Интернет програмиране с Java – част 4

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

Последна промяна: 19.07.2002

В предходната тема изяснихме как се работи с UDP и Multicast сокети, а също така и средствата които ни дава платформата Java за достъп до URL ресурси. С това приключихме първата част от нашия задочен курс по програмиране на Java за Интернет – работа със сокети. Предстои ни да се запознаем с втората част на курса – разработка на Java аплети.

JAVA аплети

До преди десетина години беше силно разпространено схващането, че езикът Java служи единствено за създаване на аплети. По това време това схващане беше в голяма степен правилно, защото Java първоначално беше планиран и точно за това и използването му за други цели започна по-късно. Развитието на езика през последните 5-6 години, обаче, коренно промени ролята му в света на програмирането и Java стана една от най-популярните платформи за разработка на корпоративни софтуерни системи. Аплетите, въпреки че вече не заемат централна част в Java платформата, все още си остават една интересна насока в Интернет програмирането, която си струва да разгледаме.

Преди да пристъпим към повече детайли, трябва да си изясним какво е Java аплет. Аплетът е компилирана програма на Java, която се вгражда като обект в обикновена Web-страница и се изпълнява от Web-браузъра по време на разглеждането на тази страница. Аплетите се вграждат в Web-страниците по начин много подобен на вграждането на картинки, но за разлика от тях, те не са просто графични изображения, а програми, които използват правоъгълната област, която броузерът им е дал, за графичния си потребителски интерфейс. Аплетите притежават почти цялата мощ която ни дава Java платформата, но с известни ограничения, предизвикани главно от съображения за сигурност. За да се осигури безопасността на потребителя, на аплетите е позволено да извършват само операции, които не могат да осъществят достъп до потребителската информацията на машината, на която се изпълняват. Аплетите представляват компилирана Java програма във вид на .class файл или съвкупност от компилирани Java класове, записани в .jar файл. Както знаем всички Java програми се изпълняват от Java виртуална машина (JVM) и затова браузърите, които поддържат аплети имат вградена в себе си или допълнително инсталирана виртуална машина. При отварянето на HTML документ с аплети, браузерът зарежда Java виртуалната си машина и стартира аплетите в нея.

В повечето случаи Java аплетите наследяват класа `java.applet.Applet` и припокриват методите му за инициализация и за изобразяване на екрана – съответно `init()` и `paint()`. В метода `paint()` аплетът изобразява графично на екрана текущото си състояние използвайки стандартните средства на Java за създаване на графичен потребителски интерфейс – AWT (Abstract Window Toolkit). Тези средства се намират в пакета `java.awt` и ще бъдат разгледани по-късно. Да разгледаме един съвсем прост аплет:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
       g.drawString("Hello world!", 50, 25);
    }
}
```

В този пример единственото, което прави аплетът, е в метода си за изобразяване на екрана да чертае текст в областта, която му е дадена от браузъра на позиция (50, 25) с шрифта по

подразбиране. Създаването на аплета `HelloWorldApplet.java` и компилирането му до `.class` файл не е достатъчно за да може той да се изпълни. За разлика от нормалните Java програми аплетите не е задължително да имат `main()` метод. За да видим резултата от нашия аплет трябва да направим Web-страница, в която да го вмъкнем като обект. Ето един пример как става това:

```
<html><body>
    <p align="center">This is the applet:<br><br>
    <applet code="HelloWorldApplet.class"
        width="150" height="50">
    </applet></p>
</body></html>
```

Използвахме HTML тага `<applet>`, в който зададохме името на класа, който искаме да вмъкнем като обект, както и размерите на областта от Web-страницата, която му се предоставя. Ако запишем този HTML код във файла `index.html` и отворим този файл с Internet Explorer, ще получим резултат подобен на следния:

Друг начин да изпълним аплета ни дава програмата `appletviewer`, която е включена към стандартната инсталация на JDK. С нея също можем да изпълняваме аплети, но за разлика от стандартния браузър, appletviewer дава повече права на аплета и работи с текущата версия на JDK. Като параметър appletviewer приема име на HTML файл, който съдържа аплет. Въпросът с версиите на JDK и съвместимостта оставяме за по-нататък, а сега нека навлезем в повече детайли относно това как се създават аплети.

Класът `java.applet.Applet` е базов клас за всички аплети и е наследник на `java.awt.Panel`. Следователно той представлява AWT контейнер и може да се използва за поставяне на различни AWT компоненти с цел изграждане на потребителски интерфейс. Освен за това, той може да се използва също и за директно рисуване със средствата на AWT, както това е направено в горния пример. Класът `java.applet.Applet` дава базовата функционалност на аплетите и предоставя на наследниците си методи за взаимодействие с браузера и външния свят. Да проследим жизнения цикъл на един аплет – какво се случва от момента на зареждане на HTML-страницата с аплета до момента, в който се затвори браузерът или се премине на друга страница. Първоначално браузерът чете HTML документа и намира `<applet>` таговете. За всеки от тях намира и зарежда клас файла, който е указан, инстанцира го в Java виртуалната си машина и започва да го изпълнява. Изпълнението на аплет се състои в следното: Извиква се `init()` метода. Извиква се `start()` метода. Докато аплетът работи, браузерът му подава всички събития, предназначени за него и го оповестява, когато е необходимо да се пречертае поради някаква причина. Събитията, отнасящи се до аплета, като кликване с мишката, натискане на клавиш и др. се подават на метода `handleEvent()`, който извършва необходимото те да се обработят от AWT контролата, за която са предназначени. Събитията за пречертаване възникват когато се промени видимата част от аплета, например при скролиране на документа или при засичане на аплета с друг прозорец. Обработчикът на тези събития предизвиква извикване на метода `paint()`, при изпълнението на който аплетът е длъжен да се пречертае. При приключване на изпълнението на аплета, браузерът

извиква последователно методите `stop()` и `destroy()`. Методът `init()` се извиква еднократно след като аплетът е инстанциран, т.е. е създаден като обект във виртуалната машина на браузъра. В него аплетът може да създаде контролите от потребителския си интерфейс, да направи някои инициализации и всичко останало, което трябва да се изпълни еднократно, преди аплетът да е стартиран. Методът `start()` се вика след инициализацията и след рестартиране на аплета. За разлика от `init()`, методът `start()` може да се извика повече от веднъж. Методът `stop()` се извиква, когато браузерът напуска страницата, в която е зареден аплета. След `stop()` всички нишки на аплета минават в състояние на пауза. При връщане обратно в страницата с аплета, се вика `start()`. Трябва да отбележим че повечето браузъри реализират по различен `start()` и `stop()` методите и затова използването им трябва да става внимателно, а при възможност да се избягва. Методът `destroy()` се извиква еднократно преди аплетът да се унищожи от браузера.

AWT представлява платформено-независима библиотека от класове, която позволява създаване на графичен потребителски интерфейс с Java и дава цялостен компонентно-ориентиран framework за създаване на приложения, които използват графика и взаимодействат активно с потребителя. Този framework ни предоставя стандартен начин за работа с графични компоненти, като бутони, текстови полета, картинки, текст и т.н., а също и механизъм за обработка на събитията възникнали в резултат от взаимодействието между потребителя и програмата, като например щракване с мишката, вход от клавиатурата и т.н. Библиотеката AWT е толкова голяма, че за подробното й описание е необходимо много повече пространство от това, с което разполагаме. Затова без да претендираме за пълнота ще направим само едно кратко частично въведение в AWT и то главно за да изясним това, което ни е необходимо за да пишем аплети. Подробно описание на библиотеката може да се намери на сайта на курса "Интернет програмиране с Java" http://inetjava.sourceforge.net, а също и в документацията на JDK.

Координатната система на аплет с размери sizeX и sizeY започва от позиция (0,0), която отговаря на горния му ляв ъгъл и завършва в позиция (sizeX-1, sizeY-1), която отговаря на долния му десен ъгъл. Изобразяването на графични обекти става чрез класа `java.awt.Graphics`, обект от който се подава на `paint()` метода на аплета. Всеки `Graphics` обект има сбоя собствена координатна система и всеки AWT графичен компонент има свой собствен `Graphics` обект, чрез който реализира визуализацията си. Класът `Graphics` ни дава методи за чертане на основните графични обекти, като линии, правоъгълници, елипси, запълнени многоъгълници, текст с различни шрифтове и много други. Описание на методите, с които се чертаят тези обекти (`drawLine()`, `drawRect()`, `fillRect()`, `clearRect()`, `drawOval()`, `fillOval()`, `drawArc()`, `fillArc()`, `drawPolygon()`, `fillPolygon()` и т.н.) може да се намери в документацията. Освен директното чертане на геометрични фигури, AWT позволява и изобразяване на картинки, заредени от GIF или JPEG файл. За целта се използва класа `java.awt.Image` и метода на класа `Graphics drawImage()`, който има няколко варианта с различни входни параметри. Най-лесният начин за зареждане на картинка в Image обект се дава от метода `getImage()` на класа `Applet`, който приема URL като параметър. Ето един пример как можем да заредим картинка:

```
URL imageURL = new URL(
    "http://www.nakov.com/images/dot.jpg");
java.awt.Image img = this.getImage(imageURL);
```

За да начертаем върху аплета заредената картинка можем да използване следния код:

```
public void paint(Graphics g) {
    g.drawImage(img, 20, 10, this);
}
```

Ако искаме да начертаем картинката с променени размери, можем да използваме отново метода drawImage(), но с други параметри:

```
g.drawImage(img, 0, 0, img.getWidth(this)/4,
    img.getHeight(this)/4, this);
```

Внимателният читател вероятно е забелязал, че методът `drawImage()` има един параметър, за който в нашите примери даваме стойност `this`. Това съвсем не е случайно и се обяснява с архитектурата на AWT и начина, по който се работи с картинки. Методът `drawImage()` приема като последен параметър обект, който реализира интерфейса `ImageObserver`. Зареждането на картинка в AWT винаги е асинхронно, т.е. извършва се паралелно с работата на програмата. Това е съвсем

обосновано, като се има предвид, че зареждането на картинка от Интернет отнема известно време, а програмата може да го използва за други цели, вместо да чака. По идея `drawImage()` не изчаква картинката да бъде заредена и тогава да я начертае, а чертае само тази част от нея, която вече е заредена и веднага връща управлението на извикващия метод. Когато картинката се зареди напълно, се извиква методът `imageUpdate()` на интерфейса `ImageObserver`, който трябва да обработи ситуацията по подходящ начин. Обикновено реализацията на метода `imageUpdate()` пречертава картинката. Класът `java.awt.Component`, който е прародител на класа `java.applet.Applet` реализира интерфейсът `ImageObserver` и в метода си `imageUpdate()` пречертава областта от екрана, която е обхваната от картинката, която се е заредила напълно. Използвайки тази базова функционалност на класа `Applet`, можем винаги, когато зареждаме картинки от аплет, да подаваме за `ImageObserver` самия аплет, т.е. обекта `this`. Да разгледаме един цялостен пример за аплет, който използва картинки и реализира проста анимация. Да си поставим за задача направата на аплет, в който една топка постоянно се движи и се отблъсква в стените на аплета при удар. Едно възможно решение на задачата е следното:

```
import java.awt.*;
import java.applet.*;
import java.net.URL;

public class BallApplet extends Applet
implements Runnable {

    public static final int SPEED = 20;
    Image ballImg;
    int x, y, px, py;
    Thread animateThread = null;
    Image bufImg;
    Graphics bufGr;

    public void init() {
        try {
            String imgName =
                getParameter("imgName");
            ballImg = getImage(
                getCodeBase(), imgName);
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
        x = 1; y = 1; px = 1; py = 1;
        bufImg = createImage(
            getSize().width, getSize().height);
        bufGr = bufImg.getGraphics();
    }

    public void paint(Graphics g) {
        if (animateThread != null) {
            // Paint in the buffer
            bufGr.fillRect(0, 0,
                getSize().width, getSize().height);
            bufGr.drawImage(ballImg, x, y, this);
            // Move the buffer to the screen
            g.drawImage(bufImg, 0, 0, this);
        }
    }

    public void start() {
        if (animateThread == null) {
            animateThread = new Thread(this);
            animateThread.start();
        }
    }

    public void stop() {
        animateThread = null;
    }

    public void run() {
        // Wait for the image to load completely
```

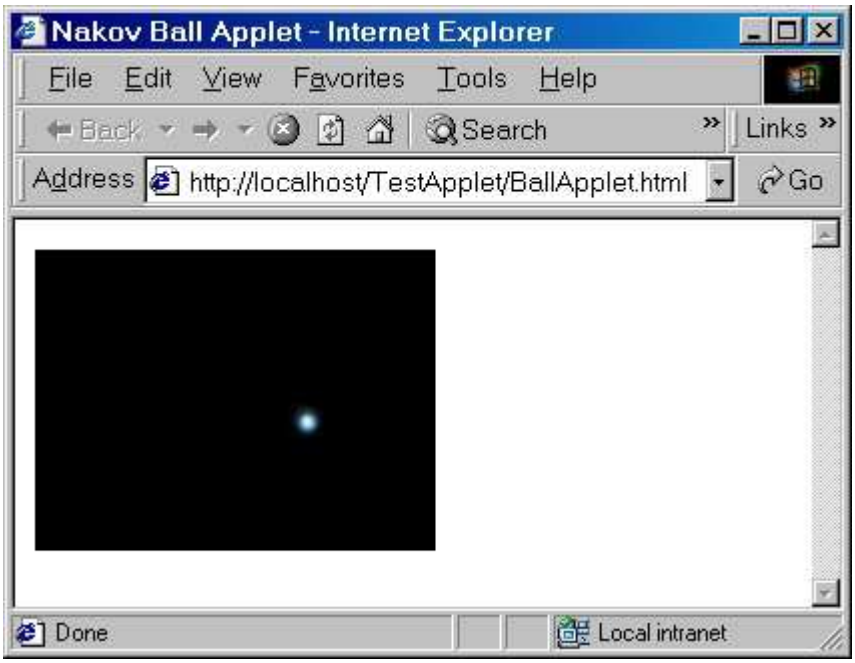
```
            MediaTracker tracker =
                new MediaTracker(this);
            tracker.addImage(ballImg,0);
            try {
                tracker.waitForAll();
            } catch (Exception ex) {}
            // Calculate the animation area size
            int maxX = this.getSize().width -
                ballImg.getWidth(this);
            int maxY = this.getSize().height -
                ballImg.getHeight(this);
            // Animate until interrupt is requested
            while (animateThread != null) {
                if ((x > maxX) || (x < 0))
                    px = -px;
                x = x + px;
                if ((y > maxY) || (y < 0))
                    py = -py;
                y = y + py;
                try {
                    Thread.sleep(SPEED);
                } catch (Exception ex) {}
                // Redraw the applet contents
                paint(getGraphics());
            }
        }
    }
```

Идеята за работата на аплета е следната: При инициализация аплетът зарежда картинката с топката и създава един обект `Image`, който ще използва за буфер. При стартиране на аплета се създава една нишка, която се грижи за анимацията. Нейната роля е да променя координатите `x` и `y` на топката съгласно текущата посока на движение, да сменя посоката при удар в стена и след всяка промяна на координатите на топката да пречертава аплета. Пречертаването на аплета работи със специален буфер. Този буфер се използва за да се избегне премигването и да се получи наистина плавно движение. Вси всяко пречертаване на аплета буферът се изчиства с `fillRect()`, след това в него се начертава топката на текущата й позиция и след това на екрана се изобразява съдържанието на този буфер. Тази техника за избягване на премигването при създаване на анимация се нарича двойно буфериране (double buffering). За реализацията й се създава обект от класа `Image` – `bufImg` и се работи чрез неговия `Graphics` обект – `bufGr`. Вместо да се рисува директно върху аплета, се рисува в буфера и готовия кадър от буфера се прехвърля в аплета. При извикване на `stop()` метода аплетът спира нишката за анимация, а при `start()` я създава и я стартира. Името на файла, който съдържа картинката се задава като параметър на аплета и се взема с метода `getParameter()`. Параметрите на аплетите служат за задаване на различни настройки без да е необходима прекомпилация, което се налага ако тези настройки са зададени като константи. За задаването им има специален таг, който се влага в така `<applet>` – тага `<param>`. Ето един примерен HTML код, който стартира нашия аплет и задава за параметъра име на картинка `imgName` стойността `ball.jpg`:

```
<html>
<head><title>Nakov Ball Applet</title></head>
<body>
<applet code="BallApplet.class" width="200"
 height="150">
    <param name="imgName" value="ball.jpg">
</applet>
</body>
</html>
```

В инициализационната си част аплетът взима параметъра imgName и зарежда картинката с това име от директорията, от която е зареден аплетът. URL, сочещо към тази директория може да се получи чрез метода `getCodeBase()`. Такъв е правилният начин за извличане на ресурс от аплет – не чрез абсолютен URL, а чрез релативен URL спрямо директорията, от която е зареден аплетът. След като е извикан методът за зареждане на картинка, тя е започнала да се извлича от зададения URL, а през това време аплетът създава картинка-буфер за целите на анимацията. При извикване на `start()` метода аплетът създава нишката за анимация и я стартира. При стартиране нишката първо изчаква картинката с топката `ballImg` да се зареди напълно. За целта се използва класа

`MediaTracker`, чрез който може да се проследи състоянието на започнали да се зареждат картинки. Да си припомним че работата с картинки в AWT е асинхронна! След като картинката е напълно заредена, можем да сме сигурни, че ако се опитаме да й вземем размерите, операцията ще е успешна. Тези размери, както и размерите на аплета използваме за изчисляване на полето, в което топката може да се движи, така че да не излиза от аплета. Това е правоъгълната област (0, 0) – (maxX, maxY). Самата анимация представлява един цикъл, в който се изчисляват новите координати на топката и аплетът се пречертава с извикване на `paint(getGraphics())`. Между всеки два кадъра от анимацията се изчаква някакво време `SPEED` с цел топката да не се движи прекалено бързо и да се движи с приблизително еднаква скорост на различни компютри. При спиране на аплета с метода `stop()` на променливата `animateThread`, която сочи към нишката за анимация се присвоява `null` и това е знак за `run()` метода на тази нишка, че трябва да приключи работа. Затова в цикъла, в който се движи топката, стойността на `animateThread` се следи непрекъснато. Такъв е препоръчителният начин за прекъсване на работата на нишка. Ето и разултатът от нашия аплет, видян през Internet Explorer:



За да изглежда добре е необходимо картинката, която представлява топката (`ball.jpg`) да е по-малка от размерите на аплета и да бъде на черен фон.

При разработката на аплети и при тестването им с Internet Explorer или друг Web-браузър, трябва да имаме предвид някои неща. Повечето браузъри използват кеширане на различни обекти от Web-страниците, например картинки, стилове и др. за постигане на по-голяма скорост при зареждане на HTML документ. Кеширането се използва и за аплетите, поради което трябва да се внимава. Трябва да се знае, че записването на нова версия на аплета и натискане на бутона "Refresh" не гарантира изпълнението на новата верия. При Internet Explorer сигурен начин за зареждането на последната версия на аплета е натискането на Ctrl+F5 или Ctrl+"Refresh", но при други браузъри клавишните комбинации са други. Най-сигурно е затваряне на браузъра и стартиране отново. Само така можем да сме сигурни, че последната промяна в кода се е отразила на аплета, и то при условие, че не сме забравили да прекомпилираме.

Друга важна особеност на браузърите е че повечето поддържат много стари версии на JDK. Например Internet Explorer 4.0, 5.0 и 5.5 поддържат само JDK 1.1. Повечето версии на Netscape Navigator също. JDK 1.3.1 се поддържа след издърпване на специален plig-in от сайта на Sun. Поради лошите отношения между Microsoft и Sun, Inetrnet Explorer 6.0 вече не поддържа стандартно аплети, а само след допълнително инсталирване на Java plug-in. Не е ясно защо, но Netscape 6.0 също не подържа стандартно Java аплети и се нуждае от plug-in. Поради изтъкнатите проблеми със съвместимостта когато пишем аплети трябва да използваме JDK 1.1. В противен случай рискът аплетът да не работи на машината на клиента не е малък. Повечето класове и методи, които биха ни потрябвали при писане на аплети ги има в JDK 1.1, така че използването на

тази версия не ни ограничава сериозно. Трябва да се съобразяваме, че имената на някои методи в различните версии на JDK са различни, но обикновено има съответствие и съвместимост отдолу нагоре. Например в `java.awt.Component` от JDK 1.2 е въведен метод `getWidth()`, а в JDK 1.1 вземането на широчината на компонент става с `getSize().width`. При използването на липсващ метод на някой клас, виртуалната машина на браузера предизвиква изключение. Изключенията, които възникват в аплетите, както и всичко, отпечатано чрез `System.out.println()` можем да видим в Java конзолата на браузъра. В повечето браузъри тя е достъпна от менюто. В Internet Explorer Java конзолата се появява в менюто "View" само след като се разреши от опциите (Internet Options | Advanced | Microsoft VM | Java console enabled) и се рестартира браузерът. Намирането на проблем в аплет без Java конзолата е немислимо, така че когато разработвате аплети и нещо не работи, винаги поглеждайте в конзолата. Типичен е случаят, в който в средата за разработка (например Jbuilder или IDEA) аплетите работят, а в браузера не искат. Причините са две – разлика във версиите на JDK и ограничените права, които се дават на аплетите. Обикновено `appletviewer` или средата за разработка стартират аплетите с повече права, отколкото един браузър би им дал и с JDK, различно от 1.1 и затова се получава несъвместимост. Преди да изясним ситуацията с правата на аплетите, трябва да отбележим, че в новите версии на JDK съществува разширение на библиотеката AWT, което се нарича Swing и се намира в пакета `javax.swing`. Използването на Swing в аплети не се препоръчва, защото много малко браузъри поддържат Swing стандартно.

(следва продължение в следващия брой)