J2EE

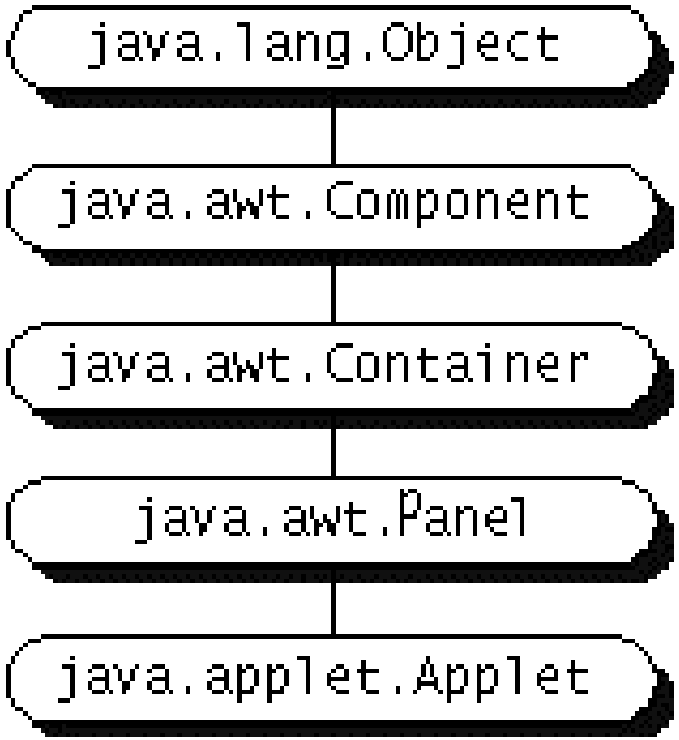
Applets

Servlet - Applet communication

Presented by Bartosz Sakowicz

Overview of Applets

Every applet is implemented by creating a subclass of the Applet class(AWT) or JApplet (Swing). The following figure shows the inheritance hierarchy of the Applet class.



Simple Applet

public class Simple extends Applet { . . .
public void init() { . . . }
public void start() { . . . }
public void stop() { . . . }
public void destroy() { . . . }
. . . }

init - To *initialize* the applet each time it's loaded (or reloaded).
start - To *start* the applet's execution, such as when the applet's loaded or when the user revisits a page that contains the applet.
stop - To *stop* the applet's execution, such as when the user leaves the applet's page or quits the browser.
destroy - To perform a *final cleanup* in preparation for unloading.

Trusted applets

JDK 1.0 assumed all applets were untrusted and ran them under the watch of a SecurityManager that severely limited what they could do.
JDK 1.1 introduced the concept of trusted applets--applets that can operate like normal applications with full access to the client machine. For an applet to be trusted, it has to be digitally signed by a person or company the client trusts
JDK 1.2 is introducing a fine-grained access control system. Under this new system, a digitally signed applet can be partially trusted, given certain abilities without being given free reign on the system.

All following transparencies are presented for <u>untrusted</u> applets in any context of security considerations.

Presented by Bartosz Sakowicz

DMCS TUL

Security restrictions

Current browsers impose the following restrictions on applet that is loaded over the network:

- An applet cannot load libraries or define native methods.
- It cannot ordinarily read or write files on the host that's executing it.
- It cannot make network connections except to the host that it came from.
- It cannot start any program on the host that's executing it.
- It cannot read certain system properties.
- Windows that an applet brings up look different than windows that an application brings up.

Each browser has a **SecurityManager object** that implements its security policies. When a SecurityManager detects a violation, it throws a **SecurityException**. Your applet can catch this SecurityException and react appropriately.

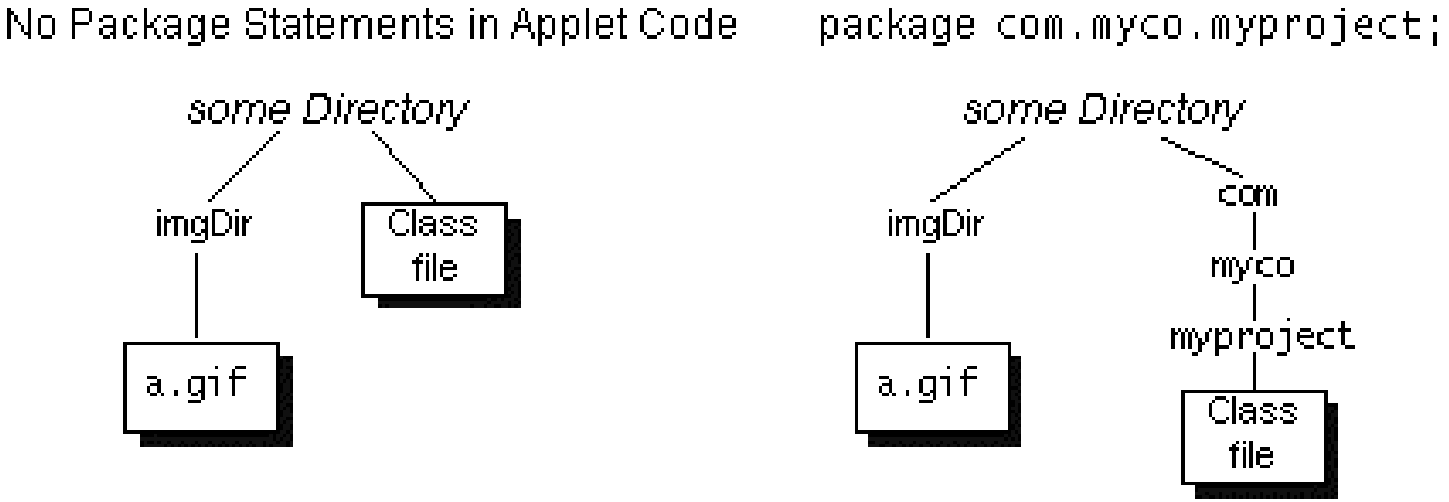
Finding and loading data files

Whenever an applet needs to load some data from a file that's specified with a relative URL, the applet usually uses either the code base or the document base to form the complete URL.

The code base, returned by the Applet ***getCodeBase*** method, is a URL that specifies the directory from which the applet's classes were loaded.
The document base, returned by the Applet ***getDocumentBase*** method, specifies the directory of the HTML page that contains the applet.
Unless the <APPLET> tag specifies a code base, both the code base and document base refer to the same directory on the same server.

Finding and loading data files(2)

The Applet class defines convenient forms of image-loading and sound-loading methods that let you specify images and sounds relative to a base URL. Example:



To create an Image object using the a.gif image file under imgDir, the applet can use the following code:

Image image = getImage(getCodeBase(), "imgDir/a.gif");

Displaying short status string

All applet viewers - from the Applet Viewer to Java-compatible browsers - allow applets to display a short status string.

In current implementations, this string appears on the status line at the bottom of the applet viewer window.

In browsers, all applets on the page, as well as the browser itself, generally share the same status line.

Applets display status lines with the showStatus method. **Example:**
showStatus("MyApplet: Loading image file " + file);

Displaying documents in the browser

With the AppletContext showDocument methods, an applet can tell the browser which URL to show and in which browser window. (the JDK Applet Viewer ignores these methods, since it can't display documents.) Here are the two forms of showDocument:

public void showDocument(java.net.URL url)
public void showDocument(java.net.URL url,
String targetWindow)

TargetWindow can be one of the following:
"_blank"
"*windowName*"
"_self"
"_parent"
"_top"

Sending messages to other applets

Applets can find other applets and send messages to them, with the following security restrictions:

- Many browsers require that the applets originate from the same server.
- Many browsers further require that the applets originate from the same directory on the server (the same code base).
- The Java API requires that the applets be running on the same page, in the same browser window.

Finding an applet by name

You can specify an applet's name in two ways:

• By specifying a NAME attribute within the applet's <APPLET> tag. **Example:**

<APPLET CODEBASE=example/ CODE=Sender.class WIDTH=450 HEIGHT=200 ***NAME="one"****> . . . </APPLET>*

• By specifying a NAME parameter with a <PARAM> tag. **Example:**

<APPLET CODEBASE=example/ CODE=Receiver.class WIDTH=450 HEIGHT=35>
<PARAM NAME="name" value="two"> *. . . </APPLET>*

Finding an applet by name(2)

The ***getApplet*** method looks through all of the applets on the current page to see if one of them has the specified name. If so, getApplet returns the applet's Applet object.

The ***getApplets*** method returns an Enumeration of all the applets on the page.

```
public void printApplets() {
        Enumeration e = getAppletContext().getApplets();
        while (e.hasMoreElements()) {
                Applet applet = (Applet)e.nextElement();
                String info = ((Applet)applet).getAppletInfo();
                // do Sth with Applet
        }
}
```

Playing sounds

Currently, the Java API supports only one sound format: 8 bit, μlaw, 8000 Hz, one-channel, Sun ".au" files.

Sound - related methods:

getAudioClip(URL), getAudioClip(URL, String)

Return an object that implements the AudioClip interface.

play(URL), play(URL, String)

Play the AudioClip corresponding to the specified URL.

The AudioClip interface defines the following methods:

loop

Starts playing the clip repeatedly.

play

Plays the clip once.

stop

Stops the clip. Works with both looping and one-time sounds.

Playing sounds(2)

Example:

```
AudioClip onceClip, loopClip;
onceClip = applet.getAudioClip(getCodeBase(), "bark.au");
loopClip = applet.getAudioClip(getCodeBase(), "train.au");
onceClip.play();        //Play it once.
loopClip.loop();        //Start the sound loop.
loopClip.stop();       //Stop the sound loop.
```

Applet parameters

Parameter values are all strings. Whether or not the user puts quotation marks around a parameter value, that value is passed to your applet as a string.
Applet can interpret the string in many ways. Applets typically interpret a parameter value as one of the following types:

- A URL
- An integer
- A floating-point number
- A boolean value -- typically "true"/"false" or "yes"/"no"
- A string -- for example, the string to use as a window title
- A list of any of the above

Applets should attempt to provide useful default values for each parameter, so that the applet will execute even if the user doesn't specify a parameter or specifies it incorrectly.

Applet parameters(2)

Applets use the Applet ***getParameter*** method to get user-specified values for applet parameters:

public String getParameter(String name)

Example:

```
int requestedWidth = 0;
String windowWidthString = getParameter("WINDOWWIDTH");
if (windowWidthString != null) {
    try { requestedWidth =
    Integer.parseInt(windowWidthString);
    } catch (NumberFormatException e) { } }
```

Besides using the *getParameter* method to get values of applet-specific parameters, you can also use getParameter to get the values of attributes of the applet's <*APPLET*> tag.

Debugging applets

Where exactly the standard output and error are displayed varies, depending on how the applet's viewer is implemented, what platform it's running on, and (sometimes) how you launch the browser or applet viewer:

• When you launch the Applet Viewer from a UNIX shell window strings displayed to the standard output and error appear in that shell window, unless you redirect the output.

• When you launch the Applet Viewer from an X Windows menu, the standard output and error go to the console window.

• Netscape Navigator always displays applet standard output and error to the Java Console.

Debugging applets(2)

Applets display to the standard output or error stream using System.out or System.err. **Example**:

boolean DEBUG = true;
if (DEBUG) { System.out.println("Called someMethod(" + x + "," + y + ")"); }

Note: Displaying to the standard output and error streams is relatively slow. If you have a timing-related problem, printing messages to either of these streams might not be helpful.

You should be sure to disable all debugging output before you release your applet.

Getting system properties

Applets can read the following system properties:

Key	Meaning
"file.separator"	File separator (for example, "/")
"java.class.version"	Java class version number
"java.vendor"	Java vendor-specific string
"java.vendor.url"	Java vendor URL
"java.version"	Java version number
"line.separator"	Line separator
"os.arch"	Operating system architecture
"os.name"	Operating system name
"path.separator"	Path separator (for example, ":")

Others can't.

Threads in applets

Every applet can run in multiple threads.

If an applet performs some time-consuming initialization -- loading images, for example -- in its `init` method this might mean that the browser can't display the applet or anything after it until the applet has finished initializing itself.

The thread that invokes `init` can not do anything else until `init` returns. So if the applet is at the top of the page, for example, then nothing would appear on the page until the applet has finished initializing itself.

If an applet performs a time-consuming task, it should create and use its own thread to perform that task.

Presented by Bartosz Sakowicz

DMCS TUL

Using threads to perform repeated task

Applets typically create threads for repetitive tasks in the applet **start** method.

Creating the thread there makes it easy for the applet to stop the thread when the user leaves the page by using **stop** method.

When the user returns to the applet's page, the start method is called again, and the applet can again create a thread to perform the repetitive task.

Presented by Bartosz Sakowicz

DMCS TUL

Using threads ...- example

```
public void start() {
        if (frozen) {
                //Do nothing. The user has requested that we
                //stop changing the image.
        } else {
                //Start animating!
                if (animatorThread == null) {
                        animatorThread = new Thread(this);
                }
                animatorThread.start();
         }
}
public void stop() {
        animatorThread = null;
 }
```

verte-->

Using threads ...- example(2)

```
public void run() {
        while (Thread.currentThread() == animatorThread){
                //Display a frame of animation and then sleep.
        }
}
```

If animatorThread refers to the same thread as the currently executing thread, the thread continues executing.

If, on the other hand, animatorThread is null, the thread exits.

Using threads to perform one time initialization

Anything that requires making a network connection should generally be done in a background thread. GIF and JPEG image loading is automatically done in the background using threads (but sound files loading is not).

Using a thread to perform a one-time initialization task for an applet is a variation of the classic **producer/consumer scenario.**

Using threads to perform one time initialization(2)

Sound loading example:

- SoundLoader (Thread)
- SoundList (List of sounds)
- SoundExample (Applet)

Scenario:

- SoundExample asks in init method SoundList object to load sounds
- SoundList creates separate thread (SoundLoader) for each file
- When SoundLoader finishes loading, it puts file to the SoundList
- When user asks an Applet to play a sound, SoundExample checks if it is already available (checks for proper position in SoundList). If it is, plays it. Otherway asks user to wait.

Before you use your applet

1. **Have you removed or disabled debugging output?**
2. **Does the applet stop running when it's offscreen?**
3. **If the applet does something that might get annoying -- play sounds or animation, for example -- does it give the user a way of stopping the annoying behavior?**

Well finished applet

Make your applet as flexible as possible.

You can often define parameters that let your applet be used in a variety of situations without any rewriting.

Implement the `getParameterInfo` and getAppletInfo methods.

Implementing this method now might make your applet easier to customize. Currently browsers doesn't support these methods but they are expected to do it in the future.

Packaging an Applet into JAR file

An important use of the JAR utility is to optimize applet loading.
Even if applet consists of one .class file this file is uncompressed so downloading is slower than it could be.
Example of packaging multiple .class files into one file:

*jar cf TicTacToe.jar *.class*

Usage on HTML page:

```
<applet    code=TicTacToe.class
           archive=TicTacToe.jar
           width=200
           height=100>
</applet>
```

Presented by Bartosz Sakowicz
DMCS TUL

Servlet - Applet comunnication

There are three (generally six) ways of communication between servlet and applet:

- Using HTTP protocol (messages as text or as serialized objects - ObjectInputStream and ObjectOutputStream)
- Using socket connections (the same)
- Using RMI (or CORBA)

Using HTTP protocol

Advantages:

- It's easy to write. The applet can take advantage of the java.net.URL and java.net.URLConnection classes to manage the communication channel.
- It works even for applets running behind a firewall.

Disadvantages:

- It's slow. It has to be reestablished a new communication channel for each request and response.
- It requires requests to be formed as an array of name/value pairs
- Only the applet can initiate communication.

Using socket connections

Advantages:

- It allows bidirectional, sustained communication. The applet and servlet can use the same socket (or even several sockets) to communicate interactively, sending messages back and forth.

Disadvantages:

- IIt fails for applets running behind firewalls.
- It can be complicated to write the code that runs on the server. There must always be some process listening on a well-known port on the server machine.
- It may require the development of a custom protocol. The applet and server need to define the protocol they use for the communication.

Using RMI

Advantages:

- It allows applets and server objects to communicate using an elegant high-level, object-oriented paradigm.
- It allows server objects to make callbacks to the methods of the applet.
- It can be made to work through firewalls (though it doesn't like it, and current browsers don't support it very well). The RMI transport layer normally relies on direct socket connections to perform its work. When an applet executes behind a firewall its socket connections fail. In this case, the RMI transport layer can automatically begin operating entirely within the HTTP protocol. This is not without cost. The HTTP overhead affects performance, and the HTTP request/response paradigm cannot support callbacks.

Using RMI(2)

Disadvantages:

- It's complicated. RMI communication uses special stub and skeleton classes for each remote object, and it requires a naming registry from which clients can obtain references to these remote objects.
- It's supported in few browsers (Netscape 4 +). Microsoft's Internet Explorer do not support RMI without installing a special plug-in.
- It can be used only by Java clients.

The hybrid approach

How should our applet communicate with its stock feed server? **It depends:**

- If we can guarantee that all our potential clients support it, RMI's elegance and power make it an ideal choice.
- When RMI isn't available, the bidirectional capabilities of the non-HTTP socket connection could be attractive. Unfortunately, that bidirectional communication becomes nonexistent communication when the applet ends up on the far side of a firewall.
- If there is no other solution use HTTP communication. It's straightforward to implement and works on every Java-enabled client. And if you can guarantee that the client supports JDK 1.1 you can use object serialization.

The hybrid approach(2)

The best solution is to use every solution, with servlets. Servlets make it possible to combine the HTTP, non-HTTP, and RMI applet-server communication techniques, supporting them all with a single servlet.

Why would anyone want to do this?

It's a handy technique when an applet wants to communicate using RMI or a non-HTTP protocol but needs to fallback to HTTP when necessary (such as when it finds itself behind a firewall).

By using the same servlet to handle every client, the core server logic and the server state can be collected in one place. When you control your environment you can drop one or more of these protocols.

Implementations

HTTP:

Applet needs to send to server normal text containg HTTP well formed request. Useful classes are URL and URLConnection.

Sockets:

Traditional multithreaded server - client solution usually served by additional class on the server side.

RMI:

Introduced later.