает `true`, если адрес является групповым, в противном случае возвращает `false`.
`String toString()`	Возвращает строку, включающую имя хоста и IP-адрес для удобства.

`Inet4Address` и `Inet6Address`

Начиная с версии 1.4, в Java включена поддержка адресов IPv6. В связи с этим были созданы два подкласса `InetAddress`: `Inet4Address` и `Inet6Address`. `Inet4Address` представляет традиционные адреса IPv4, а `Inet6Address` инкапсулируют адреса IPv6 нового стиля. Поскольку оба они являются подклассами `InetAddress`, ссылки `InetAddress` могут указывать на них. Это единственный способ, благодаря которому удалось добавить в Java функциональность IPv6, не нарушая работы существующего кода и не добавляя большого количества новых классов. В большинстве случаев вы просто можете использовать `InetAddress`, работая с IP-адресами, поскольку этот класс приспособлен для обоих стилей.

Клиентские сокеты TCP/IP

Сокеты TCP/IP применяются для реализации надежных двунаправленных, постоянных соединений между точками — хостами в Internet на основе потоков. Сокет может использоваться для подключения системы ввода-вывода Java к другим программам, которые могут находиться как на локальной машине, так и на любой другой машине в Internet.

На заметку! *Аплеты могут устанавливать сокетные соединения только с тем хостом, с которого они были загружены. Это ограничение введено в связи с тем, что было бы опасно для аплетов, загруженных через брандмауэр, иметь доступ к любой произвольной машине.*

В Java существуют два вида сокетов TCP. Один — для серверов, другой — для клиентов. Класс `ServerSocket` предназначен быть "слушателем", который ожидает подключения клиентов прежде, чем что-либо делать. То есть `ServerSocket` предназначен для серверов. Класс `Socket` предназначен для клиентов. Он разработан так, чтобы соединяться с серверными сокетами и инициировать обмен по протоколу. Поскольку клиентские советы наиболее часто применяются в Java-приложениях, их мы и рассмотрим здесь. В табл. 20.2 описаны два конструктора, используемые для создания клиентских советов.

Таблица 20.2. Конструкторы класса `Socket`

Конструктор	Описание
`Socket(String hostName, int port) throws UnknownHostException, IOException`	Создает сокет, подключенный к именованному хосту и порту.
`Socket(InetAddress ipAddress, int port) throws IOException`	Создает сокет, используя ранее существующий объект `InetAddress` и порт.

`Socket` определяет несколько методов экземпляров. Например, `Socket` может быть просмотрен в любое время на предмет извлечения информации об адресе и порте, ассоциированной с ним. Для этого применяются методы, перечисленные в табл. 20.3.

Таблица 20.3. Методы экземпляра `Socket`

Метод	Описание
`InetAddress getInetAddress()`	Возвращает `InetAddress`, ассоциированный с объектом `Socket`. В случае если сокет не подключен, возвращает `null`.
`int getPort()`	Возвращает удаленный порт, к которому подключен вызывающий объект `Socket`. Если сокет не подключен, возвращает 0.
`int getLocalPort()`	Возвращает локальный порт, к которому привязан вызывающий объект `Socket`. Если сокет не привязан, возвращает `-1`.

Вы можете получить доступ к входному и выходному потокам, ассоциированным с `Socket`, с использованием методов `getInputStream()` и `getOuputStream()`, которые описаны в табл. 20.4. Каждый из них может возбуждать исключение `IOException`, если сокет стал недействительным из-за утери соединения. Эти потоки используются точно так же, как потоки ввода-вывода, рассмотренные в главе 19, для получения и приема данных.

Таблица 20.4. Методы доступа к входному и выходному потокам, ассоциированным с `Socket`

Метод	Описание
`InputStream getInputStream() throws IOException`	Возвращает `InetAddress`, ассоциированный с вызывающим сокетом.
`OutputStream getOutputStream() throws IOException`	Возвращает `OutputStream`, ассоциированный с вызывающим сокетом.

Доступно также еще несколько других методов, включая `connect()`, позволяющий специфицировать новое подключение, `isConnected()`, возвращающий `true`, если сокет подключен к серверу, `isBound()`, возвращающий `true`, если сокет привязан к адресу, и `isClosed()`, возвращающий `true`, когда сокет закрыт.

Следующая программа представляет простой пример применения `Socket`. Она открывает соединение с портом whois (порт 43) на сервере InterNIC, посылает сокету аргументы командной строки, а затем печатает возвращенные данные. InterNIC пытается трактовать аргумент как зарегистрированное доменное имя Internet, а затем возвращает IP-адрес и контактную информацию для этого сайта.

```
// Демонстрация работы с сокетами.
import java.net.*;
import java.io.*;
```

```
class Whois {
public static void main(String args[]) throws Exception {
  int c;

  // Создает сокетное соединение с internic.net, порт 43.
  Socket s = new Socket("internic.net", 43);

  // Получает входной и выходной потоки.
  InputStream in = s.getInputStream();
  OutputStream out = s.getOutputStream();

  // Конструирует строку запроса.
  String str = (args.length == 0 ? "osborne.com" : args[0]) + "\n";

  // Преобразует в байты.
  byte buf[] = str.getBytes();

  // Посылает запрос.
  out.write(buf);

  // Читает и отображает ответ.
  while ((c = in.read()) != -1) {
     System.out.print((char) c);
  }
  s.close();
}
}
```

Если, к примеру, вы запросите информацию об `osborne.com`, то получите нечто вроде следующего:

```
Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: OSBORNE.COM
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: http://www.networksolutions.com
   Name Server: NS1.EPPG.COM
   Name Server: NS2.EPPG.COM
.
.
.
```

Вот как работает эта программа. Сначала конструируется `Socket`, специфицирующий имя хоста “internic.net” и номер порта 43. `Internic.net` — это Web-сайт InterNIC, обрабатывающий запросы whois. Порт 43 предназначен именно для этой службы. Затем и входной, и выходной потоки открываются в сокете. Далее конструируется строка, содержащая имя Web-сайта, информацию о котором вы хотите получить. В данном случае, если никакой сайт не указан в командной строке, используется “osborne.com”. Строка преобразуется в байтовый массив и отправляется в сокет. После этого ответ читается из сокета и результат отображается на экране.

Класс URL

Предыдущий пример довольно-таки невразумителен, поскольку в настоящее время в Internet не ассоциируется со старыми протоколами, такими как whois, finger и FTP. Здесь царствует WWW — всемирная паутина (World Wide Web). Web — слабо связанная коллекция высокоуровневых протоколов и форматов файлов, унифицированным образом используемых Web-браузерами. Одним из наиболее важных аспектов Web является то, что Тим Бернерс-Ли (Tim Berners-Lea) предложил масштабируемый способ нахождения всех ресурсов в Internet. Как только вы можете однозначно именовать что-либо, это становится очень мощной парадигмой. Именно это и делает унифицированный локатор ресурсов (Uniform Resource Locator — URL).

URL обеспечивает довольно четкую форму уникальной идентификации адресной информации в Web. Внутри библиотеки классов Java класс `URL` представляет простой согласованный программный интерфейс для доступа к информации по всей сети Internet посредством использования URL.

Все URL разделяют один и тот же базовый формат, хотя и допускающий некоторые вариации. Приведем два примера: `http://www.osborne.com/` и `http://www.osborne.com:80/index.htm`. Спецификация URL основана на четырех компонентах. Первый — используемый протокол, отделяемый от остальной части локатора двоеточием (`:`). Распространенными протоколами являются HTTP, FTP, gopher и file, хотя в наши дни почти все осуществляется через HTTP (фактически большинство браузеров корректно работают, даже если вы исключите из спецификации URL фрагмент “http://”). Второй компонент — имя хоста или IP-адрес, используемый хостом; он отделяется слева двойным слэшем (`//`), а справа — слэшем (`/`) или, необязательно — двоеточием (`:`). Третий компонент — номер порта, является необязательным параметром, отделяемым слева от имени хоста двоеточием, а справа — слэшем (`/`) (Если 80 является портом по умолчанию для протокола HTTP, то указывать “:80” излишне.) Четвертая часть — действительный путь к файлу. Большинство серверов HTTP добавляют имя файла `index.html` или `index.htm` к URL, которые указывают непосредственно на какой-то каталог. Таким образом, `http://www.osborne.com/` — это то же самое, что и `http://www.osborne.com/index.htm`.

Java-класс `URL` имеет несколько конструкторов; каждый из них может возбуждать исключение `MalformedURLException`. Одна из часто используемых форм специфицирует URL в виде строки, идентичной тому, что вы видите в браузере:

```
URL(String urlSpecifier) throws MalformedURLException
```

Следующие две формы конструктора позволяют вам разбить URL на части-компоненты:

```
URL(String protocolName, String hostName, int port, String path)
   throws MalformedURLException
URL(String protocolName, String hostName, String path)
   throws MalformedURLException
```

Другой часто используемый конструктор позволяет указывать существующий URL в качестве ссылочного контекста, и затем создать из этого контекста новый URL. Хотя это звучит несколько запутано, на самом деле это очень просто и удобно.

```
URL(URL urlObj, String urlSpecifier) throws MalformedURLException
```

Следующий пример создает URL страницы загрузки Osborne, а затем просматривает его свойства:

```
// Демонстрация применения URL.
import java.net.*;
class URLDemo {
public static void main(String args[]) throws MalformedURLException {
  URL hp = new URL("http://www.osborne.com/downloads");
  System.out.println("Протокол: " + hp.getProtocol());
  System.out.println("Порт: " + hp.getPort());
  System.out.println("Хост: " + hp.getHost());
  System.out.println("Файл: " + hp.getFile());
  System.out.println("Целиком: " + hp.toExternalForm());
}
}
```

Запустив это, вы получите:

```
Протокол: http
Порт: -1
Хост: www.osborne
Файл: /downloads
Целиком: http://www.osborne/downloads
```

Обратите внимание на порт `-1`; это означает, что порт явно не установлен. Передав объект `URL`, вы можете извлечь данные, ассоциированные с ним. Чтобы получить доступ к действительным битам или информации по `URL`, создайте из него объект `URLConnection`, используя его метод `openConnection()`, как показано ниже:

```
urlc = url.openConnection()
```

`openaConnection()` имеет следующую общую форму:

```
URLConnection openConnection() throws IOException
```

Он возвращает объект `URLConnection`, ассоциированный с вызывающим объектом `URL`. Обратите внимание, что он может возбуждать исключение `IOException`.

URLConnection

`URLConnection` — это класс общего назначения, предназначенный для доступа к атрибутам удаленного ресурса. Однажды установив соединение с удаленным сервером, вы можете использовать `URLConnection` для просмотра свойств удаленного объекта, прежде чем транспортировать его локально. Эти атрибуты представлены в спецификации протокола HTTP и, как таковые, имеют смысл только для объектов `URL`, использующих протокол HTTP.

`URLConnection` определяет несколько методов. Некоторые из них перечислены в табл. 20.5.

Обратите внимание, что `URLConnection` определяет несколько методов, управляющих заголовочной информацией. Заголовок состоит из пар ключей и значений, представленных в виде строк. Используя `getHeaderField()`, вы можете получить значение, ассоциированное с ключом заголовка. Вызывая `getHeaderField()`, можно получить карту, содержащую все заголовки. Несколько стандартных заголовочных полей доступны непосредственно через такие методы, как `getDate()` и `getContentType()`.

Таблица 20.5. Некоторые методы класса `URLConnection`

Метод	Описание
`int getContentLength()`	Возвращает размер содержимого, ассоциированного с ресурсом. Если длина недоступна, возвращается `-1`.
`String getContentType()`	Возвращает тип содержимого, найденного в ресурсе. Это значение поля заголовка `content-type`. Возвращает `null`, если тип содержимого недоступен.
`long getDate()`	Возвращает время и дату ответа, представленное в миллисекундах, прошедших с 1 января 1970 г.
`long getExpiration()`	Возвращает время и дату устаревания ресурса, представленное в миллисекундах, прошедших с 1 января 1970 г. Если дата устаревания недоступна, возвращается ноль.
`String getHeaderField(int idx)`	Возвращает значение заголовочного поля по индексу *`idx`*. (Индексы полей заголовка нумеруются, начиная с 0.) Возвращает `null`, если значение *`idx`* превышает количество полей.
`String getHeaderField(String fieldName)`	Возвращает значение заголовочного поля, чье имя указано в *`fieldName`*. Возвращает `null`, если указанное поле не найдено.
`String getHeaderFieldKey(int idx)`	Возвращает ключ заголовочного поля по индексу *`idx`*. (Индексы полей заголовка нумеруются, начиная с 0.) Возвращает `null`, если значение *`idx`* превышает количество полей.
`Map<String, List<String>> getHeaderFields()`	Возвращает карту, содержащую все заголовочные поля вместе с их значениями.
`long getLastModified()`	Возвращает время и дату последней модификации ресурса, представленные в миллисекундах, прошедших после 1 января 1970 г. Если эта информация недоступна, возвращается ноль.
`InputStream getInputStream() throws IOException`	Возвращает `InputStream`, привязанный к ресурсу. Данный поток может использоваться для получения содержимого ресурса.

Следующий пример создает `URLConnection`, используя метод `openConnection()` объекта `URL`, а затем применяет его для проверки свойств и содержимого документа:

```
// Демонстрация применения URLConnection.
import java.net.*;
import java.io.*;
import java.util.Date;
class UCDemo
{
public static void main(String args[]) throws Exception {
  int c;
  URL hp = new URL("http://www.internic.net");
  URLConnection hpCon = hp.openConnection();
  // получить дату
  long d = hpCon.getDate();
```

```
    if(d==0)
      System.out.println("Нет информации о дате.");
    else
      System.out.println("Дата: " + new Date(d));

    // получить тип содержимого
    System.out.println("Тип содержимого: " + hpCon.getContentType());

    // получить дату устаревания
    d = hpCon.getExpiration();
    if(d==0)
      System.out.println("Нет информации о сроке действия.");
    else
      System.out.println("Устареет: " + new Date(d));

    // получить дату последней модификации
    d = hpCon.getLastModified();
    if(d==0)
      System.out.println("Нет информации о дате последней модификации.");
    else
      System.out.println("Дата последней модификации: " + new Date(d));

    // получить длину содержимого
    int len = hpCon.getContentLength();
    if(len == -1)
      System.out.println("Длина содержимого недоступна.");
    else
      System.out.println("Длина содержимого: " + len);
    if(len != 0) {
      System.out.println("=== Содержимое ===");
      InputStream input = hpCon.getInputStream();
      int i = len;
      while (((c = input.read()) != -1)) { // && (--i > 0)) {
        System.out.print((char) c);
      }
      input.close();
    } else {
      System.out.println("Содержимое недоступно.");
    }
  }
  }
```

Эта программа устанавливает HTTP-соединение с `www.internic.net` через порт 80. Затем она отображает несколько заголовочных значений и извлекает содержимое. Приведем первые строки вывода (точное их содержание будет меняться со временем):

```
Дата: Thu Jun 08 14:41:35 CDT 2006
Тип содержимого: text/html
Нет информации о сроке действия.
Дата последней модификации: Wed Oct 05 19:49:29 CDT 2005
Длина содержимого: 4917
=== Содержимое ===
<html>
<head>
<title>InterNIC | The Internet's Network Information Center</title>
<meta name="keywords"
  content="internic,network information, domain registration">
```

```
<style type="text/css">
<!--
p, li, td, ul { font-family: Arial, Helvetica, sans-serif}
-->
</style>
</head>
```

HttpURLConnection

Java предлагает подкласс `URLConnection`, обеспечивающий поддержку соединений HTTP. Этот класс называется `HttpURLConnection`. Вы получаете `HttpURLConnection` точно так же, как было показано — вызовом `openConnection()` объекта `URL`, но результат следует приводить к типу `HttpURLConnection`. (Конечно, необходимо убедиться в том, что вы действительно открыли соединение HTTP.) Получив ссылку на объект `HttpURLConnection`, вы можете вызывать любые его методы, унаследованные от `URLConnection`. Вы также можете использовать любые методы, определенные в `HttpURLConnection`. Некоторые методы перечислены в табл. 20.6.

Таблица 20.6. Некоторые методы класса `HttpURLConnection`

Метод	Описание
`static boolean getFollowRedirects()`	Возвращает `true`, если автоматически следует перенаправление, и `false` в противном случае.
`String getRequestMethod()`	Возвращает строковое представление метода выполнения запроса. По умолчанию используется метод GET. Доступны другие методы, такие как POST.
`int getResponseCode() throws IOException`	Возвращает код ответа HTTP. Если код ответа не может быть получен, возвращается `-1`. При разрыве соединения возбуждается исключение `IOException`.
`String getResponseMessage() throws IOException`	Возвращает сообщение ответа, ассоциированное с кодом ответа. Если никакого сообщения недоступно, возвращает `null`.
`static void setFollowRedirects(boolean` *how*`)`	Если *how* равно `true`, значит, перенаправление осуществляется автоматически. Если же *how* равно `false`, значит, этого не происходит. По умолчанию перенаправление осуществляется автоматически.
`void setRequestMethod(String` *how*`) throws ProtocolException`	Устанавливает метод, которым выполняются HTTP-запросы, в соответствии с указанным в *how*. По умолчанию принят метод GET, но доступны также другие варианты, такие как POST. Если указано неправильное значение *how*, возбуждается исключение `ProtocolException`.

В следующей программе демонстрируется работа с `HttpURLConnection`. Сначала она устанавливает соединение с `www.google.com`. Затем отображает метод запроса, код ответа и сообщение ответа. И, наконец, отображает ключи и значения в заголовке ответа.

```
// Демонстрация применения HttpURLConnection.
import java.net.*;
import java.io.*;
import java.util.*;
class HttpURLDemo
{
public static void main(String args[]) throws Exception {
  URL hp = new URL("http://www.google.com");
  HttpURLConnection hpCon = (HttpURLConnection) hp.openConnection();

  // Отображение метода запроса.
  System.out.println("Метод запроса: " + hpCon.getRequestMethod());

  // Отображение кода ответа.
  System.out.println("Код ответа: " + hpCon.getResponseCode());

  // Отображение сообщения ответа.
  System.out.println("Сообщение ответа: " + hpCon.getResponseMessage());

  // Получить список полей заголовка и набор его ключей.
  Map<String, List<String>> hdrMap = hpCon.getHeaderFields();
  Set<String> hdrField = hdrMap.keySet();
  System.out.println("\nЗдесь следует заголовок:");

  // Отобразить все ключи и значения заголовка.
  for(String k : hdrField) {
     System.out.println("Ключ: " + k + " Значение: " + hdrMap.get(k));
  }
}
}
```

Вывод этой программы показан ниже (разумеется, точный ответ, возвращенный `www.google.com`, будет меняться с течением времени).

```
Метод запроса: GET
Код ответа: 200
Сообщение ответа: OK

Здесь следует заголовок:
Ключ: Set-Cookie Value:
[PREF=ID=4fbe939441ed966b:TM=1150213711:LM=1150213711:S=Qk81
WCVtvYkJOdh3; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/;
   domain=.google.com]
Ключ: null Value: [HTTP/1.1 200 OK]
Ключ: Date Value: [Tue, 13 Jun 2006 15:48:31 GMT]
Ключ: Content-Type Value: [text/html]
Ключ: Server Value: [GWS/2.1]
Ключ: Transfer-Encoding Value: [chunked]
Ключ: Cache-Control Value: [private]
```

Обратите внимание, как отображаются ключи и значения заголовка. Во-первых, карта ключей и заголовков получается вызовом `getHeaderFields()` (который унаследован от `URLConnection`). Затем набор ключей заголовка извлекается вызовом `keySet()` карты. Далее выполняется циклический проход по всему набору посредством стиля “for-each” цикла `for`. Значение, ассоциированное с каждым ключом, получается из карты вызовом `get()`.

Класс URI

Относительно недавним дополнением к Java стал класс URI, инкапсулирующий *универсальный идентификатор ресурса* (Uniform Resource Identifier — URI). URI очень похож на URL. На самом деле URL представляет собой подмножество URI. URI предоставляет стандартный способ идентификации ресурсов. URL также описывает доступ к ресурсу.

Cookie-наборы

Пакет java.net включает классы и интерфейсы, помогающие управлять cookie-наборами, которые могут использоваться для создания HTTP-сеансов с поддержкой состояния (в противоположность таковым без поддержки состояния). К таким классам относятся CookieHandler, CookieManager и HttpCookie, а к интерфейсам — CookiePolicy и CookieStore. Все, кроме CookieHandler, были добавлены в Java SE 6. (CookieHandler появился в JDK 5.) Создание HTTP-сеанса с поддержкой состояния выходит за рамки настоящей книги.

На заметку! *Информацию о применении cookie-наборов с сервлетами читайте в главе 31.*

Серверные сокеты TCP/IP

Как уже упоминалось, в Java имеются различные классы сокетов, которые должны применяться для создания серверных приложений. Класс ServerSocket используется для создания серверов, которые прослушивают обращения как локальных, так и удаленных клиентских программ, желающих установить соединения с ними через открытые порты. ServerSocket довольно-таки сильно отличается от обычных Socket. Когда вы создаете ServerSocket, он регистрирует себя в системе в качестве заинтересованного в клиентских соединениях. Конструкторы ServerSocket отражают номер порта, через который вы хотите принимать соединения, а также — необязательно — длину очереди для данного порта. Длина очереди сообщает системе о том, сколько клиентских соединений можно удерживать, прежде чем начать просто отклонять попытки подключения. По умолчанию установлено 50. При определенных условиях конструкторы могут возбуждать исключение IOException. Конструкторы этого класса описаны в табл. 20.7.

Таблица 20.7. Конструкторы класса ServerSocket

Конструктор	Описание
ServerSocket(int *port*) throws IOException	Создает серверный сокет на указанном порте с длиной очереди 50.
ServerSocket(int *port*, int *maxQueue*) throws IOException	Создает серверный сокет на указанном порте с максимальной длиной очереди в *maxQueue*.
ServerSocket(int *port*, int *maxQueue*, InetAddress *localAddress*) throws IOException	Создает серверный сокет на указанном порте с максимальной длиной очереди в *maxQueue*. На групповом хосте *localAddress* указывает IP-адрес, к которому привязан сокет.

ServerSocket включает метод по имени accept(), представляющий собой блокирующий вызов, который будет ожидать от клиента инициации соединений, и затем возвратит нормальный объект Socket, который далее может служить для взаимодействия с клиентом.

Дейтаграммы

Сетевое взаимодействие в стиле TCP/IP подходит для большинства сетевых нужд. Оно обеспечивает сериализуемые, предсказуемые и надежные потоки пакетов данных. Тем не менее, это обходится далеко не даром. TCP включает множество сложных алгоритмов управления потоками в нагруженных сетях, а также самые пессимистические предположения относительно утери пакетов. Это порождает в некоторой степени неэффективный способ транспортировки данных. В качестве альтернативы можно использовать дейтаграммы.

Дейтаграммы (datagramms) — это порции информации, передаваемые между машинами. В некотором отношении они подобны сильным броскам тренированного, но подслеповатого кетчера в сторону третьего бейсмена. Как только дейтаграмма запущена в сторону нужной цели, нет никаких гарантий, что она достигнет цели, или кто-нибудь окажется на месте, чтобы ее подхватить. Точно так же, когда дейтаграмма принимается, нет никакой гарантии, что она не была повреждена в пути, или что ее отправитель все еще ожидает ответа.

Java реализует дейтаграммы поверх протокола UDP, используя для этого два класса: DatagramPacket — контейнер данных, и DatagramSocket — механизм, используемый для обслуживания DatagramPacket. Рассмотрим их более подробно.

DatagramSocket

DatagramSocket определяет четыре общедоступных конструктора:

```
DatagramSocket( ) throws SocketException
DatagramSocket(int port) throws SocketException
DatagramSocket(int port, InetAddress ipAddress) throws SocketException
DatagramSocket(SocketAddress address) throws SocketException
```

Первый конструктор создает DatagramSocket, связанный с любым незанятым портом локального компьютера. Второй создает DatagramSocket, связанный с портом, указанным в *port*. Третий конструирует DatagramSocket, связанный с указанным портом и InetAddress. Четвертый конструирует DatagramSocket, связанный с заданным SocketAddress. SocketAddress — абстрактный класс, реализуемый конкретным классом InetSocketAddress. InetSocketAddress инкапсулирует IP-адрес с номером порта. Все конструкторы могут возбуждать исключение SocketException в случае возникновения ошибок во время создания сокета.

DatagramSocket определяет множество методов. Два наиболее важных из них — это send() и receive(), которые представлены ниже:

```
void send(DatagramPacket packet) throws IOException
void receive(DatagramPacket packet) throws IOException
```

Метод send() отправляет порту пакет, указанный в *packet*. Метод приема ожидает приема через порт пакета, указанного в *packet*, и возвращает результат.

Другие методы предоставляют вам доступ к различным атрибутам, ассоциированным с DatagramSocket. Эти методы перечислены в табл. 20.8.

Таблица 20.8. Методы класса DatagramSocket

Метод	Описание
`InetAddress getInetAddress()`	Если сокет подключен, возвращается адрес. В противном случае возвращается `null`.
`int getLocalPort()`	Возвращает номер локального порта.
`int getPort()`	Возвращает номер порта, к которому подключен сокет. Возвращает `-1`, если сокет не подключен ни к какому порту.
`boolean isBound()`	Возвращает `true`, если сокет привязан к адресу, в противном случае возвращает `false`.
`boolean isConnected()`	Возвращает `true`, если сокет подключен к серверу, в противном случае возвращает `false`.
`void setSoTimeout(int millis) throws SocketException`	Устанавливает период таймаута в число миллисекунд, переданное в *millis*.

DatagramPacket

DatagramPacket определяет множество конструкторов. Вот четыре из них:

```
DatagramPacket(byte data[ ], int size)
DatagramPacket(byte data[ ], int offset, int size)
DatagramPacket(byte data[ ], int size, InetAddress ipAddress, int port)
DatagramPacket(byte data[ ], int offset, int size, InetAddress ipAddress, int port)
```

Первый конструктор специфицирует буфер, который будет принимать данные, и размер пакета. Он используется для приема данных через DatagramSocket. Вторая форма позволяет вам указать смещение в буфере, куда должны быть помещены данные. Третья форма указывает целевой адрес и порт, используемые DatagramSocket для определения того, куда данные пакета будут отправлены. Четвертая форма передает пакеты, начиная с указанного смещения в данных. Первые две формы следует воспринимать как построение "ящика", а вторые две — как "начинку" и адресацию на конверте.

DatagramPacket определяет несколько методов, включая представленные здесь, которые предоставляют доступ к адресу и номеру порта пакета, наряду с raw-данными и их длиной (табл. 20.9). В общем случае методы get используются в принимаемых пакетах, а методы set — в отправляемых.

Таблица 20.9. Методы класса DatagramPacket

Метод	Описание
`InetAddress getAddress()`	Возвращает адрес источника (для принимаемых дейтаграмм) или места назначения (для отправляемых дейтаграмм).
`byte[] getData()`	Возвращает байтовый массив данных, содержащихся в дейтаграмме. В основном используется для извлечения данных из дейтаграммы после ее приема.

Окончание табл. 20.9

Метод	Описание
`int getLength()`	Возвращает длину корректных данных, содержащихся в байтовом массиве, который должен быть возвращен из метода `getData()`. Это может не совпадать с полной длиной байтового массива.
`int getOffset()`	Возвращает начальный индекс данных.
`int getPort()`	Возвращает номер порта.
`void setAddress(InetAddress ipAddress)`	Устанавливает адрес, по которому отправляется пакет. Адрес указывается в *ipAddress*.
`void setData(byte[] data)`	Устанавливает данные в *data*, смещение — в ноль, а длину — в количество байт *data*.
`void setData(byte[] data, int idx, int size)`	Устанавливает данные в *data*, смещение — в *idx*, а длину в *size*.
`void setLength(int size)`	Устанавливает длину пакета в *size*.
`void setPort(int port)`	Устанавливает порт в *port*.

Пример работы с дейтаграммами

В следующем примере реализуется очень простое сетевое взаимодействие: клиент и сервер. Сообщения набираются в окне на сервере и передаются по сети на сторону клиента, где и отображаются.

```
// Демонстрация применения дейтаграмм.
import java.net.*;
class WriteServer {
public static int serverPort = 998;
public static int clientPort = 999;
public static int buffer_size = 1024;
public static DatagramSocket ds;
public static byte buffer[] = new byte[buffer_size];
public static void TheServer() throws Exception {
  int pos=0;
  while (true) {
    int c = System.in.read();
    switch (c) {
      case -1:
         System.out.println("Сервер завершил работу.");
         return;
      case '\r':
         break;
      case '\n':
         ds.send(new DatagramPacket(buffer,pos,
             InetAddress.getLocalHost(),clientPort));
         pos=0;
         break;
      default:
         buffer[pos++] = (byte) c;
    }
  }
}
```

```
public static void TheClient() throws Exception {
   while(true) {
     DatagramPacket p = new DatagramPacket(buffer, buffer.length);
     ds.receive(p);
     System.out.println(new String(p.getData(), 0, p.getLength()));
   }
}

public static void main(String args[]) throws Exception {
   if(args.length == 1) {
     ds = new DatagramSocket(serverPort);
     TheServer();
   } else {
     ds = new DatagramSocket(clientPort);
     TheClient();
   }
}
}
```

Этот пример программы ограничен конструктором `DtagramSocket` для запуска между портами локальной машины. Чтобы использовать программы, выполните следующую команду:

```
java WriteServer
```

в одном окне; это будет клиент. Затем в другом окне запустите

```
java WriteServer 1
```

Это будет сервер. Все, что вы напечатаете в окне сервера, после получения символа перевода строки будет отправлено в клиентское окно.