Java™ Servlet API Specification

Version 2.1a

James Duncan Davidson
with Suzanne Ahmed

THE NETWORK IS THE COMPUTER™

Java Software Division
A Division of Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
415 960-1300 fax 415 969-9131

November 1998

Please send all comments to **servletapi-feedback@eng.sun.com**.

Contents

Preface

This document, the *Java™ Servlet API Specification*, describes Version 2.1 of the Java Servlet API. In addition to this specification, the Java Servlet API has Javadoc documentation and a reference implementation available for public download at the following location:

`http://java.sun.com/products/servlet/index.html`

Who Should Read This Specification

This specification is intended as the definitive description of the Java Servlet API, Version 2.1. As such, it will be of interest to both servlet developers and servlet engine developers.

Parts of the Java Servlet API

The Java Servlet API is divided into two packages—an HTTP-specific package and a generic, non-HTTP-specific package. The two packages will allow the Java Servlet API to be adapted to other request-response protocols in the future.

The two packages are described in this specification, as well as in the Javadoc documentation and the reference implementation. The Javadoc documentation describes how you use each method in the API.

The reference implementation provides a behavioral benchmark. In the case of a discrepancy, the order of resolution is the specification (this document), then the Javadoc documentation, and finally the reference implementation.

Important References

You may be interested in the following Internet specifications that are relevant to the development and implementation of the Servlet API. You can locate online versions of any of these RFCs at the following location:

`http://info.internet.isi.edu/7c/in-notes/rfc/.cache`

- RFC 1738 Uniform Resource Locators (URL)
- RFC 1808 Relative Uniform Resource Locators
- RFC 1945 Hypertext Transfer Protocol (HTTP/1.0)
- RFC 2045 MIME Part One: Format of Internet Message Bodies
- RFC 2046 MIME Part Two: Media Types
- RFC 2047 MIME Part Three: Message Header Extensions for Non-ASCII Text
- RFC 2048 MIME Part Four: Registration Procedures
- RFC 2049 MIME Part Five: Conformance Criteria and Examples
- RFC 2068 Hypertext Transfer Protocol (HTTP/1.1)
- RFC 2069 An Extension to HTTP: Digest Access Authentication
- RFC 2109 HTTP State Management Mechanism
- RFC 2145 Use and Interpretation of HTTP Version Numbers
- RFC 2324 Hypertext Coffee Pot Control Protocol (HTCPCP/1.0)[1]

The World Wide Web Consortium (`http://www.w3.org`) is a source of HTTP-related information that affects this specification and its implementations.

[1] A tongue-in-cheek reference.

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Acknowledgements

Many individuals and companies have given of their valuable skills and talent to this specification and the Java Servlet API.

The author gratefully acknowledges each of the following companies for contributing to the definition of the Java Servlet API—Art Technology Group, BEA Weblogic, IBM, Gefion Software, Live Software, Netscape Communications Corporation, New Atlanta Communications, The Apache Group, and Sun Microsystems, Inc.

The author also gratefully acknowledges the following individuals, each of whom has contributed in his or her unique way—Adam Messinger, Anselm Baird-Smith, Bob Pasker, Jason Hunter, Alan Williamson, Jon Stevens, Robert Clark, Rod McChesney, Satish Dharmaraj, Nathan Abramson, Stefano Mazzocchi, Jim Driscoll, Connie Weiss, and Suzanne Ahmed.

CHAPTER 1

About Java Servlets

Java™ servlets are small, platform-independent Java programs that can be used to extend the functionality of a Web server in a variety of ways. Servlets are to the server what applets are to the client—small Java programs compiled to bytecode that can be loaded dynamically and that extend the capabilities of the host.

Servlets differ from applets in that servlets do not run in a Web browser or with a graphical user interface. Instead, servlets interact with the servlet engine running on the Web server through requests and responses. The request-response paradigm is modeled on the behavior of the HyperText Transfer Protocol (HTTP).

A client program, which could be a Web browser or some other program that can make connections across the Internet, accesses a Web server and makes a request. This request is processed by the servlet engine that runs with the Web server, which returns a response to a servlet. The servlet in turn sends a response in HTTP form to the client.

In functionality, servlets lie somewhere between Common Gateway Interface (CGI) programs and proprietary server extensions such as the Netscape Server API (NSAPI). Unlike CGI programs and NSAPI modules, you do not need to modify servlets to be specific to either a platform or a server.

Overview of Java Servlets

Servlets have the following advantages over other common server extension mechanisms:

- They are faster than CGI scripts because they use a different process model.
- They use a standard API that is supported by many Web servers.
- They have all of the advantages of the Java language, including ease of development and platform independence.
- They can access the large set of APIs available for the Java platform.

Note – The methods described in this specification are methods in the classes and interfaces of the Java Servlet API. For more information, refer to the API reference in Chapter 2.

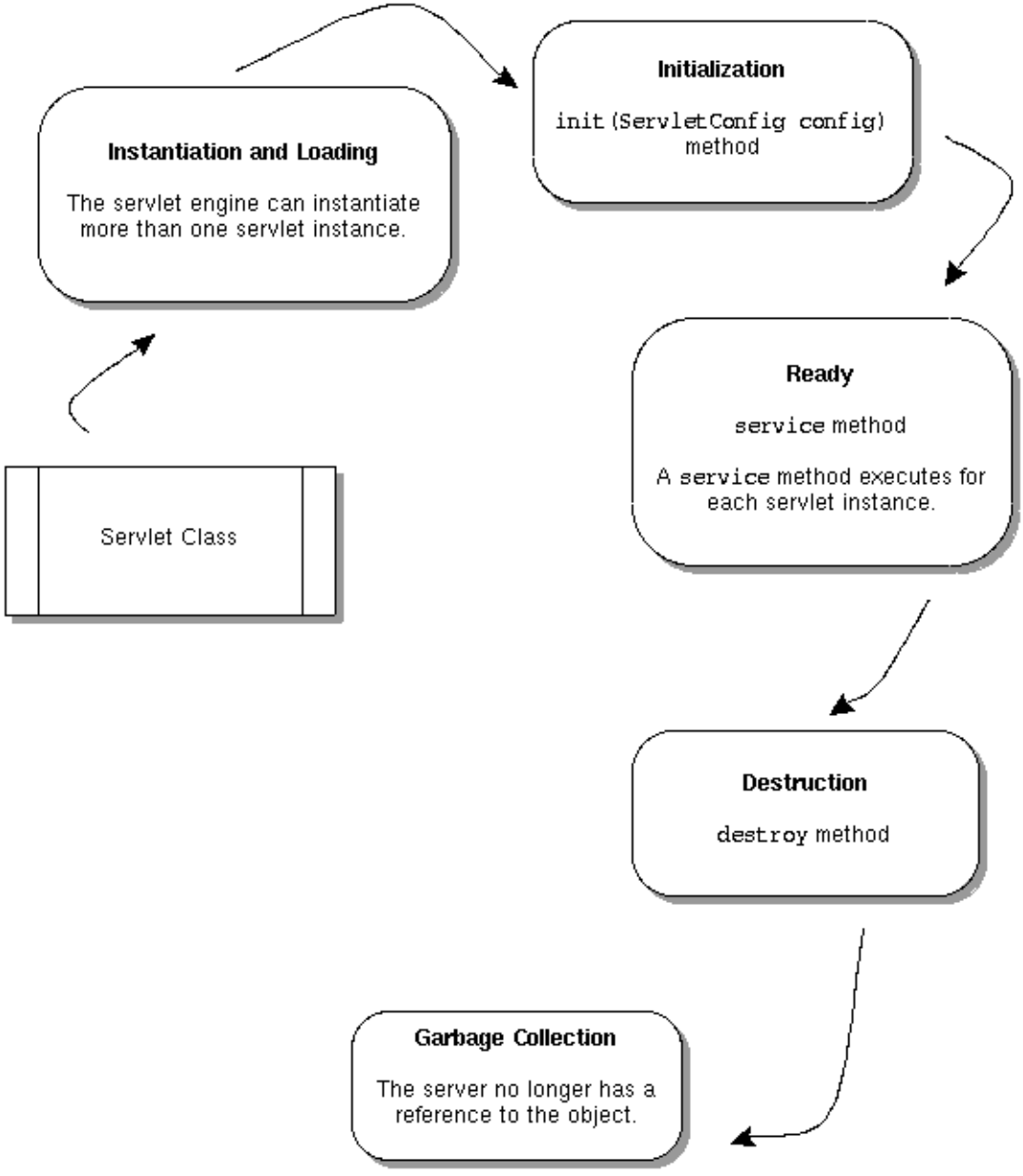
Servlet Lifecycle

A Java servlet has a lifecycle that defines how the servlet is loaded and initialized, how it receives and responds to requests, and how it is taken out of service. In code, the servlet lifecycle is defined by the `javax.servlet.Servlet` interface.

All Java servlets must, either directly or indirectly, implement the `javax.servlet.Servlet` interface so that they can run in a servlet engine. The servlet engine is a customized extension to a Web server for processing servlets, built in conformance with the Java Servlet API by the Web server vendor. The servlet engine provides network services, understands MIME requests, and runs servlet containers.

The `javax.servlet.Servlet` interface defines methods that are called at specific times and in a specific order during the servlet lifecycle. The entire servlet lifecycle is shown in FIGURE 1-1.

FIGURE 1-1 The Servlet Lifecycle



How a Servlet is Loaded and Instantiated

The servlet engine instantiates and loads a servlet. The instantiation and loading can occur when the engine starts, when it needs the servlet in order to respond to a request, or any time in between.

The servlet engine loads a servlet using the Java class loading facility. The servlet engine can load the servlet from the local file system, a remote file system, or a network source.

How a Servlet is Initialized

After the servlet engine loads the servlet, the engine must initialize the servlet. Initialization is a good time for a servlet to read any persistent data it may have stored, initialize JDBC database connections, and establish references to other costly resources.

During initialization, the `init` method of the `javax.servlet.Servlet` interface gives the servlet initialization information, so that the servlet has an opportunity to configure itself.

The `init` method takes a servlet configuration object (of type `ServletConfig`) as a parameter. The servlet configuration object is implemented in the servlet engine and allows the servlet to access name-value parameters from the engine's configuration information. The servlet configuration object also gives the servlet access to a servlet context object, of type `ServletContext`.

How a Servlet Handles Requests

After the servlet is initialized, it is ready to handle requests from the client. Each client request that is made of a servlet is represented by a servlet request object (of type `ServletRequest`). The response the servlet sends to the client is represented by a servlet response object (of type `ServletResponse`).

When the client makes a request, the servlet engine passes both the servlet request object and the servlet response object to the servlet. The objects are passed as parameters to the `service` method, defined in the `Service` interface and which the servlet implements.

The servlet can also implement the `ServletRequest` or `ServletResponse` interfaces, or both. The `ServletRequest` interface gives the servlet access to the request parameters the client sends, such as form data, request information, and protocol methods. The servlet can read the request data from an input stream object (of type `ServletInputStream`).

The `ServletResponse` interface allows the servlet to set response headers and status codes. By implementing `ServletResponse`, the servlet has access to an output stream object (of type `ServletOutputStream`) that it can use to return data to the client.

Multithreading and Mapping

In a multithreaded environment, most servlets must be written to handle multiple concurrent requests. The exception is a servlet that implements the `SingleThreadModel` interface. Such a servlet will execute only one request thread at a time.

A servlet responds to a client request according to the servlet engine's mapping. A mapping pairs a servlet instance with an URL to which the servlet returns data, for example, `HelloServlet` with `/hello/index.html`.

However, a mapping might pair an URL with more than one servlet instance. For example, a distributed servlet engine running on more than one server might have a servlet instance running on each server, to balance the processing load. As a servlet developer, you cannot assume that a servlet has only one instance.

How a Servlet is Destroyed

The servlet engine is not required to keep a servlet loaded for any period of time or for the life of the server. Servlet engines are free to use servlets or retire them at any time. Therefore, you should not rely on class or instance members to store state information.

When the servlet engine determines that a servlet should be destroyed (for example, if the engine is shut down or needs to conserve resources), the engine must allow the servlet to release any resources it is using and save persistent state. To do this, the engine calls the servlet's `destroy` method.

The servlet engine must allow any calls to the `service` method either to complete or to end with a time out (as the engine defines a time out) before the engine can destroy the servlet. Once the engine destroys a servlet, the engine cannot route any more requests to the servlet. The engine must release the servlet and make it eligible for garbage collection.

Servlet Mapping Techniques

As a servlet engine developer, you have a great deal of flexibility in how you map client requests to servlets. This specification does not mandate how the mapping should take place. However, you should feel free to use any of the following techniques:

- You can map a servlet to just one URL.

  For example, you can specify that a particular servlet is only called by requests from the file URL `/feedback/index.html`.

- You can map a servlet to any URL that begins with a certain directory name.

  For example, if you map a servlet to the URL `/catalog`, requests from `/catalog/`, `/catalog/garden`, and `/catalog/housewares/index.html` would also map to the servlet. However, requests from `/catalogtwo` or `/catalog.html` would not.

- You can map a servlet to any URL that ends with a certain file name extension.

  For example, you can map a request for any file whose name ends in `.thtml` to a particular servlet. If you use both file name extension mapping and directory mapping in your servlet engine, design the engine so that the file name resolution takes place after the directory name resolution fails.

- You can map a servlet by using the special URL `/servlet/`*servlet_name*.

  For example, if you create a servlet with the name `listattributes`, you can access the servlet by using the URL `/servlet/listattributes`.

- You can invoke a servlet by its class name.

  For example, if a servlet engine receives a request from the URL `/servlet/com.foo.servlet.MailServlet`, the servlet engine can load the class `com.foo.servlet.MailServlet`, instantiate it, cast the instance to a servlet, and then let the servlet handle the request.

The Servlet Context

The `ServletContext` interface defines a servlet context object, that is, an object that defines the servlet's view of the servlet engine. By using a servlet context, servlets can log events and obtain resources and objects (such as `RequestDispatcher` objects) from the servlet engine. A servlet can run in only one servlet context, but different servlets can have different views of the servlet engine.

If a servlet engine supports virtual hosts, each virtual host has a servlet context. A servlet context cannot be shared across virtual hosts.

Servlet engines can allow a servlet context to have as its scope part of a server's URL path.

For example, a servlet context belonging to a bank application can be mapped to the path `/bank`. In this case, a call to the `getContext` method (`/bank/foo`) would return the servlet context for `/bank`. Such a mapping will become part of the upcoming Deployment Descriptor specification.

HTTP Sessions

The HyperText Transfer Protocol (HTTP) is a stateless protocol. To build effective Web server applications, you must be able to identify a series of unique requests from a remote client as being from the same client. Many strategies for session tracking have evolved over time, but all are difficult or troublesome to use.

The Java Servlet API provides a simple interface that allows a servlet engine to use any number of approaches to track a user session.

Creating a Session

Because HTTP is a request-response protocol, a session is considered new until the client joins it. *Join* means that the client returns session tracking information to the server, indicating that a session has been established. Until the client joins a session, you cannot assume that the next client response is part of the current session.

The session is considered to be new if either of the following is true:

- The client does not yet know about the session.
- The client chooses not to join a session, for example, if the client declines to accept cookies sent by the server.

As a servlet developer, you must design your Web application to handle situations in which a client has not, or can not, join a session. The server will maintain the session object for a period of time specified by the Web server or a servlet. When a session expires, the server will release the session object and all other objects bound during the session.

Binding Objects into a Session

You might want to bind objects into a session, if it helps you handle your application's data requirements. You can bind any object into a session by a unique name, using the `HttpSession` object. Any object bound into a session is available to any other servlet that handles a request from the same session.

Some objects may require that you know when they are placed into, or removed from, a session. You can obtain this information by using the `HttpSessionBindingListener` interface. When your application stores data in or removes data from the session, the servlet engine checks whether the object being bound or unbound implements `HttpSessionBindingListener`. If it does, methods in the interface notify the object that it has been bound or unbound.

CHAPTER 2

API Class Reference

This section contains the detailed specification for each class and interface in the Java Servlet API. The API reference in this specification is similar to the Javadoc API reference, but this specification provides more information.

The API consists of 2 packages, 12 interfaces, and 9 classes, as shown in Table 2-1.

TABLE 2-1 Packages in the Java Servlet API

Package javax.servlet

Type	Name	Page Number
Interface	`RequestDispatcher`	Page 20
Interface	`Servlet`	Page 22
Interface	`ServletConfig`	Page 24
Interface	`ServletContext`	Page 25
Interface	`ServletRequest`	Page 30
Interface	`ServletResponse`	Page 34
Interface	`SingleThreadModel`	Page 36
Class	`GenericServlet`	Page 37
Class	`ServletInputStream`	Page 40
Class	`ServletOutputStream`	Page 41
Class	`ServletException`	Page 43
Class	`UnavailableException`	Page 44

Package javax.servlet.http

Type	Name	Page Number
Interface	`HttpServletRequest`	Page 46
Interface	`HttpServletResponse`	Page 51
Interface	`HttpSession`	Page 56
Interface	`HttpSessionBindingListener`	Page 60
Interface	`HttpSessionContext`	Page 61
Class	`Cookie`	Page 62
Class	`HttpServlet`	Page 66
Class	`HttpSessionBindingEvent`	Page 70
Class	`HttpUtils`	Page 71

Interface RequestDispatcher

Definition

```
public interface RequestDispatcher;
```

Defines a request dispatcher object that receives requests from the client and sends them to any resource (such as a servlet, CGI script, HTML file, or JSP file) available on the Web server. The request dispatcher object is created by the servlet engine and serves as a wrapper around a server resource defined by a particular URL.

The `RequestDispatcher` interface is defined primarily to wrap servlets, but a servlet engine can create request dispatcher objects to wrap any type of resource.

Request dispatcher objects are created by the servlet engine, not by the servlet developer.

Methods

forward

```
public void forward(ServletRequest request, ServletReponse response)
    throws ServletException, IOException;
```

Used for forwarding a request from this servlet to another resource on the Web server. This method is useful when one servlet does preliminary processing of a request and wants to let another object generate the response.

The `request` object passed to the target object will have its request URL path and other path parameters adjusted to reflect the target URL path of the target object.

You cannot use this method if a `ServletOutputStream` object or `PrintWriter` object has been obtained from the response. In that case, the method throws an `IllegalStateException`.

include

```
public void include(ServletRequest request, ServletResponse response)
    throws ServletException, IOException
```

Used for including the content generated by another server resource in the body of a response. In essence, this method enables programmatic server-side includes.

The request object passed to the target object will reflect the request URL path and path info of the calling request. The response object only has access to the calling servlet's `ServletOutputStream` object or `PrintWriter` object.

An included servlet cannot set headers. If the included servlet calls a method that needs to set headers (such as cookies), the method is not guaranteed to work. As a servlet developer, you must ensure that any methods that might need direct access to headers are properly resolved. To ensure that a session works correctly, start the session outside the included servlet, even if you use session tracking.

Interface Servlet

Definition

```
public interface Servlet
```

The `Servlet` interface defines a servlet, a Java object that extends the functionality of a Web server.

Methods

init

```
public void init(ServletConfig config) throws ServletException;
```

The servlet engine calls the `init` method exactly once on a servlet, after the servlet is instantiated but before it is placed into service. The `init` method must exit successfully before you can call the `service` method.

If the `init` method throws a `ServletException`, you must not place the servlet into service. If the `init` method does not complete within the timeout period, you can assume the servlet is nonfunctional and not in service.

service

```
public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException;
```

Called by the servlet engine to allow the servlet to respond to a request. This method must not be called until the servlet has been successfully initialized. The servlet engine must block pending requests to this servlet until such time as it is initialized.

After a servlet object has been destroyed, the servlet engine must not call `service` until a new servlet is initialized.

destroy

```
public void destroy();
```

Called by the servlet engine when the servlet is removed from service. The servlet engine may not call this method until all threads within the object's service `method` have exited or the engine's timeout period has passed.

getServletConfig

```
public ServletConfig getServletConfig();
```

Returns a `ServletConfig` object. As a servlet developer, you must store the `ServletConfig` object passed to the `init` method so that the `getServletConfig` method can return the object. For your convenience, the `GenericServlet` implementation of this interface already does this.

getServletInfo

```
public String getServletInfo();
```

Allows the servlet to provide information about itself to the host servlet runner. The returned string must be of plain text form and not composed of markup of any kind (such as HTML, XML, etc).

Interface ServletConfig

Definition

```
public interface ServletConfig
```

This interface defines an object that a servlet engine generates to configure a servlet and allow the servlet to obtain a reference to its `ServletContext` interface. Each `ServletConfig` object the servlet receives is unique to that servlet.

Methods

getInitParameter

```
public String getInitParameter(String name);
```

This method returns a String containing the value of the servlet's named initialization parameter, or null if this parameter does not exist.

getInitParameterNames

```
public Enumeration getInitParameterNames();
```

This method returns an enumeration of String objects containing the names of the initialization parameters for the calling servlet. If the calling servlet has no initialization parameters, `getInitParameterNames` returns an empty enumeration.

getServletContext

```
public ServletContext getServletContext();
```

Returns the `ServletContext` object for this servlet.

Interface ServletContext

Definition

```
public interface ServletContext
```

Defines a servlet context object that the servlet engine generates to provide a servlet with information about its environment.

A servlet context object is at least as unique as the host in which it resides. In a servlet engine that handles multiple virtual hosts (for example, by using the HTTP 1.1 host header), each virtual host must be treated as a separate context. Servlet engines can also provide context objects that are unique to a group of servlets. You can group the servlets administratively or define them with a deployment descriptor.

Methods

getAttribute

```
public Object getAttribute(String name);
```

Returns an object that is known to the servlet context object by a given name, or null if there is no such object associated with the name. This method allows access to additional information about the servlet engine not already provided by other methods in this interface.

You should use the same naming conventions for attributes as for package names. Names matching `java.*`, `javax.*`, and `sun.*` are reserved for definition by this specification or the reference implementation.

getAttributeNames

```
public Enumeration getAttributeNames();
```

Returns an enumeration of the attribute names present in the calling servlet context object.

getContext

```
public ServletContext getContext(String uripath);
```

Returns the servlet context object that contains servlets and resources for a particular URI path, or null if a context cannot be provided for the path. The format of the URI path is `/dir/dir/filename.ext`.

For security, the servlet context object that a sandboxed or otherwise restricted servlet has access to should return null to this method call.

getMajorVersion

```
public int getMajorVersion();
```

Returns the major version of the Servlet API that this servlet engine supports. All 2.1 compliant implementations must return the integer 2 to this method.

getMinorVersion

```
public int getMinorVersion();
```

Returns the minor version of the Servlet API that this servlet engine supports. All 2.1 compliant implementations must return the integer 1 to this method.

getMimeType

```
public String getMimeType(String file);
```

Returns the MIME type of the specified file, or null if not known. The MIME type is determined according to the configuration of the servlet engine.

getRealPath

```
public String getRealPath(String path);
```

Applies alias rules to the specified virtual path in URL path format, that is, `/dir/dir/filename.ext`. Returns a String representing the corresponding real path in the format that is appropriate for the machine (including the proper path separators) that the servlet engine is running on.

Returns null if the translation could not be performed for any reason.

getResource

```
public URL getResource(String uripath);
```

Returns an URL object to a resource known to the servlet context object located at the given URL path (of format `/dir/dir/filename.ext`). The servlet engine must implement whatever URLStreamHandlers are necessary to access the given context. If there is no resource known to a servlet context for a particular path, `getResource` returns null.

This method does not fill the same purpose as the `getResource` method in `java.lang.Class`. The method in `java.lang.Class` looks up resources based on `Class` class loader. This allows the server to make content visible to any servlet from any source without regard to class loaders, location, and so on.

getResourceAsStream

```
public InputStream getResourceAsStream(String uripath);
```

Returns an `InputStream` object that refers to content known to the servlet context object at the specified the URL, or null if the servlet context object is not found. The URL path is of the form `/dir/dir/filename.ext`.

This method is a convenient way to obtain an URL object from the `getResource` method and open a stream. Note that meta-information such as content length and content type are lost when you use `getResourceAsStream`.

getRequestDispatcher

```
public RequestDispatcher getRequestDispatcher(String uripath);
```

Returns a `RequestDispatcher` object for the specified URL if the context knows of an active source (such as a servlet, JSP page, CGI script, etc.) of content for the particular path, or null otherwise. The format of the URL path is `/dir/dir/filename.ext`). The servlet engine implements whatever functionality is needed to wrap the target path with a request dispatcher object that can perform request forwarding and programmatic server side includes.

getServerInfo

```
public String getServerInfo();
```

Returns a String object containing at least the name and version of the servlet engine.

log

```
public void log(String msg);
public void log(String msg, Throwable t);
public void log(Exception exception, String msg); // deprecated
```

Writes the specified message to the log file for this servlet context object. The log written to is specific to the servlet engine, but is usually an event log. When this method is called with an exception, the stack trace should be included in the log.

setAttribute

```
public void setAttribute(String name, Object o);
```

Binds the object you specify to the name you specify in the servlet context object.

Attribute names should follow the same convention as package names. Names beginning with the prefixes `java.*`, `javax.*`, and `sun.*` are reserved for definition by this specification or the reference implementation

removeAttribute

```
public void removeAttribute(String name);
```

Removes an attribute from the specified servlet context object.

Deprecated Methods

getServlet

```
// deprecated
public Servlet getServlet(String name) throws ServletException;
```

Originally defined to return a servlet with the specified name, or null if not found. When the servlet is returned it is already initialized and ready to accept service requests.

This is a dangerous method. When this method is called, the state of the servlet may not be known and this could cause problems with the server's state machine. It is also a security risk to allow any servlet to be able to access the methods of another servlet.

All servlet implementations should always return null to this call. As this method is defined to always return null, it will not be removed at this time to preserve binary compatibility with previous versions of the API. This method will be removed in a future revision of the API.

Deprecated as of Version 2.0.

getServletNames

```
// deprecated
public Enumeration getServletNames();
```

Originally defined to return an enumeration of String objects containing all the servlet object names known to this servlet context. The enumeration always includes the servlet itself.

This is a dangerous method, for the same reasons the `getServlet` method is dangerous.

All servlet implementations should return an empty enumeration to this call. As this method is defined to return an empty enumeration, it will not be removed at this time to preserve binary compatibility with previous versions of the API. This method will be removed in a future revision of the API.

Deprecated as of Version 2.0.

getServlets

```
// deprecated
public Enumeration getServlets();
```

Originally defined to return an enumeration of all the Servlet objects which are known to this servlet context. The enumeration always includes the servlet itself.

This is a dangerous method, for the same reasons the `getServlet` method is dangerous.

All servlet implementations should return an empty enumeration to this call. As this method is defined to return an empty enumeration, it will not be removed at this time to preserve binary compatibility with previous versions of the API. This method will be removed in a future revision of the API.

Deprecated as of Version 2.0.

Interface ServletRequest

Definition

```
public interface ServletRequest
```

Defines a servlet engine generated object that enables the servlet to get data about a client request. The data provided by this object includes parameter names and values, attributes, and an input stream to read data from the request's body.

Methods

getAttribute

```
public Object getAttribute(String name);
```

Returns the value of the named attribute of this request, or null if the attribute does not exist. This method allows access to request information not already provided by other methods in this interface or data that was placed in the request object by other servlets. Attribute names should follow the same convention as package names. The attribute names matching a prefix of `java.*`, `javax.*`, and `sun.*` are reserved for definition by Sun Microsystems.

getAttributeNames

```
public Enumeration getAttributeNames();
```

Returns an enumeration of all the attribute names contained in the request.

getCharacterEncoding

```
public String getCharacterEncoding();
```

Returns the character set encoding for the input body of this request, or null if no character encoding is defined.

getContentLength

```
public int getContentLength();
```

Returns the MIME-specified content length, or -1 if not known.

getContentType

```
public String getContentType();
```

Returns the Multipurpose Internet Mail Extension (MIME) type of the request body data, or null if not known.

getInputStream

```
public ServletInputStream getInputStream() throws IOException;
```

Returns an input stream for reading binary data from the request body. This method will throw an `IllegalStateException` if a reader was previously obtained via the `getReader` method.

getParameter

```
public String getParameter(String name);
```

Returns a string containing a single value of the specified parameter, or null if the parameter does not exist. For example, in an HTTP servlet, this method would return the value of a specified query string parameter or posted form data parameter. In the event that there are multiple parameter values for a single name, the value returned should be the first value in the array returned by the `getParameterValues` method. If the parameter has (or could have) multiple values, servlet writers should use the `getParameterValues` method.

getParameterNames

```
public Enumeration getParameterNames();
```

Returns all the parameter names for this request as an enumeration of String objects, or an empty enumeration if there are no input parameters.

getParameterValues

```
public String[] getParameterValues(String name);
```

Returns the values of the specified parameter as an array of String objects, or null if the named parameter does not exist.

getProtocol

```
public String getProtocol();
```

Returns the protocol being used for this request as a string of the form `protocol/major_version.minor_version`. An HTTP 1.0 request, as defined by the HTTP 1.0 specification, should return the string `HTTP/1.0`.

getReader

```
public BufferedReader getReader() throws IOException;
```

This method returns a buffered reader for reading text data from the request body. The character encoding of the reader is set according to the request data. This method must throw an `IllegalStateException` if the input stream of the request had been obtained with the `getInputStream` call.

getRemoteAddr

```
public String getRemoteAddr();
```

Returns the IP address of the agent that sent the request.

getRemoteHost

```
public String getRemoteHost();
```

Returns the fully qualified host name of the agent that sent the request. If the engine cannot or chooses not to resolve the hostname (to improve performance), this method returns the dotted-string form of the IP address.

getScheme

```
public String getScheme();
```

Returns the scheme of the URL used in this request. For example, in an HTTP request, the scheme would be `http`.

getServerName

```
public String getServerName();
```

Returns the host name of the server that received the request.

getServerPort

```
public int getServerPort();
```

Returns the port number on which this request was received.

setAttribute

```
public void setAttribute(String name, Object object);
```

This method places an attribute into the request for later use by other objects which will have access to this request object such as nested servlets.

Deprecated Methods

getRealPath

```
// deprecated
public String getRealPath(String path);
```

Applies alias rules to the specified virtual path and returns the corresponding real path, or null if the translation cannot be performed for any reason.

This method has been deprecated in preference to the `getRealPath` method in the `ServletContext` interface. In Version 2.1, the `ServletContext` interface was clarified to contain all the path mapping information available to a servlet. Implementations should return the same string as the `getRealPath` method in `ServletContext`.

Deprecated as of Version 2.1.

Interface ServletResponse

Definition

```
public interface ServletResponse
```

Defines an object generated by the servlet engine that allows a servlet to respond to a client request. This response is a MIME entity and could be an HTML page, image data, or any other MIME format.

Methods

getCharacterEncoding

```
public String getCharacterEncoding();
```

Returns the character set encoding used for this MIME body. The character encoding is either the one specified in the assigned content type, or the best match determined in part by the request header information from the client indicating what character encodings are acceptable to the client. In HTTP, this information is transmitted to the servlet engine via the “Accept-Charset” header.

See RFC 2047 for more information about character encoding and MIME.

getOutputStream

```
public ServletOutputStream getOutputStream() throws IOException;
```

Returns an output stream for writing binary response data.

Throws `IllegalStateException` if `getWriter` has already been called on the response object.

getWriter

```
public PrintWriter getWriter throws IOException;
```

This method returns a print writer for writing formatted text responses. The MIME type of the response will be modified, if necessary, to reflect the character encoding being used through the charset property. The content type of the response should be set before calling this method.

Throws `UnsupportedEncodingException` if no such encoding can be provided. Throws `IllegalStateException` if `getOutputStream` has already been called on this same request.

setContentLength

```
public void setContentLength(int length);
```

Sets the content length for this response. This method will overwrite any previously set content length.

In order for this method to successfully set the content length header of the response, this method must be called before the response is committed to the underlying output stream.

setContentType

```
public void setContentType(String type);
```

This method sets the content type for this response. This type may later be implicitly modified by the addition of properties such as the MIME charset property if the service finds it necessary and the appropriate property has not been set.

In order for this method to successfully set the content type header of the response, this method must be called before the response is committed to the underlying output stream.

Interface SingleThreadModel

Definition

```
public interface SingleThreadModel;
```

This empty interface allows servlet implements to specify how the system should handle concurrent calls to the same servlet. If the target servlet is flagged with this interface, the servlet programmer is guaranteed that no two threads will execute concurrently the service method of the servlet.

This guarantee can be accomplished by the servlet engine by maintaining a pool of separate servlet instances of each servlet so marked, or by only letting one thread execute the service method of the servlet at a time.

Class GenericServlet

Definition

```
public abstract class GenericServlet implements Servlet,
    ServletConfig, Serializable;
```

This class is a convenience implementation to aid servlet writers. It provides simple implementations of the servlet lifecycle methods as well as holds a reference to the `ServletConfig` and `ServletContext` objects provided at initialization time. Select methods from the context object is exposed via methods in this class.

Methods

destroy

```
public void destroy();
```

This implementation of the destroy method does nothing.

getInitParameter

```
public String getInitParameter(String name);
```

Convenience method which in turns calls the method of the same name on the stored servlet configuration object.

getInitParameterNames

```
public Enumeration getInitParameterNames();
```

Convenience method which in turns calls the method of the same name on the stored `ServletConfig` object.

getServletConfig

```
public ServletConfig getServletConfig();
```

Returns a reference to the `ServletConfig` object stored by the `init` method of this class.

getServletContext

```
public ServletContext getServletContext();
```

Convenience method which in turns calls the method of the same name on the stored `ServletConfig` object.

getServletInfo

```
public String getServletInfo();
```

Returns an empty servlet info string.

init

```
public void init() throws ServletException;
public void init(ServletConfig config) throws ServletException;
```

The `init(ServletConfig config)` method is a simple implementation of the servlet lifecycle initialization method. This is the version of this method that the servlet engine calls when the servlet is loaded.

The `init()` method is provided so that if you extend the `GenericServlet` class, you do not need to store the `config` object properly or call `super.init(config)`.

The `init(ServletConfig config)` method saves the `config` object and calls `init()`. If you write a method that overrides `init(ServletConfig config)`, you must call `super.init(config)`, so that the other methods in the `GenericServlet` class function correctly.

log

```
public void log(String msg);
public void log(String msg, Throwable cause);
```

Writes the class name of the servlet and the given message to the log accessible via the servlet context object.

service

```
public abstract void service(ServletRequest request, ServletResponse
    response) throws ServletException, IOException;
```

Abstract method that you must implement if you extend this class, in order to handle network requests.

Class ServletInputStream

Definition

```
public abstract class ServletInputStream extends InputStream
```

This class defines an input stream for reading requests from a client. This is an abstract class that a servlet engine provides. A servlet obtains a reference to a `ServletInputStream` object by using the `ServletRequest` interface.

Subclasses of this class must provide an implementation of the `read` method from the `InputStream` interface.

Methods

readLine

```
public int readLine(byte[] b, int off, int len) throws IOException;
```

Reads into the portion of the given array specified by the offset and length parameters bytes from the input until all requested bytes have been read or a new line character is encountered, in which case the new line character is read into the array as well and then stops.

Class ServletOutputStream

Definition

```
public abstract class ServletOutputStream extends OutputStream
```

This is an abstract class for servlet engines to implement. Servlet developers obtain a reference to an object of this type through the `ServletResponse` interface and use this output stream to return data to clients.

Subclasses of this class must provide an implementation of the `write(int)` method of the `OutputStream` interface.

When a flush or close method is called on an implementation of this interface, any data buffered by the servlet engine is sent to the client and the response is considered to be “committed”. Note that calling close on an object of this type doesn’t necessarily close the underlying socket stream.

Methods

print

```
public void print(String s) throws IOException;
public void print(boolean b) throws IOException;
public void print(char c) throws IOException;
public void print(int i) throws IOException;
public void print(long l) throws IOException;
public void print(float f) throws IOException;
public void print(double d) throws IOException;
```

Prints the argument provided to the underlying output stream.

println

```
public void println() throws IOException;
public void println(String s) throws IOException;
public void println(boolean b) throws IOException;
public void println(char c) throws IOException;
public void println(int i) throws IOException;
public void println(long l) throws IOException;
public void println(float f) throws IOException;
public void println(double d) throws IOException;
```

Prints the argument provided to the underlying output stream followed by a CRLF.

Class ServletException

Definition

```
public class ServletException extends Exception
```

This exception is thrown to indicate a servlet problem.

Constructors

```
public ServletException();
public ServletException(String message);
public ServletException(String message, Throwable cause);
public ServletException(Throwable cause);
```

Constructs a new `ServletException`. If this constructor is called with a `Throwable` parameter, the `Throwable` object is registered as the original cause of this exception.

Methods

getRootCause

```
public Throwable getRootCause();
```

If the original cause of this exception was set, this method returns the cause; otherwise, the method returns null.

Class UnavailableException

Definition

```
public class UnavailableException extends ServletException
```

This exception is thrown to indicate a servlet is either temporarily or permanently unavailable. Servlets may report this exception at any time and the servlet engine running the servlet must behave appropriately.

Temporary unavailability is when the servlet cannot handle requests at the current time due to some transient problem. For example, a server on a different application layer, such as a database, may not be available. The problem may be self-correcting or may require further corrective action.

Permanent unavailability is when the servlet will not be able to handle client requests until some administrative action is taken. For example, the servlet may be misconfigured, or the state of the servlet may be corrupted.

Servlet engines may safely treat both types of exceptions as though they are permanent, but good treatment of temporary unavailability leads to more robust servlet engines. Specifically, requests to the servlet may be blocked (or otherwise deferred) for a servlet suggested amount of time, rather than being rejected until the service itself restarts.

Constructors

```
public UnavailableException(Servlet servlet, String message);
public UnavailableException(int seconds, Servlet servlet,
    String message);
```

Constructs a new exception with the specified descriptive message. If the constructor is called with a given number of seconds, the unavailability is temporary and the value given is an estimate of when the servlet will be able to handle requests again. If the constructor is called without this argument, the servlet is permanently unavailable.

Methods

getServlet

```
public Servlet getServlet();
```

This method returns the servlet that is reporting its unavailability. This is used by the servlet engine to identify the servlet that is affected.

getUnavailableSeconds

```
public int getUnavailableSeconds();
```

Returns the amount of time the servlet expects to be unavailable. Returns -1 if the servlet is permanently unavailable.

isPermanent

```
public boolean isPermanent();
```

Returns true if the servlet is permanently unavailable, indicating that some administrative action must be taken to make the servlet usable.

Interface HttpServletRequest

Definition

```
public interface HttpServletRequest extends ServletRequest;
```

Provides access to HTTP-specific request information to a servlet in an HTTP-based request.

Methods

getAuthType

```
public String getAuthType();
```

Returns the authentication scheme of this request.

getCookies

```
public Cookie[] getCookies();
```

Returns an array containing all the cookies present in this request. If there are no cookies in the request, then an empty array is returned.

getDateHeader

```
public long getDateHeader(String name);
```

Returns the value of the requested header converted to a long representing a date expressed in milliseconds since January 1, 1970, 00:00:00GMT. The match between the given name and the request header is case insensitive.

If the header cannot be converted, this method will throw an `IllegalArgumentException`. If the header requested does not exist, this method returns -1.

getHeader

```
public String getHeader(String name);
```

Returns the value of the requested header. The match between the given name and the request header is case-insensitive.

If the header requested does not exist, this method returns null.

getHeaderNames

```
public Enumeration getHeaderNames();
```

This method returns an enumeration of String objects representing the header names for this request.

Some server implementations may not allow headers to be accessed in this way, in which case this method can return an empty enumeration.

getIntHeader

```
public int getIntHeader(String name);
```

This method returns the value of the specified header converted to an integer. The match between the given name and the request header is case insensitive.

If the header cannot be converted to an integer, this method throws a `NumberFormatException`. If the header requested does not exist, this method returns -1.

getMethod

```
public String getMethod();
```

Returns the HTTP method (for example, GET, POST, PUT) by which this request was made.

getPathInfo

```
public String getPathInfo();
```

This method returns any extra path information of the request URL following the servlet path of this request's URL. If there is a query string as part of the request URL, it is not included in the return value. The path must be URL decoded before being returned. This method returns null if there is no path information following the servlet path of the request URL.

getPathTranslated

```
public String getPathTranslated();
```

This method gets any extra path information following the servlet path of this request's URL and translates it into a real path. The request URL must be URL decoded before the translation is attempted. If there is no extra path information following the servlet path of the URL, this method returns null.

getQueryString

```
public String getQueryString();
```

Returns query string present in the request URL if any. A query string is defined as any information following a ? character in the URL. If there is no query string, this method returns null.

getRemoteUser

```
public String getRemoteUser
```

Gets the name of the user making this request. This information may be provided by HTTP authentication.

This method returns null if there is no user name information in the request.

getRequestedSessionId

```
public String getRequestedSessionId();
```

Returns the session id specified with this request. This may differ from the session id in the current session if the session id given by the client was invalid for whatever reason and a new session was created.

This method will return null if the request does not have a session associated with it.

getRequestURI

```
public String getRequestURI();
```

Returns, from the first line of the HTTP request, the part of this request's URL that defines the resource being requested. If there is a query string, it is not included in the return value. For example a request accessed via the URL path of `/catalog/books?id=1` would return `/catalog/books`. The return value of this method contains both the servlet path and the path info.

If any part of the URL path was URL encoded, the path must be decoded before being returned by this method.

getServletPath

```
public String getServletPath();
```

This method returns the part of the request URL that refers to the servlet being invoked. For example, if a servlet is mapped to the URL path of `/catalog/summer` and a request of the path `/catalog/summer/casual` is made, the servlet path would be `/catalog/summer`.

If the servlet was invoked by some other mechanism than by a path match (such as an extension match), then this method returns null.

getSession

```
public HttpSession getSession();
public HttpSession getSession(boolean create);
```

Returns the current valid session associated with this request. If this method is called with no arguments, a session will be created for the request if there is not allready a session associated with the request. If this method is called with a boolean argument, then the session will be createdonly if the argument is true.

To ensure the session is properly maintained, the servlet developer must call this method before the response is committed.

If the create flag is set to false and no session is associated with this request, then this method will return null.

isRequestedSessionIdValid

```
public boolean isRequestedSessionIdValid();
```

This method checks whether this request is associated with a session that is currently valid. If the session used by the request is not valid, it will not be returned via the `getSession` method.

isRequestedSessionIdFromCookie

```
public boolean isRequestedSessionIdFromCookie();
```

Returns `true` if the session id for this request was provided from the client as a cookie; `false` otherwise.

isRequestedSessionIdFromURL

```
public boolean isRequestedSessionIdFromURL();
```

Returns true if the session id for this request was provided from the client as part of a URL; false otherwise. Note that the spelling `URL` in the method name indicates that the method is new.

Deprecated Methods

isRequestedSessionIdFromUrl

```
// deprecated
public boolean isRequestedSessionIdFromUrl();
```

Deprecated in favor of `isRequestedSessionIdFromURL` for naming consistency. Note that the spelling `Url` in the method name indicates that the method is deprecated.

This method is deprecated as of Version 2.1.

Interface HttpServletResponse

Definition

```
public interface HttpServletResponse extends ServletResponse
```

Represents an HTTP response back to the client. This interface allows a servlet programmer to manipulate HTTP-protocol-specific header information. It is implemented by servlet engine developers for use within servlets.

Member Variables

```
public static final int SC_CONTINUE = 100;
public static final int SC_SWITCHING_PROTOCOLS = 101;
public static final int SC_OK = 200;
public static final int SC_CREATED = 201;
public static final int SC_ACCEPTED = 202;
public static final int SC_NON_AUTHORITATIVE_INFORMATION = 203;
public static final int SC_NO_CONTENT = 204;
public static final int SC_RESET_CONTENT = 205;
public static final int SC_PARTIAL_CONTENT = 206;
public static final int SC_MULTIPLE_CHOICES = 300;
public static final int SC_MOVED_PERMANENTLY = 301;
public static final int SC_MOVED_TEMPORARILY = 302;
public static final int SC_SEE_OTHER = 303;
public static final int SC_NOT_MODIFIED = 304;
public static final int SC_USE_PROXY = 305;
public static final int SC_BAD_REQUEST = 400;
public static final int SC_UNAUTHORIZED = 401;
public static final int SC_PAYMENT_REQUIRED = 402;
public static final int SC_FORBIDDEN = 403;
public static final int SC_NOT_FOUND = 404;
public static final int SC_METHOD_NOT_ALLOWED = 405;
public static final int SC_NOT_ACCEPTABLE = 406;
public static final int SC_PROXY_AUTHENTICATION_REQUIRED = 407;
public static final int SC_REQUEST_TIMEOUT = 408;
public static final int SC_CONFLICT = 409;
public static final int SC_GONE = 410;
public static final int SC_LENGTH_REQUIRED = 411;
public static final int SC_PRECONDITION_FAILED = 412;
public static final int SC_REQUEST_ENTITY_TOO_LARGE = 413;
public static final int SC_REQUEST_URI_TOO_LONG = 414;
```

```
public static final int SC_UNSUPPORTED_MEDIA_TYPE = 415;
public static final int SC_INTERNAL_SERVER_ERROR = 500;
public static final int SC_NOT_IMPLEMENTED = 501;
public static final int SC_BAD_GATEWAY = 502;
public static final int SC_SERVICE_UNAVAILABLE = 503;
public static final int SC_GATEWAY_TIMEOUT = 504;
public static final int SC_HTTP_VERSION_NOT_SUPPORTED = 505;
```

Constants representing HTTP status codes as defined by the HTTP/1.1 working drafts.

Methods

addCookie

```
public void addCookie(Cookie cookie);
```

Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie. This method must be called before the response is committed so that the appropriate headers can be set.

containsHeader

```
public boolean containsHeader(String name);
```

Checks whether or not a given response header has been set.

encodeRedirectURL

```
public String encodeRedirectURL(String url);
```

Encodes the specified URL for use in the `sendRedirect` method or, if encoding is not needed, returns the URL unchanged. This additional encoding method is provided because the rules for determining whether or not to encode the URL may be different in the redirect case. The given URL must be an absolute URL. Relative URLs are not permitted and must throw an `IllegalArgumentException`.

All URLs sent to the `sendRedirect` method should be run through this method to ensure that session tracking is seamless with all browsers.

encodeURL

```
public String encodeURL(String url);
```

Encodes the URL by including the session ID in it, or if encoding is not needed, returns the URL unchanged. URL encoding must be provided by the servlet engine if URL rewriting is present and enabled and there is a valid session for the request that this response is part of and the session is not being maintained via a cookie or other non URL means.

All URLs emitted by a servlet should be run through this method to ensure that session tracking is seamless with all browsers.

sendError

```
public void sendError(int statusCode) throws IOException;
public void sendError(int statusCode, String message) throws
    IOException;
```

Sends an error response to the client using the specified status code. If a message is provided to this method, it is emitted as the response body, otherwise the server should return a standard message body for the error code given.

This is a convenience method that immediately commits the response. No further output should be made by the servlet after calling this method.

sendRedirect

```
public void sendRedirect(String location) throws IOException;
```

Sends a temporary redirect response to the client (`SC_MOVED_TEMPORARILY`) using the specified location. The given location must be an absolute URL. Relative URLs are not permitted and throw an `IllegalArgumentException`.

This method will must be called before the response is committed. This is a convenience method that immediately commits the response. No further output is be made by the servlet after calling this method.

setDateHeader

```
public void setDateHeader(String name, long date);
```

Sets a response header with the specified name and date valued field appropriate for that header. The date is specified in terms of milliseconds since January 1, 1970, 00:00:00GMT. If the header has already been set, the new value overwrites the previous one.

setHeader

```
public void setHeader(String name, String value);
```

Sets a response header with the specified name and field. If the field has already been set, the new value overwrites the previous one.

setIntHeader

```
public void setIntHeader(String name, int value);
```

Sets a response header with the specified name and integer value. If the header has already been set, the new value overwrites the previous one.

setStatus

```
public void setStatus(int statusCode);
```

This method sets the status code of the response. If the status code has already been set, the new value overrides the previous value. If a message is provided, it is sent back as the body of the response.

Deprecated Methods

encodeRedirectUrl

```
// deprecated
public String encodeRedirectUrl(String url);
```

Deprecated in favor of `encodeRedirectURL` for naming consistency.

Deprecated as of Version 2.1.

encodeUrl

```
// deprecated
public String encodeUrl(String url);
```

Deprecated in favor of `encodeURL` for naming consistency.

Deprecated as of Version 2.1.

setStatus

```
// deprecated
public void setStatus(int statusCode, String message);
```

This method sets the status code of the response. If the status code has already been set, the new value overrides the previous value. If a message is provided, it is sent back as the body of the response.

Deprecated as of Version 2.1.

Interface HttpSession

Definition

```
public interface HttpSession
```

This interface is implemented by servlet engine to provide an associates between an HTTP client and an HTTP session. This association, or session, persists over multiple connection and/or requests during a given time period. Sessions are used to maintain state and user identity across multiple page requests over the normally stateless HTTP protocol.

A session can be maintained by either using cookies or URL rewriting.

Methods

getCreationTime

```
public long getCreationTime();
```

Returns the time at which this session was created in milliseconds since January 1, 1970, 00:00:00GMT.

getId

```
public String getId();
```

Returns the identifier assigned to this session. An HTTP session's identifier is a unique string that is created and maintained by the server.

getLastAccessedTime

```
public long getLastAccessedTime();
```

Returns the last time the client sent a request carrying the identifier assigned to the session, or -1 if the session is new. Time is expressed as milliseconds since 00:00:00GMT January 1, 1970.

getMaxInactiveInterval

```
public int getMaxInactiveInterval();
```

Returns the maximum amount of time, in seconds, that a session is guaranteed to be maintained in the servlet engine without a request from the client. After the maximum inactive time, the session may be expired by the servlet engine. If this session will not expire, this method will return -1.

This method should throw an `IllegalStateException` if it is called after this session has been invalidated.

getValue

```
public Object getValue(String name);
```

Returns the object bound to a given name in the session. Returns null if there is no such binding.

This method must throw an `IllegalStateException` if it is called after this session has been invalidated.

getValueNames

```
public String[] getValueNames();
```

Returns an array of the names of all the values bound into this session.

This method should throw an `IllegalStateException` if it is called after this session has been invalidated.

invalidate

```
public void invalidate();
```

This method will invalidate this session. All values bound in the session are removed. Any values that implement the `HttpSessionBindingListener` interface will be notified via the `valueUnbound` method.

isNew

```
public boolean isNew();
```

Returns a Boolean value indicating whether this session is considered to be new. A session is considered to be new if it has been created by the server and not received from the client as part of this request. This means that the client has not "acknowledged" or "joined" the session and may not ever return the appropriate session identification information when it makes its next request.

This method should throw an `IllegalStateException` if it is called after this session has been invalidated.

putValue

```
public void putValue(String name, Object value);
```

Binds the specified object into the session with the given name. Any existing binding with the same name is replaced. Objects placed into the session which implement the `HttpSessionBindingListener` interface will call its `valueBound` method.

Some servlet engine implementations will persist session data or distribute it amongst multiple network nodes. For an object bound into the session to be distributed or persisted to disk, it must implement the `Serializable` interface.

This method should throw an `IllegalStateException` if it is called after this session has been invalidated.

removeValue

```
public void removeValue(String name);
```

Unbinds an object in the session with the given name. If there is no object bound to the given name, this method does nothing. If the object bound to the name implements the `HttpSessionBindingListener`, its `valueUnbound` method will be called.

This method should throw an `IllegalStateException` if it is called after this session has been invalidated.

setMaxInactiveInterval

```
public int setMaxInactiveInterval(int interval);
```

Sets the amount of time that a session can be inactive before the servlet engine is allowed to expire it.

Deprecated Methods

getSessionContext

```
// deprecated
public HttpSessionContext getSessionContext();
```

Returns the context object within which sessions on the server are held. This method has been deprecated as all the methods of `HttpSessionContext` are deprecated. This method should now return an object which has an empty implementation of the `HttpSessionContext` interface.

Deprecated as of Version 2.1.

Interface HttpSessionBindingListener

Definition

```
public interface HttpSessionBindingListener
```

Object that are meant to be placed into HTTP sessions can implement this interface to be notified when they are being bound or unbound from an HTTP session.

Methods

valueBound

```
public void valueBound(HttpSessionBindingEvent event);
```

Method called when an object is bound into a session. This method must be called by the servlet engine during the `putValue` method call in HTTP session.

valueUnbound

```
public void valueUnbound(HttpSessionBindingEvent event);
```

Method called when an object is unbound from a session. This method must be called by the servlet engine during the `removeValue` method call in HTTP session.

It is possible that another servlet instance may have a reference to the object when the object is unbound. The object that is being removed from the session must behave appropriately.

Interface HttpSessionContext

Definition

```
// deprecated
public interface HttpSessionContext
```

An `HttpSessionContext` object is a grouping of HTTP sessions associated with a single entity.

This interface has been deprecated for security reasons and is only present in the current version of the API to preserve compatibility. The methods of this interface have been defined to return values which are legal according to the previous definition of the API.

Deprecated as of Version 2.1.

Methods

getSession

```
public HttpSession getSession(String sessionId);
```

Returns the session associated with a particular session id. It should always return null.

getIds

```
public Enumeration getIds();
```

Returns an enumeration of all of the session ids in this context. This method should always return an empty enumeration.

Class Cookie

Definition

```
public class Cookie implements Cloneable
```

This class represents a cookie as defined by the original cookie specification from Netscape Communications Corporation as well as the updated RFC 2109 specification.

Constructors

```
public Cookie(String name, String value);
```

Defines a cookie with an initial name-value pair. The name must be an HTTP/1.1 token value; alphanumeric strings work.

Names starting with a $ character are reserved by RFC 2109.

This method will throw an `IllegalArgumentException` if the given name is not a valid HTTP/1.1 token.

Methods

getComment

```
public String getComment();
```

Returns the comment describing the purpose of this cookie, or null if a comment has not been defined.

getDomain

```
public String getDomain();
```

Returns the domain of this cookie, or null if not defined.

getMaxAge

```
public int getMaxAge();
```

This method returns the maximum specified age of the cookie. If no maximum age was specified, this method returns -1.

getName

```
public String getName();
```

This method returns the name of the cookie.

getPath

```
public String getPath();
```

Returns the prefix of all the URL paths for which this cookie is valid, or null if not defined.

getSecure

```
public boolean getSecure();
```

Returns true if this cookie is only to be transmitted via secure channels, false otherwise.

getValue

```
public String getValue();
```

This method returns the value of the cookie.

getVersion

```
public int getVersion();
```

Returns the version of the cookie. Version 1 compiles with RFC 2109. Version 0 indicates that the cookie complies with the original Netscape cookie specification. Newly constructed cookies use version 0 by default to maximize interoperability.

setComment

```
public void setComment(String purpose);
```

If a user agent presents this cookie to a user, the cookie's purpose will be described by this comment. Version 0 cookies do not support this attribute.

setDomain

```
public void setDomain(String pattern);
```

This method sets the domain attribute of the cookie. This attribute defines which hosts the cookie should be presented to by the client. A domain begins with a dot (`.foo.com`) and means that hosts in that DNS zone (`www.foo.com` but not `a.b.foo.com`) should see the cookie. By default, cookies are only returned to the host which saved them.

See RFC 2109 for a more complete discussion of domains and cookies.

setMaxAge

```
public void setMaxAge(int expiry);
```

This method sets the maximum age of the cookie. The cookie will expire after the given number of seconds have passed. Negative values will ensure that the cookie will not persist on the client. A zero value causes the cookie to be deleted from the client.

setPath

```
public void setPath(String uri);
```

This method sets the path attribute of the cookie. The client should only return cookies to paths that begin with the given path string.

setSecure

```
public void setSecure(boolean flag);
```

Indicates to the user agent that this cookie should only be sent via secure channels (such as HTTPS). This should only be set when the cookie's originating server used a secure protocol to set the cookie's value.

setValue

```
public void setValue(String newValue);
```

Sets the value of the cookie. BASE64 encoding is suggested for use with binary values.

Version 0 cookies should not use whitespace, brackets, parentheses, the equals sign, commas, double quotes, slashes, question marks, the @ character, colons or semicolons. Empty values may not behave the same way on all browsers.

setVersion

```
public void setVersion(int v);
```

Sets the version of the cookie format used with the cookie is written to the client. Since the IETF standards are still being finalized, consider version 1 as experimental.

Class HttpServlet

Definition

```
public class HttpServlet extends GenericServlet implements
Serializable
```

This class is an abstract class that simplifies the process of writing HTTP servlets. It extends the `GenericServlet` base class and provides a framework for handling the HTTP protocol. A servlet writer can subclass this class and provide an implementation for any method. This allows a convenient base class from which to build HTTP servlets.

The service method provided in this class supports standard HTTP methods such as GET and POST by dispatching them to appropriate methods such as `doGet` and `doPost`.

Methods

doDelete

```
protected void doDelete(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the service method of this class to handle an HTTP DELETE operation. This operation allows a client to request that an URL be removed from the server. Operations requested through DELETE can have side effects, for which users may be held accountable.

The default implementation of this method returns an HTTP BAD_REQUEST error. A servlet must explicitly implement this method in order to handle a DELETE method request.

doGet

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the `service` method of this class to handle an HTTP GET operation. This operation allows the client to simply "get" a resource from an HTTP server. Users that override this method automatically allow support for the HEAD method.

The GET operation is expected to be safe without any side effects for which users might be held responsible. For example, most form queries have no side effects. Requests intended to change stored data should use some other HTTP method. This operation is also expected to be repeated safely.

The default implementation of this method returns an HTTP BAD_REQUEST error.

doHead

```
protected void doHead(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the `service` method of this class to handle an HTTP HEAD operation. By default, this is done in terms of a unconditional GET method but returning no data to the client. Only the headers, including content length, are returned.

As with the GET operation, this operation should be safe (without side effects) and repeatable.

The default implementation of this method automatically handles the HTTP HEAD operation and does not need to be implemented by a subclass.

doOptions

```
protected void doOptions(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the `service` method of this class to handle an HTTP OPTIONS operation. This operation automatically determines what HTTP methods are supported. For example, if a servlet writer subclasses `HttpServlet` and overrides the `doGet` method, then `doOptions` returns the following header:

```
Allow: GET,HEAD,TRACE,OPTIONS
```

You do not ordinarily need to override the `doOptions` method.

doPost

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the service method of this class to handle an HTTP POST operation. This operation includes data in the request body that should be acted upon by the servlet.

Operations requested through POST can have side effects for which the user can be held accountable. Specific examples include updating store data or buying items online.

The default implementation of this method returns an HTTP BAD_REQUEST error. When you develop servlets, you must explicitly implement this method in the `HttpServlet` subclass in order to support POST operations.

doPut

```
protected void doPut(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the `service` method of this class to handle an HTTP PUT operation. This operation is analogous to sending a file via FTP.

Operations requested through PUT can have side effects for which the user can be held accountable. Specific examples include updating store data or buying items online.

The default implementation of this method returns a HTTP BAD_REQUEST error. The servlet programmer must explicitly implement this method in a `HttpServlet` subclass in order to support POST operations.

doTrace

```
protected void doTrace(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
```

Called by the `service` method of this class to handle a HTTP TRACE operation. The default implementation of this method causes a response with a message containing all of the headers sent in the trace request.

When you develop servlets, you override this method in most cases.

getLastModified

```
protected long getLastModified(HttpServletRequest request);
```

Returns the time the requested entity was last modified. Implementations supporting the GET request should override this method to provide an accurate object modification time. This helps browser and proxy caches work more effectively reducing the load on server and network resources. The return value is milliseconds since January 1, 1970, 00:00:00GMT.

The default implementation returns a negative number indicating that the modification time is unknown and should not be used for conditional GET operations.

service

```
protected void service(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException;
public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException;
```

This is an HTTP-specific version of the servlet `service` method that dispatches requests to the methods in this class which support them.

When you develop servlets, you do not usually override this method.

Class HttpSessionBindingEvent

Definition

```
public class HttpSessionBindingEvent extends EventObject
```

This event is communicated to a `HttpSessionBindingListener` whenever the listener is bound to or unbound from a `HttpSession`. This action may be the result of a session expiring or being invalidated.

The events source is the `HttpSession`. Binding occurs with a call to `HttpSession.putValue`, unbinding occurs with a call to `HttpSession.removeValue`.

Constructors

```
public HttpSessionBindingEvent(HttpSession session, String name);
```

Constructs a new `HttpSessionBindingEvent` with the session that is originating this event and the name to which the object is being bound or unbound.

Methods

getName

```
public String getName();
```

Returns the name to which the object is being bound or from which the object is being unbound.

getSession

```
public HttpSession getSession();
```

Returns the session to which the object is being bound or from which the object is being unbound.

Class HttpUtils

Definition

```
public class HttpUtils
```

A collection of static utility methods useful to HTTP servlets.

Methods

getRequestURL

```
public static StringBuffer getRequestURL(HttpServletRequest
    request);
```

Reconstructs the URL used by the client to make the given request on the server. This method accounts for difference in scheme (such as `http`, `https`) and `ports`, but does not attempt to include query parameters.

This method returns a `StringBuffer` instead of a `String` so that the URL can be modified efficiently by the servlet programmer.

parsePostData

```
public static Hashtable parsePostData(int len,
    ServletInputstream in);
```

Parses a stream which contains data of the MIME type `application/x-www-form-urlencoded` and builds a hash table of key-value pairs where the keys are strings and the values are arrays of Strings. A key can appear one or more times in the POST data. Each time a key appears, its corresponding value is inserted into its string array in the hash table.

The data read from the POST data is URL decoded. + characters are converted into spaces and characters sent in hexadecimal notation (%xx) are converted back into characters.

This method will throw an `IllegalArgumentException` if the POST data is invalid.

parseQueryString

```
public static Hashtable parseQueryString(String s);
```

Parses a query string and builds a hash table of key-value pairs where the values are arrays of Strings. A key can appear one or more times in a query string. Each time a key appears, its corresponding value is inserted into its string array in the hash table.

The data read from the query string is URL decoded. + characters are converted into spaces and characters sent in hexadecimal notation (%xx) are converted back into characters.

This method will throw an `IllegalArgumentException` if the query string is invalid.

Glossary

bytecode Machine-independent code generated by the Java compiler and executed by the Java interpreter.

cookie Data created by a Web server that is stored on the user's computer, providing a method for the Web site to keep track of a user's preferences and store them on the user's own hard disk.

HTTP HyperText Transfer Protocol. A request-response protocol used to connect to servers on the World Wide Web and transmit HTML pages to client browsers.

input stream object An object, defined by the `ServletInputStream` class, that the servlet uses to read requests from a client.

mapping A pairing of a servlet instance with an URL to which the servlet returns data, for example, `HelloServlet` with `/hello/index.html`.

output stream object An object, defined by the `ServletOutputStream` class, that a servlet uses to return data to the client.

request dispatcher object An object defined by the `RequestDispatcher` interface that receives requests from the client and sends them to any resource (such as a servlet, CGI script, HTML file, or JSP file) available on the Web server.

sandboxed servlet A servlet that runs with security restrictions.

servlet A small, platform-independent Java program without a graphical user interface that can extend the functionality of a Web server in a number of ways.

servlet configuration object An object, defined by the `ServletConfig` interface, that configures a servlet

servlet context object An object, defined by the `ServletContext` interface, that gives the servlet information about the servlet engine.

servlet engine An environment written by a Web server vendor in accordance with this specification that allows servlets to run with a particular Web server.

servlet request object An object defined by the `ServletRequest` interface and that allows a servlet to obtain data about a client request.

servlet response object An object, defined by the `ServletResponse` interface, that allows a servlet to respond

servlet runner The `sun.servlet.http.HttpServer` process in the Java Servlet Developer's Kit (JSDK), which allows servlets to run.

session tracking In a Web application, the ability to identify a series of unique requests from a client as being from the same client

SSL Secure Sockets Layer. A security protocol used on the Internet that specifies how the client browser and server exchange keys and encrypted data.

URI Uniform Resource Identifier. A definition for an Internet address that is a superset of an URL.

URL Uniform Resource Locator. The address that defines the route to a file on the World Wide Web, usually consisting of a protocol prefix, domain name, subdirectory name, and file name.

Java Software, Inc.
901 San Antonio Road
Palo Alto, CA 94303
415 960-1300

For U.S. Sales Office locations, call:
800 821-4643
In California:
800 821-4642

Australia: (02) 844 5000
Belgium: 32 2 716 7911
Canada: 416 477-6745
Finland: +358-0-525561
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Italy: 039 60551
Japan: (03) 5717-5000
Korea: 822-563-8700
Latin America: 415 688-9464
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
PRC: 861-849 2828
Singapore: 224 3388
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
UK: 0276 20444

Elsewhere in the world,
call Corporate Headquarters:
415 960-1300
Intercontinental Sales: 415 688-9000