Программирование на языке Java. Работа с изображениями

Картузов А.В.

Java работает с наиболее популярными во Всемирной паутине форматами изображений—JPEG и GIF. JPEG лучше подходит для естественных цветных изображений, таких, как фотографии, а формат GIF является наилучшими для графических эмблем, изображений кнопок, и т.п.

Сначала мы загрузим изображение с помощью очень короткой программы. Затем мы научимся использовать классы, которые управляют загрузкой одного или нескольких изображений. Кроме того, существует набор абстрактных классов, которые помогают создать поток изображений, и фильтры, позволяющие обращаться к отдельным элементам изображений и модифицировать их.

1. Простой загрузчик изображений

Простейший случай—загрузка в страницу одиночного изображения. Вот маленький апплет, выполняющий эту работу:

```
/* <title>SimpleImageLoad</title>
*  <applet code="SimpleImageLoad" width=300 height=150>
*     <param name="img" value="mupk.gif">
*  </applet>
*/
import java.applet.*;
import java.awt.*;
        public class SimpleImageLoad extends Applet {
                Image art;
                public void init() {
                        art = getImage(getDocumentBase(),
```

```
                        getParameter("img"));
                        }
                public void paint(Graphics g) {
                        g.drawImage(art, 0, 0, this);
                        }
                }
```

Метод paint использует drawlmage с четырьмя аргументами: это ссылка на изображение art, координаты левого верхнего угла рисунка x, y и объект типа ImageObserver. Мы поговорим подробнее об ImageObserver в следующем параграфе; здесь мы использовали this в качестве имени ImageObserver, поскольку он встроен в апплет. Когда этот апплет запускается, он в методе init начинает загрузку art. Процесс загрузки изображения по сети хорошо заметен—SimpleImageLoad.html, поскольку встроенный интерфейс ImageObserver вызывает процедуру paint при каждом поступлении новой порции данных из сети. Вы можете использовать ImageObserver для отслеживания загрузки изображения, а в это время выводить на экран другую информацию.

2. ImageObserver

ImageObserver—это абстрактный интерфейс, используемый для получения сообщения о создании изображения, Метод imageUpdate из ImageObserver—это все, что вы должны реализовать в своем апплете для его использования. В то время, как вы получаете информацию о загрузке, вы можете показывать любую понравившуюся вам мультипликацию, индикатор степени завершения загрузки или любую другую заставку. Для использования ImageObserver в своем подклассе Applet вы должны добавить в него строку implement ImageObserver, как показано в этом фрагменте программы:

```
        public class MyApplet extends Applet implement ImageObserver {
```

Затем вам придется вставить в свой класс метод imageUpdate для интерфейса ImageObserver, как показано в следующем фрагменте :

```
public boolean imageUpdate(Image img, int status,
        int x, int y int width, int height) {
                if((status & ALLBITS) != 1) {
```

```
                        System.out.println("Still processing the image");
                        return true;
                        }
                else {
                        System.out.println("Done processing the image");
                        return false;
                        }
                }
```

Метод imageUpdate вызывается с изображением Image, которое находится в процессе изменения, целым параметром status, отражающим состояние изменения, и с координатами прямоугольника (x, y, width, height), которые соответствуют различным величинам в зависимости от информационных флагов, перечисленных ниже. ImageUpdate должен возвращать false по окончании загрузки изображения и true—если изображение еще обрабатывается.

Целая переменная status поразрядно проверяется на наличие одного или нескольких флагов. Возможные флаги и информация, которую они несут, перечислены ниже:

WIDTH	Ширина изображения доступна и может быть взята из аргумента width.
HEIGHT	Высота изображения доступна и может быть взята из аргумента height.
PROPERTIES	Свойства изображения теперь доступны. Вы можете получить их посредством art.properties.
SOMEBITS	Доступны пиксели, необходимые для рисования масштабированного варианта изображения. Область, содержащая новые пиксели, задается параметрами x, y, width и height.
FRAMEBITS	Еще один кадр ранее нарисованного изображения с несколькими кадрами, готов для перерисовки. Параметры x, y, width, height не содержат информации.
ALLBITS	Обработка перерисовываемого изображения окончена, и оно может быть отрисовано в конечном виде. Значения аргументов x, y, width и height не содержат значимой информации.

ERROR	При пересылке изображения возникла ошибка. Поступление дальнейшей информации стало невозможным и рисование прервано. Для удобства выставляется и флаг ABORT для индикации прерывания загрузки изображения.
ABORT	Пересылка изображения была прервана до полного его получения. Поступление новой информации стало невозможным без дополнительных действий по повторному запуску операций по получению изображения. Если флаг ERROR не был выставлен, то приход любых данных изображения снова запустит процесс его получения.

Теперь давайте рассмотрим программный пример, который использует ImageObserver для показа количества обработанных строк изображения и выводит эту информацию (переменная progress) на консоль:

```
/*  <title>ObservedImageLoad</title>
*   <applet code="ObservedImageLoad" width=290 height=140>
*     <param name="img" value="mupk.gif">
*   </applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
        public class ObservedImageLoad extends Applet
        implements Runnable, ImageObserver {
                Image art;
                Dimension d;
                int progress;
                Thread motor;
                boolean loaded;
                public void init() {
                        art = getImage(getDocumentBase(),
                        getParameter("img"));
                        loaded = false;
                        progress = 0;
                        }
```

```
public void paint(Graphics g) {
        d = this.getSize();
        loaded = g.drawImage(art, 0, 0, this);
        }
public boolean imageUpdate(Image img, int info,
        int x, int y, int width, int height) {
        if((info & ALLBITS) != 1) {
                if(progress<d.height) {
                        progress = progress+height;
                        }
               System.out.println(progress+"/"+d.height);
                return true;
                }
        else {
                return false;
                }}
public void start() {
        motor = new Thread(this);
        motor. start();
        }
public void stop() {
        motor.stop();
        }
public void run() {
        motor.setPriority(Thread.MIN_PRIORITY);
        while(!loaded) {
        // update progress indicator (5 fps)
                repaint();
                try {
                        motor.sleep(200);
                        }
                catch(InterruptedException e) {}
                }
        }
}
```

Метод imageUpdate обрабатывает статус загрузки изображения. Информация о статусе передается через переменную info, с которой сравнивается статическая переменная ALLBITS. Если еще не получено все изображение, то мы добавляем величину heihgt к общему числу обработанных строк изображения. Для проверки этой концепции мы

выводим количество обработанных строк изображения на консоль. Метод run перерисовывает апплет пять раз в секунду (каждые 200 миллисекунд) до тех пор, пока изображение art не загрузится. То, как долго монитор статуса загрузки будет работать, зависит от скорости передачи данных изображения по сети—ObservedImageLoad.html.

3. MediaTracker

MediaTracker—это класс, предоставляющий удобный интерфейс для контроля статуса нескольких изображений. В следующих версиях этот класс будет контролировать другие мультимедийные форматы, такие, как звуковые файлы. Для использования MediaTracker нужно создать новый объект этого класса и использовать метод addImage для контроля статуса загрузки. Используйте MediaTracker при загрузке группы изображений. Пока все изображения, которые вас интересуют, не загружены, пользователя будет развлекать демонстрационный экран.

4. ImageProducer

ImageProducer—это абстрактный интерфейс для объектов, которые готовят данные для Image. Объект, который реализует интерфейс ImageProducer, должен предоставлять массивы целых или байтовых переменных, представляющих собой данные изображений. Давайте познакомимся с очень полезным классом MemoryImageSource, реализующий ImageProducer. Мы создадим новый объект Image из данных, которые сгенерировал ImageProducer.

5. MemoryImageSource

MemoryImageSource—класс, используемый для создания нового изображения из массива пикселей. Вот конструктор, используемый для создания объекта MemoryImageSource:

MemoryImageSource(int width, int height, int pixel[], int offset, int scanLineWidth)

Объект MemoryImageSource собирается из массива целых величин pixel[] в используемой по умолчанию модели цветов RGB для генерации данных объекта Image. В используемой по умолчанию цветовой модели пиксель—это целая величина

состоящая из Alpha, Red, Green и Blue (OxAARRGGBB). Величина Alpha обозначает степень прозрачности элемента изображения.

MemoryImageSource возвращает объект ImageProducer, который используется с createImage для получения изображения, пригодного к использованию. Приведенный ниже короткий пример создает MemoryImageSource, используя вариант простого алгоритма (побитовое исключающее ИЛИ значений x и y координат каждого элемента изображения) из книги Gerard J.Holzmann "Beyond Photography, The Digital Darkroom".

```
/*  <title>Memory Image Generator</title>
*   <applet code="MemoryImager" width=256 height=256>
*   </applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
public class MemoryImager extends Applet {
        Image art;
        Dimension d;
        public void init() {
                generateImage();
                }
        public void generateImage() {
                int pixels[] = new int[d.width * d.height];
                int i = 0;
                int r, g, b;
                for(int y=0; y<h; y++) {
                        for(int x=0; x<h; x++) {
                                r = (x^y)&0xff; // red is x XOR y
                                g = (x*2^y*2)&0xff; //green is 2x XOR 2y
                                b = (x*4^y*4)&0xff; // blue is 4x XOR 4y
                                pixels[i++]=(255<<24)|(r<<16)|(g<<8)|b;
                                }
                        }
                art = createImage
               (new MemoryImageSource(d.width,d.height,pixels,0,d.width));
                }
        public void paint(Graphics g) {
                g.drawlmage(art, 0, 0, this);
```

```
                }
        }
```

Посмотрите как это интересное изображение выглядит на экране—MemoryImager.html.

6. ImageFilter и ImageFilterSource

Подклассы классов ImageFilter и ImageFilterSource используются совместно для создания новых изображений фильтрованием уже существующих. С двумя такими подклассами из пакета java.awt.image вы сейчас познакомитесь.

6.1. CropImageFilter

CropImageFilter создает новое изображение из фрагмента существующего. Использование этого фильтра полезно тогда, когда вы хотите использовать несколько маленьких изображений в одном апплете. Загрузка по сети двадцати изображений по 2 Кбайта происходит намного медленнее, чем загрузка одного файла размером 40 Кбайт. Если ваши изображения—одинакового размера, вы можете собрать их в единый блок и использовать CropImageFilter для разделения блока на отдельные изображения в Java-клиенте.

6.2. RGBImageFilter

RGBImageFilter используется для получения данных о каждом пикселе изображения, которые мы можем модифицировать, и таким образом модифицировать изображение.

7. Мультимедиа-горизонты

Существующая система обработки изображений в Java пока не полностью поддерживает потребительские стандарты из-за ограниченной переносимости в сегодняшнем многообразии компьютерных платформ. Но в Java нет никаких "врожденных" ограничений на разработку мультимедийных приложений. Я уверен, что мы станем свидетелями больших успехов в развитии и совершенствовании этой технологии в течении ближайших лет.