Интернет програмиране с Java – част 5

Автори:

Светлин Наков, СУ "Св. Климент Охридски"
Web-site: http://www.nakov.com

Борис Червенков, "Информационно обслужване" АД
E-mail: boris@abv.bg

Последна промяна: 10.07.2002

В предходната част от курса започнахме темата за създаването на Java аплети. В тази част ще довършим изложението по тази тема и ще продължим с темата "Web-приложения".

JAVA аплети и сигурност

Сигурността на аплетите е важна тяхна черта. Никой потребител не би се съгласил да разглежда сайтове с аплети, ако те могат да пишат свободно по диска му, ако могат да откраднат негова лична информация, да изпращат email-и от негово име или да извършват някаква друга злонамерена дейност. Ето защо по идея аплетите работят с ограничени права. Сигурността в Java е част от самата платформа и се конфигурира от един специален файл с име `java.policy`. В зависимост от правата, които Web-браузърът иска да даде на аплета, се подготвя съответен файл, който ги описва и виртуалната машина се конфигурира по него. В някои браузъри правата могат да се настройват и се допуска възможност потребителят да дава пълно доверие на определени сайтове, с което аплетите се освобождават от ограниченията си. Ако потребителят не е посочил нещо друго, се използват стандартните настройки за правата на аплетите, които налагат следните ограничения: Аплетите не могат да четат и пишат по диска на машината, на която се изпълняват. Не могат да осъществяват достъп до чужда памет, дори в операционни системи, в които няма защита на паметта. Не могат да отварят сокет до произволен сървър в Интернет. Могат да отварят сокет само до хост-а, от който са заредени. Не могат да извикват директно native код. Не могат да предизвикат претоварване или забиване на машината, на която се изпълняват. Последното е възможно да се случи в някои специфични ситуации, но това се дължи на грешки и пропуски в сигурността на съответните браузъри и виртуалните машини, които те използват. Трябва да обърнем специално внимание на сокетите. Свидетели сме на много аплети, които извършват активна мрежова дейност, като например аплети за Chat, аплети за четене на email, аплети за изпращане на email, различни игри и т.н. Всички те използват сокет базирана комуникация и изглежда, че отварят сокет към Интернет. Например при изпращането на поща аплетът комуникира със зададен от потребителя SMTP сървър. Това, обаче, не става директно, както при обикновените програми на Java. Аплетите имат право да се свързват чрез сокет само до сървъра, от който са заредени, т.е. към хост-а върнат от метода `getCodeBase().getHost()`. Ето защо аплети, които не са заредени от някой Web-сървър, а локално от файловата система, чрез отваряне на локален HTML файл, нямат право да отварят никакви сокети. Това защитава потребителите от атака чрез HTML документи съдържащи аплети със злонамерено действие. Всички аплети, които изглежда, че отварят сокети към Интернет, всъщност отварят сокети към сървъра, от който са заредени и от там получават пренасочване към заявения хост, т.е. използват Web-сървъра като прокси (междинен пренасочващ сървър). Когато се наложи да пишем аплет, който комуникира чрез сокети, е необходимо на Web-сървъра, където се хоства този аплет да пуснем някакъв допълнителен сървър, който осигурява комуникацията на аплета с услугата, до която той трябва да осъществява достъп. Разбира се, това трябва да става след успешна автентикация на потребителя в системата. За целта най-удобно е сървърът, който се грижи за комуникацията на аплета да се интегрира в Web-сървъра, за да може да използва информацията от сесията на потребителя, който е изпълнил аплета. Как точно може да стане това ще изясним по-нататък в нашия курс в частта за Web-приложения.

Нека сега дадем още един пример за аплет, с който да демонстрираме работа с компонентите на AWT и обработката на събитията, възникнали в резултат от действията на потребителя. Да си поставим за задача реализацията на прост калкулатор, който събира числа. Трябват ни две полета за двете събираеми, още едно поле за резултата и един бутон за събиране. За демонстрация на работата със шрифтове ще добавим в нашия аплет-калкулатор заглавен текст със сянка. За демонстрация на работата със събития от мишката при щракване върху аплета цветът му ще се променя, а при отпускане бутона на мишката ще се възстановява обратно. Ето една примерна реализация на аплета:

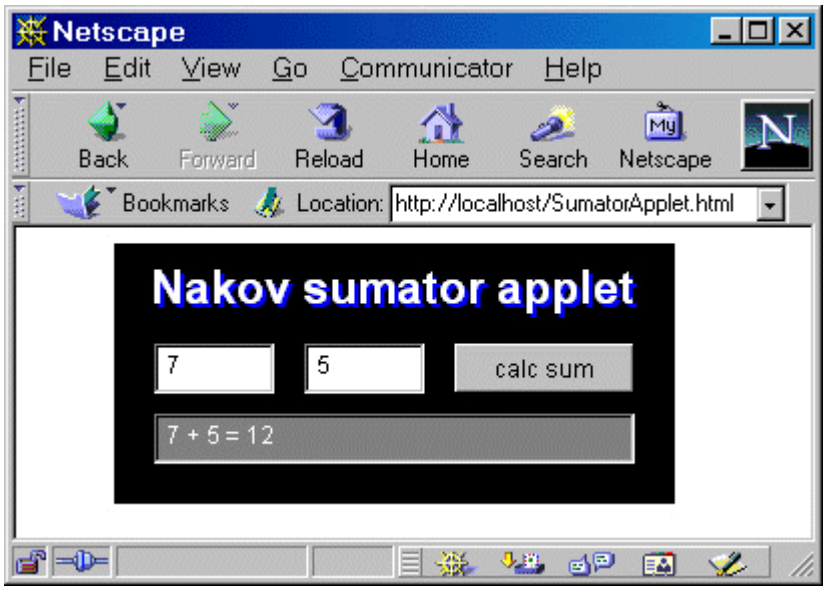
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class SumatorApplet extends Applet {
    TextField number1Field = new TextField();
    TextField number2Field = new TextField();
    Button calcButton = new Button();
    TextField sumField = new TextField();
    Color lastBackground;

    public void init() {
        this.setBackground(Color.black);
        // Set layout manager to null
        this.setLayout(null);
        // Create the first text field
        number1Field.setBounds(
            new Rectangle(20, 50, 60, 25));
        number1Field.setBackground(Color.white);
        this.add(number1Field, null);
        // Create the second text field
        number2Field.setBounds(
            new Rectangle(95, 50, 60, 25));
        number2Field.setBackground(Color.white);
        this.add(number2Field, null);
        // Create the "calculate sum" button
        calcButton.setBounds(
            new Rectangle(170, 50, 90, 25));
        calcButton.addActionListener(
            new ActionListener() {
                public void actionPerformed(
                    ActionEvent e) {
                        calcSum();
                }
            });
        calcButton.setLabel("calc sum");
        this.add(calcButton, null);
        // Create the result text field
        sumField.setEditable(false);
        sumField.setBackground(Color.gray);
        sumField.setForeground(Color.white);
        sumField.setBounds(
            new Rectangle(20, 85, 240, 25));
        this.add(sumField, null);
    }

    public boolean mouseDown(Event evt,
    int x, int y) {
        lastBackground = this.getBackground();
        this.setBackground(Color.red);
        return true;
    }

    public boolean mouseUp(Event evt,
    int x, int y) {
        this.setBackground(lastBackground);
        return true;
    }

    private void calcSum() {
        try {
            long s1 = new Long(number1Field.
                getText()).longValue();
```

```
                long s2 = new Long(number2Field.
                    getText()).longValue();
                sumField.setText(s1 + " + " +
                    s2 + " = " + (s1+s2));
            } catch (Exception ex) {
                sumField.setText("Error!");
            }
        }

        public void paint(Graphics g) {
            super.paint(g);
            Font font = new Font(
                "Dialog", Font.BOLD, 23);
            g.setFont(font);
            g.setColor(Color.blue);
            g.drawString(
                "Nakov sumator applet", 20, 32);
            g.setColor(Color.white);
            g.drawString(
                "Nakov sumator applet", 18, 30);
        }

        public static void main(String[] args) {
            Frame frame = new Frame("Sumator");
            frame.setSize(280,160);
            SumatorApplet applet =
                new SumatorApplet();
            applet.init();
            frame.add(applet);
            frame.setVisible(true);
            applet.start();
        }
    }
```

В инициализационната част на аплета първо се задава стойност `null` за Layout Manager. Layout Manager-ът служи за подреждане на AWT компонентите в един AWT контейнер и може да е много полезен при създаване на форми, които могат да променят размерите си. В случая искаме да работим с абсолютни координати и размери на компонентите, а не с размери и координати, определени от Layout Manager-а и затова задаваме за LayoutManager стойност null, понеже по подразбиране тази стойност е друга. След това създаваме компонентите една по една – първото текстово поле, второто текстово поле, бутонът. За да прихванем събитието "натискене на бутона за сумиране", използваме методът `addActionListener`, който приема като параметър обект от клас, който имплементира интерфейса `ActionListener`. За спестяване на някои неудобства използваме дефиниция на място на анонимен клас, който реализира ActionListener интерфейса и в метода за натискане на бутон `actionPerformed()` извиква метода за изчисляване на сумата `calcSum()`. За прихващане на събитията от мишката има два начина. Единият, който ние сме използвали е да се припокрият методите `MouseDown()`, `MouseUp()` и т.н. на базовия клас, а другият е да се добави MouseListener с метода `addMouseListener()` по начин подобен на този с добавянето на ActionListener. Процедурата за пресмятане на резултата взима стойностите от двете текстови полета, превръща ги в числа и показва резултата от събирането в полето за резултата. Ако не успее при превръщането тескта от полетата в числа или ако се случи препълване или някаква друга грешка, резултатът е "Error!". За демонстрация на работата с шрифтове в метода paint() се отпечатва текст със сянка. Важно е методът `paint()` да извиква `paint()` метода на базовия си клас (`super.paint()`), за да могат компонентите, добавени в аплета да се пречертават всеки път, когато аплетът се пречертава. В нашата реализация на `paint()` метода след извикването на базовия `paint()` метод, чертането продължава със задаване на шрифта и цвета на текста и отпечатването му. След това цветът се сменя и се отпечатва същия текст, изместен с 2 позиции нагоре и наляво. Така се създава впечатлението за сянка. Нашият аплет има още една интересна възможност – може да работи и като самостоятелна програма. За целта той има `main()` метод, в който се създават аплета и един обект `java.awt.Frame`, след което във Frame-а се поставя аплета, задават му се размерите и се показва на екрана. Така аплетът може да бъде стартиран за тестови цели като самостоятелна програма, а когато е готов, може да се тества и в браузера, защото, както знаете аплетите се държат по различен начин в различните среди, в които се стартират. Ето примерен HTML код, с който се стартира аплета:

```
<html><body><center>
<applet code="SumatorApplet.class"
    codebase="." width="280" height="130">
</applet>
</center></body></html>
```

Забелязваме параметъра codebase=”.”. С него може да се задава пътя до директорията, в която се намира .class файла на аплета. Тагът <applet> има и други параметри, например `ARCHIVE="SomeArchive.jar"`, с който може да се зададе пътя и името на JAR архив, който съдържа класовете на аплета. Ето как изглежда нашият аплет под Netscape Navigator:



В Интернет е пълно със сайтове, които активно използват аплети. Типичен пример за такъв сайт е Web версията на ICQ, достъпна от адрес http://icqlite.icq.com. Липсата на поддръжка за по-високи версии на JDK от 1.1 ни навежда на мисълта, че аплетите са малко остаряла и изоставена технология. Това е вярно в някаква степен и то главно заради отказа на Microsoft да поддържа Java. От няколко години насам Internet Explorer е най-масово използваният браузър и той поддържа само JDK 1.1. До момента, обаче, няма масово навлязла технология която да замести аплетите и затова те си остават единственото решение на много проблеми при разработката на Web-базиран софтуер. Единствената масово навлязла технология в тази насока е Macromedia Flash, която е предназначена главно за мултимедия. Тя, обаче, няма същата мощ, която има Java и поради това не може да замести аплетите.

Повече информация за AWT, разработка на аплети, примери и други материали можете да намерите на сайта на курса: http://inetjava.sourceforge.net.

По идея нашият курс по “Интернет програмиране с Java” има за цел изясняването на три основни технологии: сокет програмиране, аплети и Web-приложения. До момента разгледахме първите две от тях – многонишковото програмиране с TCP, UDP и Multicast сокети и разработката на Java аплети. Предстои ни последната и най-важна част от курса – разработката на Web-приложения.

Web-приложения с JAVA

Web-приложенията представляват информационни системи, които са достъпни през Интернет или локална мрежа чрез стандартен Web-браузър. Например всички Web-базирани системи за електронна поща (като `mail.yahoo.com`, `abv.bg`, `mail.bg` и др.) представляват Web-приложения. За достъп до една Web-базирана система, е необходимо потребителят да разполага със стандартен Web-браузър (например Internet Explorer) и връзка с компютъра, на който се намира тази система. Обикновено връзката се осъществява чрез Интернет, а за достъп до системата се използва адресът на нейния Web-сайт в рамките на глобалната информационна система WWW. Глобалната разпределена информационна система WWW (World Wide Web) представлява съвкупността от

всички сървъри в Интернет, предоставящи достъп до ресурси чрез стандартен Web-браузър. Огромно е разнообразието от технологии, на които тя е изградена. Огромни са и възможностите за взаимодействие между сървърите, които я изграждат. За публикуването на информация в Web пространството (WWW) се използват Web-сървъри. Web-сървърът представлява софтуер, който предоставя достъп до някаква информация чрез протокола HTTP. Web-сървърите могат да предоставят както статични ресурси, така и ресурси, които се създават в момента на заявката за достъп до тях (динамично генерирана информация). На един Web-сървър може да има един или няколко Web-сайта, всеки от които е комбинация между статично и динамично съдържание. Един сайт, разбира се, може да е разположен на няколко Web-сървъра. Едно Web-приложение представлява софтуерна система с Web-базиран потребителски интерфейс, работеща на някакъв Web-сървър в Интернет или в локална мрежа. На един Web-сървър може да има няколко независими Web-приложения, както и статична информация, която не е обвързана с никое от тях. Web-приложенията могат да взаимодействат по между си по най-разнообразни начини. Възможно е резултите от няколко независими Web-приложения, работещи на различни и отдалечени един от друг сървъри да се използват съвместно в една Web-страница. Типичен пример за такова взаимодействие са рекламите по сайтовете в Интернет. Например приложението за електронна поща достъпно от `mail.yahoo.com` показва в един HTML документ резултата от работата на две различни Web-приложения. Едното е приложението за четене на поща, което дава потребителски интерфейс за четене и изпращане на писма, а другото е приложението, което се грижи за рекламите и се изпълнява на съвсем друг сървър. Друг пример за съвместно използване на няколко Web-приложения са броячите на посетитети, които се вграждат в различни сайтове и представляват независими приложения, обикновено работещи на отделни сървъри.

Обикновено един Web-сайт представлява съвкупност от статично съдържание (Web-страници, картинки, документи и др.), динамичнно съдържание (Web-страници и други документи) и Web-приложения. Както статичното, така и динамичното съдържание в един сайт може да е разпределено на различни сървъри. Трудно е да се дефинира точната граница между две Web-приложения, защото и те могат да бъдат разпределени на няколко сървъра и да работят като една цяла система.

Най-често под Web-приложение се разбира цялостна софтуерна система за предоставяне на някаква услуга на потребителя през Web. От гледна точка на програмирането Web-приложенията представляват стандартни клиент-сървър системи. Клиентът, както вече знаем, е стандартният Web-браузър, а сървърът е Web-сървърът, на който работи Web-приложението. Характерна черта за Web-приложенията е, че към тях осъществяват достъп много потребители едновременно. Всеки потребител се обслужва независимо от другите потребители, така сякаш е единствен. Друга характерна черта на Web-приложенията е, че работят с еднопосочна комуникация, на принципа заявка-отговор (request-response). Браузърът на клиента дава заявка за някакъв ресурс и сървърът отговаря на тази заявка с изпращането на поискания ресурс или със съобщение за грешка. Този модел на комуникация лишава сървъра от възможността да изпраща асинхронно данни на клиента по свое желание. Това ограничение сериозно затруднява системите, които осъществявят достъп до информация в реално време.

Web-приложения могат да се разработват на различни езици и за различни Web-платформи – CGI, Perl, PHP, ASP, Java/JSP и др. При разработката на Web-приложения с Java се използват технологиите Java-сървлети и Java Server Pages (JSP) и платформата за Web-приложения на Sun, която е част от J2EE (Java 2 Enterprise Edition). Тази платформа ни дава стандартен framework (съвкупност от програмни средства и стандарти) за разработка на Web-приложения, който ще разгледаме по-късно. Нека започнем с основите концепции на Web-програмирането.

Web-сървър

От гледна точка на Интернет програмирането Web-сървърите са приложения, които “слушат” на определен порт (обикновено това е стандартният порт за HTTP – 80), и отговарят на HTTP заявките, получени от клиентски приложения (най-често това са Web браузърите). Простите Web-сървъри могат само да връщат в отговор на заявките файловете, които са разположени в

директорията, обозначена като главна Web-директория. Например ако имаме един прост Web-сървър стартиран на компютъра с Интернет адрес `www.mywebserver.com` и сме указали, че главната му директория е `C:\MyWebSite`, то когато Web-браузърът поиска ресурса `http://www.mywebserver.com/pictures/index.html`, нашият прост Web-сървър ще му предостави файла `C:\MyWebSite\pictures\index.html` (ако съществува). Всички съвременни Web-сървъри имат възможността да предоставят на клиентите си не само файлове от главната Web-директория и нейните поддиректории, но и динамично генериран HTML, получен от работата на външна за Web-сървъра програма. Тази технология се нарича CGI (Common Gateway Interface). При CGI на базата на HTTP заявката Web-сървърът стартира някоя CGI-програма и връща на клиента това, което тази CGI-програма изпише на стандартния изход като резултат от изпълнението си. CGI-програмата може да бъде написана на практически всеки език или script за програмиране (например на C, C++, Pascal, Perl, PHP и др.). Има и други възможности за динамично генериране на HTML – например не чрез външна програма, а чрез модул вграден директно към Web-сървъра. Такъв подход използва ISAPI технологията на Microsoft, която дава възможност за динамично вграждане в сървъра на компилиран програмен код от DLL файл. Технологията, която лежи в основата на Web-програмирането с Java – JSP/Servlets, също използва вграждане в Web-сървъра на компилиран програмен код (Java класове), който генерира динамично HTML. Вграждането на компилиран програмен код пред извикването на външна програма има известни предимства. По скорост на изпълнение е по-ефективно, защото не се налага за всеки динамично генериран HTML документ да се извиква външно приложение, при което в операционната система се създава нов процес. Интеграцията между сървъра и вградения в него компилиран код е по-лесна отколкото интеграцията с външна CGI-програма.

HTTP протокол

Не е разумно да говорим за Web-програмиране преди да сме изяснили протокола по който си комуникират Web-сървърите и Web-браузърите – HTTP (Hyper Text Transfer Protocol). HTTP представлява прост текстов протокол, който се използва за пренос на практически вякакъв вид данни, наричани събирателно **ресурси**. В HTTP протокола има понятия като клиент (обикновено това са Web-браузърите) и сървър (това са Web-сървърите). Обикновено HTTP протоколът работи през стандартен TCP сокет отворен от клиента към сървъра. Стандартния порт за HTTP протокола е 80, но може да се използва и всеки друг свободен TCP порт. Комуникацията по HTTP се състои от заявка (request) – съобщение от клиента към сървъра и отговор (response) – отговор на сървъра на съобщението от клиента.

HTTP заявки

HTTP заявката има следния формат:

```
<метод> <URI> HTTP/1.1
<header-полета>
<празен ред>
```

Забелязват се 3 основни елемента: **метод**, **URI** и **header-полета**. **Методът** описва вида на HTTP заявката, изпратена от клиента. Най-често използваните методи са GET и POST. Чрез GET клиентът изисква някакъв ресурс от Web сървъра. POST служи за предаване на данни към сървъра. Имената на методите в HTTP заявките се изписват винаги с главни букви. Уникалният идентификатор **URI (Unique Resource Identifier)** еднозначно определя ресурса, над който ще оперира заявката. Това е частта от URL, която стои след името на хост-а (сървъра) в URL-то. Фрагментът **HTTP/1.1** с който завършва първият ред задава версията на HTTP протокола, която ще бъде използвана за осъществяването на HTTP сесията. **Header-полетата** от заглавната част задават допълнителни параметри на заявката и определят различни изисквания относно ресурса, който се очаква да бъде върнат от Web-сървъра. **Празният ред** определя края на заявката. Да дадем един пример за HTTP заявка, която връща началната страница от сайта `http://www.dir.bg/`:

```
GET / HTTP/1.1
Host: www.dir.bg
```

↵

Методи на HTTP заявката

Протоколът HTTP версия 1.1 поддържа общо 8 различни метода: GET, POST, HEAD, PUT, DELETE, OPTIONS, TRACE, CONNECT. Най-често използваните методи са GET и POST и те имат най-голямо отношение към Web-програмирането.

GET методът преставлява команда за извличане на ресурс, указан от зададено URI. Всичко, което прави Web-сървърът за извличането на статичен ресурс чрез GET заявка е да го прочете от файловата система и да го върне на клиентите в подходящ HTTP отговор. При извличане на динамичен ресурс сървърът извиква компилираният програмен код, който генерира ресурса и връща резултата от него в HTTP отговор. Ето един реален пример за HTTP заявка с GET метод:

```
GET /InetJava-2002-program.html HTTP/1.1
Host: inetjava.sourceforge.net
Accept: */*
Accept-Language: bg
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
↵
```

Изпращането на тази заявка към Web-сървъра, който слуша на порт 80 на машината с Интернет адрес `inetjava.sourceforge.net`, ще върне файла `InetJava-2002-program.html`, достъпен от URL `http://intejava.sourceforge.net/InetJava-2002-program.html`. Ако към искания ресурс трябва да се зададат параметри, това става като към URI-то се добави въпросителен знак, а след него двойки от вида `<име на параметър>=<стойност>`, като двойките от този вид се разделят една от друга със `&`. За избягването на някои непозволени символи се използва така нареченото URL-кодиране, за което ще говорим по-късно.

POST методът служи за изпращане на данни от клиента към Web-сървъра. Обикновено сървърът предава получените от POST заявката данни на някакъв CGI скрипт или вграден модул за динамично генериране на HTML, който ги обработва и връща някакви резултати. Тези резултати се връщат на клиента като отговор на неговата заявка. Ето и един реален пример за HTTP заявка с POST метод, изпратена от Internet Explorer 6.0 при опит за влизане в Web-базираната система за електронна поща на `www.abv.bg`:

```
POST /webmail/login.phtml HTTP/1.1
Host: www.abv.bg
Accept: */*
Accept-Language: bg
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0(compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 59
↵
LOGIN_USER=boris
DOMAIN_NAME=abv.bg
LOGIN_PASS=tajnamajna
↵
```

Както се вижда, параметрите се предават след самата заявка, като в header-полетата се указва общата дължина в символи на всички параметри и техните стойности. За разделител между header-полетата и параметрите се използва празен ред. За край на заявката се използва също празен ред. Ако параметрите съдържат непозволени символи, те се кодират по специален начин. Кодирането и разкодирането на параметрите и стойностите им се прави автоматично от Web-браузъра и Web-сървъра. Като Web-програмисти не е необходимо да знаем в детайли точно как става това.

Отговори на HTTP заявки

На всяка HTTP заявка, независимо дали е валидна или не, Web-сървърът връща някакъв отговор. При валидна заявка за съществуващ ресурс Web-сървърът връща този ресурс, а в противен случай връща грешка. Отговорът на HTTP заявка има следния формат:

```
HTTP/1.1 <код> <текст>
<header-полета>
<празен ред>
<ресурс>
```

Първият ред се нарича **статус линия** и съдържа версията на HTTP протокола, по който се изпраща отговора, трицифрен код на резултата или код на грешка и кратко текстово описание на този код. Следват **header-полетата.** Те съдържат различни параметри на върнатия ресурс, както и информация за Web-сървъра. Следва **празен ред**, а след него **ресурса**, кодиран по описания в header-полетета начин. В зависимост от типа на ресурса, сървърът може да го върне кодиран по различен начин и по различен начин да укаже на клиента колко байта е дълъг HTTP отговорът. Стойностите на header-полетата и формата на ресурса са от интерес основно за Web-браузъра и затова няма да ги разглеждаме в детайли. Основното, което трябва да знаем е, че на всяка HTTP заявка сървърът отговаря с HTTP отговор, който съдържа искания ресурс или грешка. Кодовете на грешките започват с цифрата 4 или 5. Кодовете на успешен резултат започват с 2, а кодовете, носещи специална информация – с 3. Най-често срещаните кодове при HTTP отговор са: 200 – успех; 304 – документът не е променян от времето, зададено в header-а (използва се от браузърите за кеширане на документи); 404 – ресурсът не е намерен; 500 – грешка на сървъра. Ето един пример за изпращане на HTTP заявка за извличане на главната страница от локално стартиран Web-сървър и отговорът на тази заявка:

```
C:\> telnet localhost 80
GET / HTTP/1.1
Host: localhost
↵

HTTP/1.1 200 OK
Date: Sat, 10 Aug 2002 16:09:18 GMT
Server: Apache/1.3.9 (Win32)
Accept-Ranges: bytes
Content-Length: 73
Content-Type: text/html
↵
<html>
<head> <title> Test </title> </head>
    Test HTML page.
</html>
```

Както се вижда, сървърът е върнал отговор на HTTP заявката с код 200 (успех) и върнал искания ресурс. Ето и един пример за неуспешно завършила заявка:

```
C:\> telnet localhost 80
GET /img/nakov.gif HTTP/1.0
↵

HTTP/1.1 404 Not Found
Date: Sat, 10 Aug 2002 16:20:17 GMT
Server: Apache/1.3.9 (Win32)
Connection: close
Content-Type: text/html
↵
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /img/nakov.gif
was not found on this server.<P>
<HR><ADDRESS>Apache/1.3.9
Server at test Port 80</ADDRESS>
</BODY></HTML>
```

Вероятно сте забелязали, че в последния пример използваме заявка по протокол HTTP/1.0, а в предходния – по HTTP/1.1. Най-съществената разликата между двете версии на протокола е, че при HTTP/1.0 след връщането на отговора на HTTP заявка сървърът веднага затваря сокета с клиента, а при HTTP/1.1 може с едно отваряне на сокет да се изпълнят последователно няколко HTTP заявки. Това прави HTTP/1.1 протоколът по-бърз заради което е предпочитан от повечето HTTP клиенти.

(следва продължение в следващия брой)