Package javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

See:

Description

Interface Summary	
HttpServletRequest	Extends the `ServletRequest` interface to provide request information for HTTP servlets.
HttpServletResponse	Extends the `ServletResponse` interface to provide HTTP-specific functionality in sending a response.
HttpSession	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
HttpSessionActivationListener	Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.
HttpSessionAttributeListener	Interface for receiving notification events about HttpSession attribute changes.
HttpSessionBindingListener	Causes an object to be notified when it is bound to or unbound from a session.
HttpSessionContext	**Deprecated.** *As of Java(tm) Servlet API 2.1 for security reasons, with no replacement.*
HttpSessionListener	Interface for receiving notification events about HttpSession lifecycle changes.
Part	This class represents a part or form item that was received within a `multipart/form-data` POST request.

Class Summary	
Cookie	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
HttpServlet	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
HttpServletRequestWrapper	Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.

HttpServletResponseWrapper	Provides a convenient implementation of the HttpServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet.
HttpSessionBindingEvent	Events of this type are either sent to an object that implements `HttpSessionBindingListener` when it is bound or unbound from a session, or to a `HttpSessionAttributeListener` that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.
HttpUtils	**Deprecated.** *As of Java(tm) Servlet API 2.3.*

Package javax.servlet.http Description

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

Overview **Package** Class **Tree** **Deprecated** **Index** **Help**

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes

Generated on 10-February-2011 12:41

javax.servlet.http

Interface HttpServletRequest

All Superinterfaces:

ServletRequest

All Known Implementing Classes:

HttpServletRequestWrapper

```
public interface HttpServletRequest
extends ServletRequest
```

Extends the `ServletRequest` interface to provide request information for HTTP servlets.

The servlet container creates an `HttpServletRequest` object and passes it as an argument to the servlet's service methods (`doGet`, `doPost`, etc).

Author:

Various

Field Summary

static java.lang.String	**BASIC_AUTH** String identifier for Basic authentication.
static java.lang.String	**CLIENT_CERT_AUTH** String identifier for Client Certificate authentication.
static java.lang.String	**DIGEST_AUTH** String identifier for Digest authentication.
static java.lang.String	**FORM_AUTH** String identifier for Form authentication.

Method Summary

boolean	**authenticate**(HttpServletResponse response) Use the container login mechanism configured for the ServletContext to authenticate the user making this request.
java.lang.String	**getAuthType**() Returns the name of the authentication scheme used to protect the servlet.

java.lang.String	**getContextPath**() Returns the portion of the request URI that indicates the context of the request.
Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
long	**getDateHeader**(java.lang.String name) Returns the value of the specified request header as a long value that represents a Date object.
java.lang.String	**getHeader**(java.lang.String name) Returns the value of the specified request header as a String.
java.util.Enumeration<java.lang.String>	**getHeaderNames**() Returns an enumeration of all the header names this request contains.
java.util.Enumeration<java.lang.String>	**getHeaders**(java.lang.String name) Returns all the values of the specified request header as an Enumeration of String objects.
int	**getIntHeader**(java.lang.String name) Returns the value of the specified request header as an int.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
Part	**getPart**(java.lang.String name) Gets the Part with the given name.
java.util.Collection<Part>	**getParts**() Gets all the Part components of this request, provided that it is of type multipart/form-data.
java.lang.String	**getPathInfo**() Returns any extra path information associated with the URL the client sent when it made this request.
java.lang.String	**getPathTranslated**() Returns any extra path information after the servlet name but before the query string, and translates it to a real path.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
java.lang.String	**getRemoteUser**() Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.

java.lang.String	**getRequestedSessionId**() Returns the session ID specified by the client.
java.lang.String	**getRequestURI**() Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
java.lang.StringBuffer	**getRequestURL**() Reconstructs the URL the client used to make the request.
java.lang.String	**getServletPath**() Returns the part of this request's URL that calls the servlet.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.
HttpSession	**getSession**(boolean create) Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.
java.security.Principal	**getUserPrincipal**() Returns a java.security.Principal object containing the name of the current authenticated user.
boolean	**isRequestedSessionIdFromCookie**() Checks whether the requested session ID came in as a cookie.
boolean	**isRequestedSessionIdFromUrl**() **Deprecated.** *As of Version 2.1 of the Java Servlet API, use isRequestedSessionIdFromURL() instead.*
boolean	**isRequestedSessionIdFromURL**() Checks whether the requested session ID came in as part of the request URL.
boolean	**isRequestedSessionIdValid**() Checks whether the requested session ID is still valid.
boolean	**isUserInRole**(java.lang.String role) Returns a boolean indicating whether the authenticated user is included in the specified logical "role".
void	**login**(java.lang.String username, java.lang.String password) Validate the provided username and password in the password validation realm used by the web container login

	mechanism configured for the `ServletContext`.
`void`	**`logout`**`()` Establish `null` as the value returned when `getUserPrincipal`, `getRemoteUser`, and `getAuthType` is called on the request.

Methods inherited from interface javax.servlet.ServletRequest
`getAsyncContext`, `getAttribute`, `getAttributeNames`, `getCharacterEncoding`, `getContentLength`, `getContentType`, `getDispatcherType`, `getInputStream`, `getLocalAddr`, `getLocale`, `getLocales`, `getLocalName`, `getLocalPort`, `getParameter`, `getParameterMap`, `getParameterNames`, `getParameterValues`, `getProtocol`, `getReader`, `getRealPath`, `getRemoteAddr`, `getRemoteHost`, `getRemotePort`, `getRequestDispatcher`, `getScheme`, `getServerName`, `getServerPort`, `getServletContext`, `isAsyncStarted`, `isAsyncSupported`, `isSecure`, `removeAttribute`, `setAttribute`, `setCharacterEncoding`, `startAsync`, `startAsync`

Field Detail

BASIC_AUTH

```
static final java.lang.String BASIC_AUTH
```

String identifier for Basic authentication. Value "BASIC"

See Also:
Constant Field Values

FORM_AUTH

```
static final java.lang.String FORM_AUTH
```

String identifier for Form authentication. Value "FORM"

See Also:
Constant Field Values

CLIENT_CERT_AUTH

```
static final java.lang.String CLIENT_CERT_AUTH
```

String identifier for Client Certificate authentication. Value "CLIENT_CERT"

See Also:
Constant Field Values

DIGEST_AUTH

`static final java.lang.String` **`DIGEST_AUTH`**

String identifier for Digest authentication. Value "DIGEST"

See Also:

Constant Field Values

Method Detail

getAuthType

`java.lang.String` **`getAuthType`**`()`

Returns the name of the authentication scheme used to protect the servlet. All servlet containers support basic, form and client certificate authentication, and may additionally support digest authentication. If the servlet is not authenticated `null` is returned.

Same as the value of the CGI variable AUTH_TYPE.

Returns:

one of the static members BASIC_AUTH, FORM_AUTH, CLIENT_CERT_AUTH, DIGEST_AUTH (suitable for == comparison) or the container-specific string indicating the authentication scheme, or `null` if the request was not authenticated.

---

getCookies

`Cookie[]` **`getCookies`**`()`

Returns an array containing all of the `Cookie` objects the client sent with this request. This method returns `null` if no cookies were sent.

Returns:

an array of all the `Cookies` included with this request, or `null` if the request has no cookies

---

getDateHeader

`long` **`getDateHeader`**`(java.lang.String name)`

Returns the value of the specified request header as a `long` value that represents a `Date` object. Use this method with headers that contain dates, such as `If-Modified-Since`.

The date is returned as the number of milliseconds since January 1, 1970 GMT. The header name is case insensitive.

If the request did not have a header of the specified name, this method returns -1. If the header can't be converted to a date, the method throws an `IllegalArgumentException`.

Parameters:

`name` - a `String` specifying the name of the header

Returns:

a `long` value representing the date specified in the header expressed as the number of milliseconds since January 1, 1970 GMT, or -1 if the named header was not included with the request

Throws:

`IllegalArgumentException` - If the header value can't be converted to a date

---

getHeader

`java.lang.String` **`getHeader`**`(java.lang.String name)`

Returns the value of the specified request header as a `String`. If the request did not include a header of the specified name, this method returns `null`. If there are multiple headers with the same name, this method returns the first head in the request. The header name is case insensitive. You can use this method with any request header.

Parameters:

`name` - a `String` specifying the header name

Returns:

a `String` containing the value of the requested header, or `null` if the request does not have a header of that name

---

getHeaders

`java.util.Enumeration<java.lang.String>` **`getHeaders`**`(java.lang.String name)`

Returns all the values of the specified request header as an `Enumeration` of `String` objects.

Some headers, such as `Accept-Language` can be sent by clients as several headers each with a different value rather than sending the header as a comma separated list.

If the request did not include any headers of the specified name, this method returns an empty `Enumeration`. The header name is case insensitive. You can use this method with any request header.

Parameters:

`name` - a `String` specifying the header name

Returns:

an `Enumeration` containing the values of the requested header. If the request does not have any headers of that name return an empty enumeration. If the container does not allow access to header information, return null

---

getHeaderNames

`java.util.Enumeration<java.lang.String>` **`getHeaderNames`**`()`

Returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration.

Some servlet containers do not allow servlets to access headers using this method, in which case this method returns `null`

Returns:

an enumeration of all the header names sent with this request; if the request has no headers, an empty enumeration; if the servlet container does not allow servlets to use this method, `null`

---

getIntHeader

`int` **`getIntHeader`**`(java.lang.String name)`

Returns the value of the specified request header as an `int`. If the request does not have a header of the specified name, this method returns -1. If the header cannot be converted to an integer, this method throws a `NumberFormatException`.

The header name is case insensitive.

Parameters:

`name` - a `String` specifying the name of a request header

Returns:

an integer expressing the value of the request header or -1 if the request doesn't have a header of this name

Throws:

`java.lang.NumberFormatException` - If the header value can't be converted to an `int`

---

getMethod

`java.lang.String` **`getMethod`**`()`

Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT. Same as the value of the CGI variable REQUEST_METHOD.

Returns:

a `String` specifying the name of the method with which this request was made

---

getPathInfo

`java.lang.String` **`getPathInfo`**`()`

Returns any extra path information associated with the URL the client sent when it made this request. The extra path information follows the servlet path but precedes the query string and will start with a "/" character.

This method returns `null` if there was no extra path information.

Same as the value of the CGI variable PATH_INFO.

Returns:

a `String`, decoded by the web container, specifying extra path information that comes after the servlet path but before the query string in the request URL; or `null` if the URL does not have any extra path information

---

getPathTranslated

`java.lang.String` **`getPathTranslated`**`()`

Returns any extra path information after the servlet name but before the query string, and translates it to a real path. Same as the value of the CGI variable PATH_TRANSLATED.

If the URL does not have any extra path information, this method returns `null` or the servlet container cannot translate the virtual path to a real path for any reason (such as when the web application is executed from an archive). The web container does not decode this string.

Returns:

a `String` specifying the real path, or `null` if the URL does not have any extra path information

---

getContextPath

`java.lang.String` **`getContextPath`**`()`

Returns the portion of the request URI that indicates the context of the request. The context path always comes first in a request URI. The path starts with a "/" character but does not end with a "/" character. For servlets in the default (root) context, this method returns "". The container does not decode this string.

It is possible that a servlet container may match a context by more than one context path. In such cases this method will return the actual context path used by the request and it may differ from the path returned by the `ServletContext.getContextPath()` method. The context path returned by `ServletContext.getContextPath()` should be considered as the prime or preferred context path of the application.

Returns:

a `String` specifying the portion of the request URI that indicates the context of the request

See Also:

`ServletContext.getContextPath()`

---

getQueryString

`java.lang.String` **`getQueryString`**`()`

Returns the query string that is contained in the request URL after the path. This method returns `null` if the URL does not have a query string. Same as the value of the CGI variable QUERY_STRING.

Returns:

a `String` containing the query string or `null` if the URL contains no query string. The value is not decoded by the container.

---

getRemoteUser

`java.lang.String` **`getRemoteUser`**`()`

Returns the login of the user making this request, if the user has been authenticated, or `null` if the user has not been authenticated. Whether the user name is sent with each subsequent request depends on the browser and type of authentication. Same as the value of the CGI variable REMOTE_USER.

Returns:

a `String` specifying the login of the user making this request, or `null` if the user login is not known

---

isUserInRole

`boolean` **`isUserInRole`**`(java.lang.String role)`

Returns a boolean indicating whether the authenticated user is included in the specified logical "role". Roles and role membership can be defined using deployment descriptors. If the user has not been authenticated, the method returns `false`.

Parameters:

`role` - a `String` specifying the name of the role

Returns:

a `boolean` indicating whether the user making this request belongs to a given role; `false` if the user has not been authenticated

---

getUserPrincipal

`java.security.Principal` **`getUserPrincipal`**`()`

Returns a `java.security.Principal` object containing the name of the current authenticated user. If the user has not been authenticated, the method returns `null`.

Returns:

a `java.security.Principal` containing the name of the user making this request; `null` if the user has not been authenticated

---

getRequestedSessionId

`java.lang.String` **`getRequestedSessionId`**`()`

Returns the session ID specified by the client. This may not be the same as the ID of the current valid session for this request. If the client did not specify a session ID, this method returns `null`.

Returns:

a `String` specifying the session ID, or `null` if the request did not specify a session ID

See Also:

isRequestedSessionIdValid()

---

getRequestURI

`java.lang.String` **`getRequestURI`**`()`

Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request. The web container does not decode this String. For example:

First line of HTTP request	Returned Value
POST /some/path.html HTTP/1.1	/some/path.html
GET http://foo.bar/a.html HTTP/1.0	/a.html
HEAD /xyz?a=b HTTP/1.1	/xyz

To reconstruct an URL with a scheme and host, use `HttpUtils#getRequestURL`.

Returns:

a `String` containing the part of the URL from the protocol name up to the query string

See Also:

`HttpUtils#getRequestURL`

---

getRequestURL

`java.lang.StringBuffer` **`getRequestURL`**`()`

Reconstructs the URL the client used to make the request. The returned URL contains a protocol, server name, port number, and server path, but it does not include query string parameters.

If this request has been forwarded using `RequestDispatcher.forward(javax.servlet.ServletRequest, javax.servlet.ServletResponse)`, the server path in the reconstructed URL must reflect the path used to obtain the RequestDispatcher, and not the server path specified by the client.

Because this method returns a `StringBuffer`, not a string, you can modify the URL easily, for example, to append query parameters.

This method is useful for creating redirect messages and for reporting errors.

Returns:

a `StringBuffer` object containing the reconstructed URL

---

getServletPath

`java.lang.String` **`getServletPath`**`()`

Returns the part of this request's URL that calls the servlet. This path starts with a "/" character and includes either the servlet name or a path to the servlet, but does not include any extra path information or a query string. Same as the value of the CGI variable SCRIPT_NAME.

This method will return an empty string ("") if the servlet used to process this request was matched using the "/*" pattern.

Returns:

a `String` containing the name or path of the servlet being called, as specified in the request URL, decoded, or an empty string if the servlet used to process the request is matched using the "/*" pattern.

---

getSession

`HttpSession` **`getSession`**`(boolean create)`

Returns the current `HttpSession` associated with this request or, if there is no current session and `create` is true, returns a new session.

If `create` is `false` and the request has no valid `HttpSession`, this method returns `null`.

To make sure the session is properly maintained, you must call this method before the response is committed. If the container is using cookies to maintain session integrity and is asked to create a new session when the response is committed, an IllegalStateException is thrown.

Parameters:

`create` - `true` to create a new session for this request if necessary; `false` to return `null` if there's no current session

Returns:

the `HttpSession` associated with this request or `null` if `create` is `false` and the request has no valid session

See Also:

`getSession()`

---

getSession

`HttpSession` **`getSession`**`()`

Returns the current session associated with this request, or if the request does not have a session, creates one.

Returns:

the `HttpSession` associated with this request

See Also:

`getSession(boolean)`

---

isRequestedSessionIdValid

`boolean` **`isRequestedSessionIdValid`**`()`

Checks whether the requested session ID is still valid.

If the client did not specify any session ID, this method returns `false`.

Returns:

`true` if this request has an id for a valid session in the current session context; `false` otherwise

See Also:

`getRequestedSessionId()`, `getSession(boolean)`, `HttpSessionContext`

---

isRequestedSessionIdFromCookie

```
boolean isRequestedSessionIdFromCookie()
```

Checks whether the requested session ID came in as a cookie.

Returns:

`true` if the session ID came in as a cookie; otherwise, `false`

See Also:

`getSession(boolean)`

---

isRequestedSessionIdFromURL

```
boolean isRequestedSessionIdFromURL()
```

Checks whether the requested session ID came in as part of the request URL.

Returns:

`true` if the session ID came in as part of a URL; otherwise, `false`

See Also:

`getSession(boolean)`

---

isRequestedSessionIdFromUrl

```
boolean isRequestedSessionIdFromUrl()
```

Deprecated. *As of Version 2.1 of the Java Servlet API, use `isRequestedSessionIdFromURL()` instead.*

---

authenticate

```
boolean authenticate(HttpServletResponse response)
                     throws java.io.IOException,
                            ServletException
```

Use the container login mechanism configured for the `ServletContext` to authenticate the user making this request.

This method may modify and commit the argument `HttpServletResponse`.

Parameters:

`response` - The `HttpServletResponse` associated with this `HttpServletRequest`

Returns:

`true` when non-null values were or have been established as the values returned by `getUserPrincipal`, `getRemoteUser`, and `getAuthType`. Return `false` if authentication is incomplete and the underlying login mechanism has committed, in the response, the message (e.g., challenge) and HTTP status code to be returned to the user.

Throws:

`java.io.IOException` - if an input or output error occurred while reading from this request or writing to the given response

`IllegalStateException` - if the login mechanism attempted to modify the response and it was already committed

`ServletException` - if the authentication failed and the caller is responsible for handling the error (i.e., the underlying login mechanism did NOT establish the message and HTTP status code to be returned to the user)

Since:

Servlet 3.0

---

login

```
void login(java.lang.String username,
           java.lang.String password)
           throws ServletException
```

Validate the provided username and password in the password validation realm used by the web container login mechanism configured for the `ServletContext`.

This method returns without throwing a `ServletException` when the login mechanism configured for the `ServletContext` supports username password validation, and when, at the time of the call to login, the identity of the caller of the request had not been established (i.e, all of `getUserPrincipal`, `getRemoteUser`, and `getAuthType` return null), and when validation of the provided credentials is successful. Otherwise, this method throws a `ServletException` as described below.

When this method returns without throwing an exception, it must have established non-null values as the values returned by `getUserPrincipal`, `getRemoteUser`, and `getAuthType`.

Parameters:

`username` - The `String` value corresponding to the login identifier of the user.

`password` - The password `String` corresponding to the identified user.

Throws:

`ServletException` - if the configured login mechanism does not support username password authentication, or if a non-null caller identity had already been established (prior to the call to login), or if validation of the provided username and password fails.

Since:

Servlet 3.0

---

logout

```
void logout()
           throws ServletException
```

Establish `null` as the value returned when `getUserPrincipal`, `getRemoteUser`, and

`getAuthType` is called on the request.

Throws:

`ServletException` - if logout fails

Since:

Servlet 3.0

---

getParts

```
java.util.Collection<Part> getParts()
                                  throws java.io.IOException,
                                         ServletException
```

Gets all the `Part` components of this request, provided that it is of type `multipart/form-data`.

If this request is of type `multipart/form-data`, but does not contain any Part components, the returned `Collection` will be empty.

Any changes to the returned `Collection` must not affect this `HttpServletRequest`.

Returns:

a (possibly empty) `Collection` of the Part components of this request

Throws:

`java.io.IOException` - if an I/O error occurred during the retrieval of the `Part` components of this request

`ServletException` - if this request is not of type `multipart/form-data`

`IllegalStateException` - if the request body is larger than `maxRequestSize`, or any Part in the request is larger than `maxFileSize`

Since:

Servlet 3.0

See Also:

`MultipartConfig.maxFileSize()`, `MultipartConfig.maxRequestSize()`

---

getPart

```
Part getPart(java.lang.String name)
             throws java.io.IOException,
                    ServletException
```

Gets the `Part` with the given name.

Parameters:

`name` - the name of the requested Part

Returns:

The Part with the given name, or `null` if this request is of type `multipart/form-data`, but does not contain the requested Part

Throws:

`java.io.IOException` - if an I/O error occurred during the retrieval of the requested Part

`ServletException` - if this request is not of type `multipart/form-data`

`IllegalStateException` - if the request body is larger than `maxRequestSize`, or any Part in the request is larger than `maxFileSize`

Since:

Servlet 3.0

See Also:

`MultipartConfig.maxFileSize()`, `MultipartConfig.maxRequestSize()`

---

javax.servlet.http

Interface HttpServletResponse

All Superinterfaces:

ServletResponse

All Known Implementing Classes:

HttpServletResponseWrapper

```
public interface HttpServletResponse
extends ServletResponse
```

Extends the `ServletResponse` interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies.

The servlet container creates an `HttpServletResponse` object and passes it as an argument to the servlet's service methods (`doGet`, `doPost`, etc).

Author:

Various

See Also:

`ServletResponse`

Field Summary

static int	**SC_ACCEPTED** Status code (202) indicating that a request was accepted for processing, but was not completed.
static int	**SC_BAD_GATEWAY** Status code (502) indicating that the HTTP server received an invalid response from a server it consulted when acting as a proxy or gateway.
static int	**SC_BAD_REQUEST** Status code (400) indicating the request sent by the client was syntactically incorrect.
static int	**SC_CONFLICT** Status code (409) indicating that the request could not be completed due to a conflict with the current state of the resource.
static int	**SC_CONTINUE**

	Status code (100) indicating the client can continue.
static int	**SC_CREATED** Status code (201) indicating the request succeeded and created a new resource on the server.
static int	**SC_EXPECTATION_FAILED** Status code (417) indicating that the server could not meet the expectation given in the Expect request header.
static int	**SC_FORBIDDEN** Status code (403) indicating the server understood the request but refused to fulfill it.
static int	**SC_FOUND** Status code (302) indicating that the resource reside temporarily under a different URI.
static int	**SC_GATEWAY_TIMEOUT** Status code (504) indicating that the server did not receive a timely response from the upstream server while acting as a gateway or proxy.
static int	**SC_GONE** Status code (410) indicating that the resource is no longer available at the server and no forwarding address is known.
static int	**SC_HTTP_VERSION_NOT_SUPPORTED** Status code (505) indicating that the server does not support or refuses to support the HTTP protocol version that was used in the request message.
static int	**SC_INTERNAL_SERVER_ERROR** Status code (500) indicating an error inside the HTTP server which prevented it from fulfilling the request.
static int	**SC_LENGTH_REQUIRED** Status code (411) indicating that the request cannot be handled without a defined *Content-Length*.
static int	**SC_METHOD_NOT_ALLOWED** Status code (405) indicating that the method specified in the *Request-Line* is not allowed for the resource identified by the *Request-URI*.
static int	**SC_MOVED_PERMANENTLY** Status code (301) indicating that the resource has permanently moved to a new location, and that future references should use a new URI with their requests.
static int	**SC_MOVED_TEMPORARILY** Status code (302) indicating that the resource has temporarily moved to another location, but that future references should still use the original URI to access the resource.
static int	**SC_MULTIPLE_CHOICES** Status code (300) indicating that the requested resource corresponds to any one of a set of representations, each with its own specific location.
static int	**SC_NO_CONTENT** Status code (204) indicating that the request succeeded but that there was no new

	information to return.
static int	**SC_NON_AUTHORITATIVE_INFORMATION** Status code (203) indicating that the meta information presented by the client did not originate from the server.
static int	**SC_NOT_ACCEPTABLE** Status code (406) indicating that the resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
static int	**SC_NOT_FOUND** Status code (404) indicating that the requested resource is not available.
static int	**SC_NOT_IMPLEMENTED** Status code (501) indicating the HTTP server does not support the functionality needed to fulfill the request.
static int	**SC_NOT_MODIFIED** Status code (304) indicating that a conditional GET operation found that the resource was available and not modified.
static int	**SC_OK** Status code (200) indicating the request succeeded normally.
static int	**SC_PARTIAL_CONTENT** Status code (206) indicating that the server has fulfilled the partial GET request for the resource.
static int	**SC_PAYMENT_REQUIRED** Status code (402) reserved for future use.
static int	**SC_PRECONDITION_FAILED** Status code (412) indicating that the precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
static int	**SC_PROXY_AUTHENTICATION_REQUIRED** Status code (407) indicating that the client *MUST* first authenticate itself with the proxy.
static int	**SC_REQUEST_ENTITY_TOO_LARGE** Status code (413) indicating that the server is refusing to process the request because the request entity is larger than the server is willing or able to process.
static int	**SC_REQUEST_TIMEOUT** Status code (408) indicating that the client did not produce a request within the time that the server was prepared to wait.
static int	**SC_REQUEST_URI_TOO_LONG** Status code (414) indicating that the server is refusing to service the request because the *Request-URI* is longer than the server is willing to interpret.
static int	**SC_REQUESTED_RANGE_NOT_SATISFIABLE** Status code (416) indicating that the server cannot serve the requested byte range.
static int	

	SC_RESET_CONTENT Status code (205) indicating that the agent *SHOULD* reset the document view which caused the request to be sent.
static int	**SC_SEE_OTHER** Status code (303) indicating that the response to the request can be found under a different URI.
static int	**SC_SERVICE_UNAVAILABLE** Status code (503) indicating that the HTTP server is temporarily overloaded, and unable to handle the request.
static int	**SC_SWITCHING_PROTOCOLS** Status code (101) indicating the server is switching protocols according to Upgrade header.
static int	**SC_TEMPORARY_REDIRECT** Status code (307) indicating that the requested resource resides temporarily under a different URI.
static int	**SC_UNAUTHORIZED** Status code (401) indicating that the request requires HTTP authentication.
static int	**SC_UNSUPPORTED_MEDIA_TYPE** Status code (415) indicating that the server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
static int	**SC_USE_PROXY** Status code (305) indicating that the requested resource *MUST* be accessed through the proxy given by the `Location` field.

Method Summary

void	**addCookie**(Cookie cookie) Adds the specified cookie to the response.
void	**addDateHeader**(java.lang.String name, long date) Adds a response header with the given name and date-value.
void	**addHeader**(java.lang.String name, java.lang.String value) Adds a response header with the given name and value.
void	**addIntHeader**(java.lang.String name, int value) Adds a response header with the given name and integer value.
boolean	**containsHeader**(java.lang.String name) Returns a boolean indicating whether the named response

	header has already been set.
java.lang.String	**encodeRedirectUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeRedirectURL(String url) instead*
java.lang.String	**encodeRedirectURL**(java.lang.String url) Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
java.lang.String	**encodeUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeURL(String url) instead*
java.lang.String	**encodeURL**(java.lang.String url) Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
java.lang.String	**getHeader**(java.lang.String name) Gets the value of the response header with the given name.
java.util.Collection<java.lang.String>	**getHeaderNames**() Gets the names of the headers of this response.
java.util.Collection<java.lang.String>	**getHeaders**(java.lang.String name) Gets the values of the response header with the given name.
int	**getStatus**() Gets the current status code of this response.
void	**sendError**(int sc) Sends an error response to the client using the specified status code and clears the buffer.
void	**sendError**(int sc, java.lang.String msg) Sends an error response to the client using the specified status and clears the buffer.
void	**sendRedirect**(java.lang.String location) Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
void	**setDateHeader**(java.lang.String name, long date) Sets a response header with the given name and date-value.
void	**setHeader**(java.lang.String name, java.lang.String value) Sets a response header with the given name and value.
void	**setIntHeader**(java.lang.String name, int value)

	Sets a response header with the given name and integer value.
void	**setStatus**(int sc) Sets the status code for this response.
void	**setStatus**(int sc, java.lang.String sm) **Deprecated.** *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use `setStatus(int)`, to send an error with a description use `sendError(int, String)`. Sets the status code and message for this response.*

Methods inherited from interface javax.servlet.ServletResponse
flushBuffer, getBufferSize, getCharacterEncoding, getContentType, getLocale, getOutputStream, getWriter, isCommitted, reset, resetBuffer, setBufferSize, setCharacterEncoding, setContentLength, setContentType, setLocale

Field Detail

SC_CONTINUE

```
static final int SC_CONTINUE
```

Status code (100) indicating the client can continue.

See Also:
Constant Field Values

SC_SWITCHING_PROTOCOLS

```
static final int SC_SWITCHING_PROTOCOLS
```

Status code (101) indicating the server is switching protocols according to Upgrade header.

See Also:
Constant Field Values

SC_OK

```
static final int SC_OK
```

Status code (200) indicating the request succeeded normally.

See Also:

Constant Field Values

SC_CREATED

```
static final int SC_CREATED
```

Status code (201) indicating the request succeeded and created a new resource on the server.

See Also:

Constant Field Values

SC_ACCEPTED

```
static final int SC_ACCEPTED
```

Status code (202) indicating that a request was accepted for processing, but was not completed.

See Also:

Constant Field Values

SC_NON_AUTHORITATIVE_INFORMATION

```
static final int SC_NON_AUTHORITATIVE_INFORMATION
```

Status code (203) indicating that the meta information presented by the client did not originate from the server.

See Also:

Constant Field Values

SC_NO_CONTENT

```
static final int SC_NO_CONTENT
```

Status code (204) indicating that the request succeeded but that there was no new information to return.

See Also:

Constant Field Values

SC_RESET_CONTENT

```
static final int SC_RESET_CONTENT
```

Status code (205) indicating that the agent *SHOULD* reset the document view which caused the request to be sent.

See Also:

Constant Field Values

SC_PARTIAL_CONTENT

```
static final int SC_PARTIAL_CONTENT
```

Status code (206) indicating that the server has fulfilled the partial GET request for the resource.

See Also:

Constant Field Values

SC_MULTIPLE_CHOICES

```
static final int SC_MULTIPLE_CHOICES
```

Status code (300) indicating that the requested resource corresponds to any one of a set of representations, each with its own specific location.

See Also:

Constant Field Values

SC_MOVED_PERMANENTLY

```
static final int SC_MOVED_PERMANENTLY
```

Status code (301) indicating that the resource has permanently moved to a new location, and that future references should use a new URI with their requests.

See Also:

Constant Field Values

SC_MOVED_TEMPORARILY

```
static final int SC_MOVED_TEMPORARILY
```

Status code (302) indicating that the resource has temporarily moved to another location, but that future references should still use the original URI to access the resource. This definition is being retained for backwards compatibility. SC_FOUND is now the preferred definition.

See Also:

Constant Field Values

SC_FOUND

```
static final int SC_FOUND
```

Status code (302) indicating that the resource reside temporarily under a different URI. Since the redirection might be altered on occasion, the client should continue to use the Request-URI for future requests.(HTTP/1.1) To represent the status code (302), it is recommended to use this variable.

See Also:

Constant Field Values

SC_SEE_OTHER

```
static final int SC_SEE_OTHER
```

Status code (303) indicating that the response to the request can be found under a different URI.

See Also:

Constant Field Values

SC_NOT_MODIFIED

```
static final int SC_NOT_MODIFIED
```

Status code (304) indicating that a conditional GET operation found that the resource was available and not modified.

See Also:

Constant Field Values

SC_USE_PROXY

```
static final int SC_USE_PROXY
```

Status code (305) indicating that the requested resource *MUST* be accessed through the proxy given by the `Location` field.

See Also:

Constant Field Values

SC_TEMPORARY_REDIRECT

```
static final int SC_TEMPORARY_REDIRECT
```

Status code (307) indicating that the requested resource resides temporarily under a different URI. The temporary URI *SHOULD* be given by the `Location` field in the response.

See Also:

Constant Field Values

SC_BAD_REQUEST

```
static final int SC_BAD_REQUEST
```

Status code (400) indicating the request sent by the client was syntactically incorrect.

See Also:

Constant Field Values

SC_UNAUTHORIZED

```
static final int SC_UNAUTHORIZED
```

Status code (401) indicating that the request requires HTTP authentication.

See Also:

Constant Field Values

SC_PAYMENT_REQUIRED

```
static final int SC_PAYMENT_REQUIRED
```

Status code (402) reserved for future use.

See Also:

Constant Field Values

SC_FORBIDDEN

```
static final int SC_FORBIDDEN
```

Status code (403) indicating the server understood the request but refused to fulfill it.

See Also:

Constant Field Values

SC_NOT_FOUND

```
static final int SC_NOT_FOUND
```

Status code (404) indicating that the requested resource is not available.

See Also:
Constant Field Values

SC_METHOD_NOT_ALLOWED

```
static final int SC_METHOD_NOT_ALLOWED
```

Status code (405) indicating that the method specified in the *Request-Line* is not allowed for the resource identified by the *Request-URI*.

See Also:
Constant Field Values

SC_NOT_ACCEPTABLE

```
static final int SC_NOT_ACCEPTABLE
```

Status code (406) indicating that the resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

See Also:
Constant Field Values

SC_PROXY_AUTHENTICATION_REQUIRED

```
static final int SC_PROXY_AUTHENTICATION_REQUIRED
```

Status code (407) indicating that the client *MUST* first authenticate itself with the proxy.

See Also:
Constant Field Values

SC_REQUEST_TIMEOUT

```
static final int SC_REQUEST_TIMEOUT
```

Status code (408) indicating that the client did not produce a request within the time that the server was

prepared to wait.

See Also:

Constant Field Values

SC_CONFLICT

```
static final int SC_CONFLICT
```

Status code (409) indicating that the request could not be completed due to a conflict with the current state of the resource.

See Also:

Constant Field Values

SC_GONE

```
static final int SC_GONE
```

Status code (410) indicating that the resource is no longer available at the server and no forwarding address is known. This condition *SHOULD* be considered permanent.

See Also:

Constant Field Values

SC_LENGTH_REQUIRED

```
static final int SC_LENGTH_REQUIRED
```

Status code (411) indicating that the request cannot be handled without a defined `Content-Length`.

See Also:

Constant Field Values

SC_PRECONDITION_FAILED

```
static final int SC_PRECONDITION_FAILED
```

Status code (412) indicating that the precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.

See Also:

Constant Field Values

SC_REQUEST_ENTITY_TOO_LARGE

```
static final int SC_REQUEST_ENTITY_TOO_LARGE
```

Status code (413) indicating that the server is refusing to process the request because the request entity is larger than the server is willing or able to process.

See Also:

Constant Field Values

SC_REQUEST_URI_TOO_LONG

```
static final int SC_REQUEST_URI_TOO_LONG
```

Status code (414) indicating that the server is refusing to service the request because the *Request-URI* is longer than the server is willing to interpret.

See Also:

Constant Field Values

SC_UNSUPPORTED_MEDIA_TYPE

```
static final int SC_UNSUPPORTED_MEDIA_TYPE
```

Status code (415) indicating that the server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

See Also:

Constant Field Values

SC_REQUESTED_RANGE_NOT_SATISFIABLE

```
static final int SC_REQUESTED_RANGE_NOT_SATISFIABLE
```

Status code (416) indicating that the server cannot serve the requested byte range.

See Also:

Constant Field Values

SC_EXPECTATION_FAILED

```
static final int SC_EXPECTATION_FAILED
```

Status code (417) indicating that the server could not meet the expectation given in the Expect request

header.

See Also:

Constant Field Values

SC_INTERNAL_SERVER_ERROR

```
static final int SC_INTERNAL_SERVER_ERROR
```

Status code (500) indicating an error inside the HTTP server which prevented it from fulfilling the request.

See Also:

Constant Field Values

SC_NOT_IMPLEMENTED

```
static final int SC_NOT_IMPLEMENTED
```

Status code (501) indicating the HTTP server does not support the functionality needed to fulfill the request.

See Also:

Constant Field Values

SC_BAD_GATEWAY

```
static final int SC_BAD_GATEWAY
```

Status code (502) indicating that the HTTP server received an invalid response from a server it consulted when acting as a proxy or gateway.

See Also:

Constant Field Values

SC_SERVICE_UNAVAILABLE

```
static final int SC_SERVICE_UNAVAILABLE
```

Status code (503) indicating that the HTTP server is temporarily overloaded, and unable to handle the request.

See Also:

Constant Field Values

SC_GATEWAY_TIMEOUT

```
static final int SC_GATEWAY_TIMEOUT
```

Status code (504) indicating that the server did not receive a timely response from the upstream server while acting as a gateway or proxy.

See Also:

Constant Field Values

SC_HTTP_VERSION_NOT_SUPPORTED

```
static final int SC_HTTP_VERSION_NOT_SUPPORTED
```

Status code (505) indicating that the server does not support or refuses to support the HTTP protocol version that was used in the request message.

See Also:

Constant Field Values

Method Detail

addCookie

```
void addCookie(Cookie cookie)
```

Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.

Parameters:

`cookie` - the Cookie to return to the client

---

containsHeader

```
boolean containsHeader(java.lang.String name)
```

Returns a boolean indicating whether the named response header has already been set.

Parameters:

`name` - the header name

Returns:

`true` if the named response header has already been set; `false` otherwise

---

encodeURL

`java.lang.String` **`encodeURL`**`(java.lang.String url)`

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged. The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.

For robust session tracking, all URLs emitted by a servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers which do not support cookies.

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

encodeRedirectURL

`java.lang.String` **`encodeRedirectURL`**`(java.lang.String url)`

Encodes the specified URL for use in the `sendRedirect` method or, if encoding is not needed, returns the URL unchanged. The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL. Because the rules for making this determination can differ from those used to decide whether to encode a normal link, this method is separated from the `encodeURL` method.

All URLs sent to the `HttpServletResponse.sendRedirect` method should be run through this method. Otherwise, URL rewriting cannot be used with browsers which do not support cookies.

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

See Also:

`sendRedirect(java.lang.String)`, `encodeUrl(java.lang.String)`

---

encodeUrl

`java.lang.String` **`encodeUrl`**`(java.lang.String url)`

Deprecated. *As of version 2.1, use encodeURL(String url) instead*

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

encodeRedirectUrl

```
java.lang.String encodeRedirectUrl(java.lang.String url)
```

Deprecated. *As of version 2.1, use encodeRedirectURL(String url) instead*

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

sendError

```
void sendError(int sc,
               java.lang.String msg)
               throws java.io.IOException
```

Sends an error response to the client using the specified status and clears the buffer. The server defaults to creating the response to look like an HTML-formatted server error page containing the specified message, setting the content type to "text/html". The server will preserve cookies and may clear or update any headers needed to serve the error page as a valid response. If an error-page declaration has been made for the web application corresponding to the status code passed in, it will be served back in preference to the suggested msg parameter and the msg parameter will be ignored.

If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

Parameters:

`sc` - the error status code

`msg` - the descriptive message

Throws:

`java.io.IOException` - If an input or output exception occurs

`IllegalStateException` - If the response was committed

---

sendError

```
void sendError(int sc)
               throws java.io.IOException
```

Sends an error response to the client using the specified status code and clears the buffer. The server will preserve cookies and may clear or update any headers needed to serve the error page as a valid response. If an error-page declaration has been made for the web application corresponding to the status code passed in, it will be served back the error page

If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

Parameters:

`sc` - the error status code

Throws:

`java.io.IOException` - If an input or output exception occurs

`IllegalStateException` - If the response was committed before this method call

---

sendRedirect

```
void sendRedirect(java.lang.String location)
                  throws java.io.IOException
```

Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer. The buffer will be replaced with the data set by this method. Calling this method sets the status code to `SC_FOUND` 302 (Found). This method can accept relative URLs;the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URI. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root.

If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

Parameters:

`location` - the redirect location URL

Throws:

`java.io.IOException` - If an input or output exception occurs

`IllegalStateException` - If the response was committed or if a partial URL is given and cannot be converted into a valid URL

---

setDateHeader

```
void setDateHeader(java.lang.String name,
                   long date)
```

Sets a response header with the given name and date-value. The date is specified in terms of milliseconds since the epoch. If the header had already been set, the new value overwrites the previous one. The `containsHeader` method can be used to test for the presence of a header before setting its value.

Parameters:

`name` - the name of the header to set

`date` - the assigned date value

See Also:

`containsHeader(java.lang.String)`, `addDateHeader(java.lang.String, long)`

---

addDateHeader

```
void addDateHeader(java.lang.String name,
                   long date)
```

Adds a response header with the given name and date-value. The date is specified in terms of milliseconds since the epoch. This method allows response headers to have multiple values.

Parameters:

`name` - the name of the header to set

`date` - the additional date value

See Also:

`setDateHeader(java.lang.String, long)`

---

setHeader

```
void setHeader(java.lang.String name,
               java.lang.String value)
```

Sets a response header with the given name and value. If the header had already been set, the new value overwrites the previous one. The `containsHeader` method can be used to test for the presence of a header before setting its value.

Parameters:

`name` - the name of the header

`value` - the header value If it contains octet string, it should be encoded according to RFC 2047 (http://www.ietf.org/rfc/rfc2047.txt)

See Also:

`containsHeader(java.lang.String)`, `addHeader(java.lang.String, java.lang.String)`

---

addHeader

```
void addHeader(java.lang.String name,
               java.lang.String value)
```

Adds a response header with the given name and value. This method allows response headers to have multiple values.

Parameters:

`name` - the name of the header

`value` - the additional header value If it contains octet string, it should be encoded according to RFC 2047 (http://www.ietf.org/rfc/rfc2047.txt)

See Also:

`setHeader(java.lang.String, java.lang.String)`

---

setIntHeader

```
void setIntHeader(java.lang.String name,
                  int value)
```

Sets a response header with the given name and integer value. If the header had already been set, the new value overwrites the previous one. The `containsHeader` method can be used to test for the presence of a header before setting its value.

Parameters:

`name` - the name of the header

`value` - the assigned integer value

See Also:

`containsHeader(java.lang.String)`, `addIntHeader(java.lang.String, int)`

---

addIntHeader

```
void addIntHeader(java.lang.String name,
                  int value)
```

Adds a response header with the given name and integer value. This method allows response headers to have multiple values.

Parameters:

`name` - the name of the header

`value` - the assigned integer value

See Also:

`setIntHeader(java.lang.String, int)`

---

setStatus

```
void setStatus(int sc)
```

Sets the status code for this response.

This method is used to set the return status code when there is no error (for example, for the SC_OK or SC_MOVED_TEMPORARILY status codes).

If this method is used to set an error code, then the container's error page mechanism will not be triggered. If there is an error and the caller wishes to invoke an error page defined in the web application, then `sendError(int, java.lang.String)` must be used instead.

This method preserves any cookies and other response headers.

Valid status codes are those in the 2XX, 3XX, 4XX, and 5XX ranges. Other status codes are treated as container specific.

Parameters:

`sc` - the status code

See Also:

sendError(int, java.lang.String)

---

setStatus

```
void setStatus(int sc,
               java.lang.String sm)
```

Deprecated. *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use* `setStatus(int)`*, to send an error with a description use* `sendError(int, String)`*. Sets the status code and message for this response.*

Parameters:

`sc` - the status code

`sm` - the status message

---

getStatus

```
int getStatus()
```

Gets the current status code of this response.

Returns:

the current status code of this response

Since:

Servlet 3.0

---

getHeader

```
java.lang.String getHeader(java.lang.String name)
```

Gets the value of the response header with the given name.

If a response header with the given name exists and contains multiple values, the value that was added first will be returned.

This method considers only response headers set or added via setHeader(java.lang.String, java.lang.String), addHeader(java.lang.String, java.lang.String), setDateHeader(java.lang.String, long), addDateHeader(java.lang.String, long), setIntHeader(java.lang.String, int), or addIntHeader(java.lang.String, int), respectively.

Parameters:

`name` - the name of the response header whose value to return

Returns:

the value of the response header with the given name, or `null` if no header with the given name has been set on this response

Since:

Servlet 3.0

---

getHeaders

`java.util.Collection<java.lang.String>` **`getHeaders`**`(java.lang.String name)`

Gets the values of the response header with the given name.

This method considers only response headers set or added via `setHeader(java.lang.String, java.lang.String)`, `addHeader(java.lang.String, java.lang.String)`, `setDateHeader(java.lang.String, long)`, `addDateHeader(java.lang.String, long)`, `setIntHeader(java.lang.String, int)`, or `addIntHeader(java.lang.String, int)`, respectively.

Any changes to the returned `Collection` must not affect this `HttpServletResponse`.

Parameters:

`name` - the name of the response header whose values to return

Returns:

a (possibly empty) `Collection` of the values of the response header with the given name

Since:

Servlet 3.0

---

getHeaderNames

`java.util.Collection<java.lang.String>` **`getHeaderNames`**`()`

Gets the names of the headers of this response.

This method considers only response headers set or added via `setHeader(java.lang.String, java.lang.String)`, `addHeader(java.lang.String, java.lang.String)`, `setDateHeader(java.lang.String, long)`, `addDateHeader(java.lang.String, long)`, `setIntHeader(java.lang.String, int)`, or `addIntHeader(java.lang.String, int)`, respectively.

Any changes to the returned `Collection` must not affect this `HttpServletResponse`.

Returns:

a (possibly empty) `Collection` of the names of the headers of this response

Since:

Servlet 3.0

Submit a bug or feature

Generated on 10-February-2011 12:41

javax.servlet.http

Interface HttpSession

`public interface` **`HttpSession`**

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
- Bind objects to sessions, allowing user information to persist across multiple user connections

When an application stores an object in or removes an object from a session, the session checks whether the object implements `HttpSessionBindingListener`. If it does, the servlet notifies the object that it has been bound to or unbound from the session. Notifications are sent after the binding methods complete. For session that are invalidated or expire, notifications are sent after the session has been invalidated or expired.

When container migrates a session between VMs in a distributed container setting, all session attributes implementing the `HttpSessionActivationListener` interface are notified.

A servlet should be able to handle cases in which the client does not choose to join a session, such as when cookies are intentionally turned off. Until the client joins the session, `isNew` returns `true`. If the client chooses not to join the session, `getSession` will return a different session on each request, and `isNew` will always return `true`.

Session information is scoped only to the current web application (`ServletContext`), so information stored in one context will not be directly visible in another.

Author:

Various

See Also:

`HttpSessionBindingListener`, `HttpSessionContext`

Method Summary

java.lang.Object	**getAttribute**(java.lang.String name) Returns the object bound with the specified name in this session, or null if no object is bound under the name.
java.util.Enumeration<java.lang.String>	**getAttributeNames**() Returns an Enumeration of String objects containing the names of all the objects bound to this session.
long	**getCreationTime**() Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
java.lang.String	**getId**() Returns a string containing the unique identifier assigned to this session.
long	**getLastAccessedTime**() Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
int	**getMaxInactiveInterval**() Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
ServletContext	**getServletContext**() Returns the ServletContext to which this session belongs.
HttpSessionContext	**getSessionContext**() **Deprecated.** *As of Version 2.1, this method is deprecated and has no replacement. It will be removed in a future version of the Java Servlet API.*
java.lang.Object	**getValue**(java.lang.String name) **Deprecated.** *As of Version 2.2, this method is replaced by getAttribute(java.lang.String).*
java.lang.String[]	**getValueNames**() **Deprecated.** *As of Version 2.2, this method is replaced by getAttributeNames()*
void	**invalidate**() Invalidates this session then unbinds any objects bound to it.
boolean	**isNew**()

	Returns `true` if the client does not yet know about the session or if the client chooses not to join the session.
void	**putValue**(java.lang.String name, java.lang.Object value) **Deprecated.** *As of Version 2.2, this method is replaced by setAttribute(java.lang.String, java.lang.Object)*
void	**removeAttribute**(java.lang.String name) Removes the object bound with the specified name from this session.
void	**removeValue**(java.lang.String name) **Deprecated.** *As of Version 2.2, this method is replaced by removeAttribute(java.lang.String)*
void	**setAttribute**(java.lang.String name, java.lang.Object value) Binds an object to this session, using the name specified.
void	**setMaxInactiveInterval**(int interval) Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Method Detail

getCreationTime

`long` **`getCreationTime`**`()`

Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

Returns:

a `long` specifying when this session was created, expressed in milliseconds since 1/1/1970 GMT

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

getId

`java.lang.String` **`getId`**`()`

Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.

Returns:

a string specifying the identifier assigned to this session

---

getLastAccessedTime

`long` **`getLastAccessedTime`**`()`

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.

Actions that your application takes, such as getting or setting a value associated with the session, do not affect the access time.

Returns:

a `long` representing the last time the client sent a request associated with this session, expressed in milliseconds since 1/1/1970 GMT

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

getServletContext

`ServletContext` **`getServletContext`**`()`

Returns the ServletContext to which this session belongs.

Returns:

The ServletContext object for the web application

Since:

Servlet 2.3

---

setMaxInactiveInterval

`void` **`setMaxInactiveInterval`**`(int interval)`

Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

An `interval` value of zero or less indicates that the session should never timeout.

Parameters:

`interval` - An integer specifying the number of seconds

---

getMaxInactiveInterval

`int` **`getMaxInactiveInterval`**`()`

Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses. After this interval, the servlet container will invalidate the session. The maximum time interval can be set with the `setMaxInactiveInterval` method.

A return value of zero or less indicates that the session will never timeout.

Returns:

an integer specifying the number of seconds this session remains open between client requests

See Also:

`setMaxInactiveInterval(int)`

---

getSessionContext

`HttpSessionContext` **`getSessionContext`**`()`

Deprecated. *As of Version 2.1, this method is deprecated and has no replacement. It will be removed in a future version of the Java Servlet API.*

---

getAttribute

`java.lang.Object` **`getAttribute`**`(java.lang.String name)`

Returns the object bound with the specified name in this session, or `null` if no object is bound under the name.

Parameters:

`name` - a string specifying the name of the object

Returns:

the object with the specified name

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

getValue

`java.lang.Object` **`getValue`**`(java.lang.String name)`

Deprecated. *As of Version 2.2, this method is replaced by `getAttribute(java.lang.String)`.*

Parameters:

`name` - a string specifying the name of the object

Returns:

the object with the specified name

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

getAttributeNames

```
java.util.Enumeration<java.lang.String> getAttributeNames()
```

Returns an `Enumeration` of `String` objects containing the names of all the objects bound to this session.

Returns:

an `Enumeration` of `String` objects specifying the names of all the objects bound to this session

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

getValueNames

```
java.lang.String[] getValueNames()
```

Deprecated. *As of Version 2.2, this method is replaced by* *`getAttributeNames()`*

Returns:

an array of `String` objects specifying the names of all the objects bound to this session

Throws:

`IllegalStateException` - if this method is called on an invalidated session

---

setAttribute

```
void setAttribute(java.lang.String name,
                  java.lang.Object value)
```

Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.

After this method executes, and if the new object implements `HttpSessionBindingListener`, the container calls `HttpSessionBindingListener.valueBound`. The container then notifies any `HttpSessionAttributeListener`s in the web application.

If an object was already bound to this session of this name that implements `HttpSessionBindingListener`, its `HttpSessionBindingListener.valueUnbound` method is called.

If the value passed in is null, this has the same effect as calling `removeAttribute()`.

Parameters:

```
name - the name to which the object is bound; cannot be null
value - the object to be bound
```

Throws:

```
IllegalStateException - if this method is called on an invalidated session
```

---

putValue

void **putValue**(java.lang.String name,
java.lang.Object value)

Deprecated. *As of Version 2.2, this method is replaced by* *setAttribute(java.lang.String, java.lang.Object)*

Parameters:
name - the name to which the object is bound; cannot be null
value - the object to be bound; cannot be null
Throws:
IllegalStateException - if this method is called on an invalidated session

---

removeAttribute

void **removeAttribute**(java.lang.String name)

Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.

After this method executes, and if the object implements HttpSessionBindingListener, the container calls HttpSessionBindingListener.valueUnbound. The container then notifies any HttpSessionAttributeListeners in the web application.

Parameters:
name - the name of the object to remove from this session
Throws:
IllegalStateException - if this method is called on an invalidated session

---

removeValue

void **removeValue**(java.lang.String name)

Deprecated. *As of Version 2.2, this method is replaced by* *removeAttribute(java.lang.String)*

Parameters:
name - the name of the object to remove from this session
Throws:
IllegalStateException - if this method is called on an invalidated session

---

invalidate

void **invalidate**()

Invalidates this session then unbinds any objects bound to it.

Throws:
IllegalStateException - if this method is called on an already invalidated session

---

isNew

boolean **isNew**()

Returns true if the client does not yet know about the session or if the client chooses not to join the session. For example, if the server used only cookie-based sessions, and the client had disabled the use of cookies, then a session would be new on each request.

Returns:

true if the server has created a session, but the client has not yet joined

Throws:

IllegalStateException - if this method is called on an already invalidated session

---

---

javax.servlet.http

Interface HttpSessionActivationListener

All Superinterfaces:

java.util.EventListener

```
public interface HttpSessionActivationListener
extends java.util.EventListener
```

Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated. A container that migrates session between VMs or persists sessions is required to notify all attributes bound to sessions implementing HttpSessionActivationListener.

Since:

Servlet 2.3

Method Summary

void	**sessionDidActivate**(HttpSessionEvent se) Notification that the session has just been activated.
void	**sessionWillPassivate**(HttpSessionEvent se) Notification that the session is about to be passivated.

Method Detail

sessionWillPassivate

```
void sessionWillPassivate(HttpSessionEvent se)
```

Notification that the session is about to be passivated.

---

sessionDidActivate

```
void sessionDidActivate(HttpSessionEvent se)
```

Notification that the session has just been activated.

Submit a bug or feature

Generated on 10-February-2011 12:41

javax.servlet.http

Interface HttpSessionAttributeListener

All Superinterfaces:

java.util.EventListener

```
public interface HttpSessionAttributeListener
extends java.util.EventListener
```

Interface for receiving notification events about HttpSession attribute changes.

In order to receive these notification events, the implementation class must be either declared in the deployment descriptor of the web application, annotated with `WebListener`, or registered via one of the addListener methods defined on `ServletContext`.

The order in which implementations of this interface are invoked is unspecified.

Since:

Servlet 2.3

Method Summary

void	**attributeAdded**(HttpSessionBindingEvent event) Receives notification that an attribute has been added to a session.
void	**attributeRemoved**(HttpSessionBindingEvent event) Receives notification that an attribute has been removed from a session.
void	**attributeReplaced**(HttpSessionBindingEvent event) Receives notification that an attribute has been replaced in a session.

Method Detail

attributeAdded

```
void attributeAdded(HttpSessionBindingEvent event)
```

Receives notification that an attribute has been added to a session.

Parameters:

`event` - the HttpSessionBindingEvent containing the session and the name and value of the attribute that was added

---

attributeRemoved

`void` **`attributeRemoved`**`(HttpSessionBindingEvent event)`

Receives notification that an attribute has been removed from a session.

Parameters:

`event` - the HttpSessionBindingEvent containing the session and the name and value of the attribute that was removed

---

attributeReplaced

`void` **`attributeReplaced`**`(HttpSessionBindingEvent event)`

Receives notification that an attribute has been replaced in a session.

Parameters:

`event` - the HttpSessionBindingEvent containing the session and the name and (old) value of the attribute that was replaced

---

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

javax.servlet.http

Interface HttpSessionBindingListener

All Superinterfaces:

java.util.EventListener

```
public interface HttpSessionBindingListener
extends java.util.EventListener
```

Causes an object to be notified when it is bound to or unbound from a session. The object is notified by an `HttpSessionBindingEvent` object. This may be as a result of a servlet programmer explicitly unbinding an attribute from a session, due to a session being invalidated, or due to a session timing out.

Author:

Various

See Also:

`HttpSession`, `HttpSessionBindingEvent`

Method Summary

void	**valueBound**(HttpSessionBindingEvent event) Notifies the object that it is being bound to a session and identifies the session.
void	**valueUnbound**(HttpSessionBindingEvent event) Notifies the object that it is being unbound from a session and identifies the session.

Method Detail

valueBound

```
void valueBound(HttpSessionBindingEvent event)
```

Notifies the object that it is being bound to a session and identifies the session.

Parameters:

`event` - the event that identifies the session

See Also:

valueUnbound(javax.servlet.http.HttpSessionBindingEvent)

valueUnbound

`void` **`valueUnbound`**`(`HttpSessionBindingEvent `event)`

Notifies the object that it is being unbound from a session and identifies the session.

Parameters:

`event` - the event that identifies the session

See Also:

`valueBound(javax.servlet.http.HttpSessionBindingEvent)`

Submit a bug or feature

Generated on 10-February-2011 12:41

javax.servlet.http

Interface HttpSessionListener

All Superinterfaces:

java.util.EventListener

```
public interface HttpSessionListener
extends java.util.EventListener
```

Interface for receiving notification events about HttpSession lifecycle changes.

In order to receive these notification events, the implementation class must be either declared in the deployment descriptor of the web application, annotated with `WebListener`, or registered via one of the addListener methods defined on `ServletContext`.

Implementations of this interface are invoked at their `sessionCreated(javax.servlet.http.HttpSessionEvent)` method in the order in which they have been declared, and at their `sessionDestroyed(javax.servlet.http.HttpSessionEvent)` method in reverse order.

Since:

Servlet 2.3

See Also:

`HttpSessionEvent`

Method Summary

void	**sessionCreated**(HttpSessionEvent se) Receives notification that a session has been created.
void	**sessionDestroyed**(HttpSessionEvent se) Receives notification that a session is about to be invalidated.

Method Detail

sessionCreated

```
void sessionCreated(HttpSessionEvent se)
```

Receives notification that a session has been created.

Parameters:

`se` - the HttpSessionEvent containing the session

---

sessionDestroyed

`void` **`sessionDestroyed`**`(HttpSessionEvent se)`

Receives notification that a session is about to be invalidated.

Parameters:

`se` - the HttpSessionEvent containing the session

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

javax.servlet.http

Interface Part

public interface **Part**

This class represents a part or form item that was received within a `multipart/form-data` POST request.

Since:

Servlet 3.0

Method Summary

void	**delete**() Deletes the underlying storage for a file item, including deleting any associated temporary disk file.
java.lang.String	**getContentType**() Gets the content type of this part.
java.lang.String	**getHeader**(java.lang.String name) Returns the value of the specified mime header as a `String`.
java.util.Collection<java.lang.String>	**getHeaderNames**() Gets the header names of this Part.
java.util.Collection<java.lang.String>	**getHeaders**(java.lang.String name) Gets the values of the Part header with the given name.
java.io.InputStream	**getInputStream**() Gets the content of this part as an `InputStream`
java.lang.String	**getName**() Gets the name of this part
long	**getSize**() Returns the size of this fille.
void	**write**(java.lang.String fileName) A convenience method to write this uploaded item to disk.

Method Detail

getInputStream

```
java.io.InputStream getInputStream()
                                    throws java.io.IOException
```

Gets the content of this part as an `InputStream`

Returns:

The content of this part as an `InputStream`

Throws:

`java.io.IOException` - If an error occurs in retrieving the contet as an `InputStream`

---

getContentType

```
java.lang.String getContentType()
```

Gets the content type of this part.

Returns:

The content type of this part.

---

getName

```
java.lang.String getName()
```

Gets the name of this part

Returns:

The name of this part as a `String`

---

getSize

```
long getSize()
```

Returns the size of this fille.

Returns:

a `long` specifying the size of this part, in bytes.

---

write

```
void write(java.lang.String fileName)
```

```
        throws java.io.IOException
```

A convenience method to write this uploaded item to disk.

This method is not guaranteed to succeed if called more than once for the same part. This allows a particular implementation to use, for example, file renaming, where possible, rather than copying all of the underlying data, thus gaining a significant performance benefit.

Parameters:

`fileName` - the name of the file to which the stream will be written. The file is created relative to the location as specified in the MultipartConfig

Throws:

`java.io.IOException` - if an error occurs.

---

delete

```
void delete()
        throws java.io.IOException
```

Deletes the underlying storage for a file item, including deleting any associated temporary disk file.

Throws:

`java.io.IOException` - if an error occurs.

---

getHeader

```
java.lang.String getHeader(java.lang.String name)
```

Returns the value of the specified mime header as a `String`. If the Part did not include a header of the specified name, this method returns `null`. If there are multiple headers with the same name, this method returns the first header in the part. The header name is case insensitive. You can use this method with any request header.

Parameters:

`name` - a `String` specifying the header name

Returns:

a `String` containing the value of the requested header, or `null` if the part does not have a header of that name

---

getHeaders

```
java.util.Collection<java.lang.String> getHeaders(java.lang.String name)
```

Gets the values of the Part header with the given name.

Any changes to the returned `Collection` must not affect this `Part`.

Part header names are case insensitive.

Parameters:

`name` - the header name whose values to return

Returns:

a (possibly empty) `Collection` of the values of the header with the given name

---

getHeaderNames

`java.util.Collection<java.lang.String>` **`getHeaderNames`**`()`

Gets the header names of this Part.

Some servlet containers do not allow servlets to access headers using this method, in which case this method returns `null`

Any changes to the returned `Collection` must not affect this `Part`.

Returns:

a (possibly empty) `Collection` of the header names of this Part

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

javax.servlet.http

Class Cookie

```
java.lang.Object
  └ javax.servlet.http.Cookie
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable

```
public class Cookie
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable
```

Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number. Some Web browsers have bugs in how they handle the optional attributes, so use them sparingly to improve the interoperability of your servlets.

The servlet sends cookies to the browser by using the `HttpServletResponse#addCookie` method, which adds fields to HTTP response headers to send cookies to the browser, one at a time. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.

The browser returns cookies to the servlet by adding fields to HTTP request headers. Cookies can be retrieved from a request by using the `HttpServletRequest#getCookies` method. Several cookies might have the same name but different path attributes.

Cookies affect the caching of the Web pages that use them. HTTP 1.0 does not cache pages that use cookies created with this class. This class does not support the cache control defined with HTTP 1.1.

This class supports both the Version 0 (by Netscape) and Version 1 (by RFC 2109) cookie specifications. By default, cookies are created using Version 0 to ensure the best interoperability.

Author:

Various

See Also:

Serialized Form

Constructor Summary

Cookie(java.lang.String name, java.lang.String value) Constructs a cookie with the specified name and value.

Method Summary

java.lang.Object	**clone**() Overrides the standard java.lang.Object.clone method to return a copy of this Cookie.
java.lang.String	**getComment**() Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
java.lang.String	**getDomain**() Gets the domain name of this Cookie.
int	**getMaxAge**() Gets the maximum age in seconds of this Cookie.
java.lang.String	**getName**() Returns the name of the cookie.
java.lang.String	**getPath**() Returns the path on the server to which the browser returns this cookie.
boolean	**getSecure**() Returns true if the browser is sending cookies only over a secure protocol, or false if the browser can send cookies using any protocol.
java.lang.String	**getValue**() Gets the current value of this Cookie.
int	**getVersion**() Returns the version of the protocol this cookie complies with.
boolean	**isHttpOnly**() Checks whether this Cookie has been marked as *HttpOnly*.
void	**setComment**(java.lang.String purpose) Specifies a comment that describes a cookie's purpose.
void	**setDomain**(java.lang.String domain) Specifies the domain within which this cookie should be presented.
void	**setHttpOnly**(boolean isHttpOnly) Marks or unmarks this Cookie as *HttpOnly*.
void	**setMaxAge**(int expiry) Sets the maximum age in seconds for this Cookie.
void	**setPath**(java.lang.String uri) Specifies a path for the cookie to which the client should return the cookie.

void	**setSecure**(boolean flag) Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.
void	**setValue**(java.lang.String newValue) Assigns a new value to this Cookie.
void	**setVersion**(int v) Sets the version of the cookie protocol that this Cookie complies with.

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Cookie

```
public Cookie(java.lang.String name,
              java.lang.String value)
```

Constructs a cookie with the specified name and value.

The name must conform to RFC 2109. However, vendors may provide a configuration option that allows cookie names conforming to the original Netscape Cookie Specification to be accepted.

The name of a cookie cannot be changed once the cookie has been created.

The value can be anything the server chooses to send. Its value is probably of interest only to the server. The cookie's value can be changed after creation with the `setValue` method.

By default, cookies are created according to the Netscape cookie specification. The version can be changed with the `setVersion` method.

Parameters:

`name` - the name of the cookie

`value` - the value of the cookie

Throws:

`IllegalArgumentException` - if the cookie name is null or empty or contains any illegal characters (for example, a comma, space, or semicolon) or matches a token reserved for use by the cookie protocol

See Also:

`setValue(java.lang.String)`, `setVersion(int)`

Method Detail

setComment

```
public void setComment(java.lang.String purpose)
```

Specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user. Comments are not supported by Netscape Version 0 cookies.

Parameters:

`purpose` - a `String` specifying the comment to display to the user

See Also:

`getComment()`

---

getComment

```
public java.lang.String getComment()
```

Returns the comment describing the purpose of this cookie, or `null` if the cookie has no comment.

Returns:

the comment of the cookie, or `null` if unspecified

See Also:

`setComment(java.lang.String)`

---

setDomain

```
public void setDomain(java.lang.String domain)
```

Specifies the domain within which this cookie should be presented.

The form of the domain name is specified by RFC 2109. A domain name begins with a dot (`.example.com`) and means that the cookie is visible to servers in a specified Domain Name System (DNS) zone (for example, `www.example.com`, but not `a.b.example.com`). By default, cookies are only returned to the server that sent them.

Parameters:

`domain` - the domain name within which this cookie is visible; form is according to RFC 2109

See Also:

`getDomain()`

---

getDomain

```
public java.lang.String getDomain()
```

Gets the domain name of this Cookie.

Domain names are formatted according to RFC 2109.

Returns:

the domain name of this Cookie

See Also:

setDomain(java.lang.String)

---

setMaxAge

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

Sets the maximum age in seconds for this Cookie.

A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the *maximum* age when the cookie will expire, not the cookie's current age.

A negative value means that the cookie is not stored persistently and will be deleted when the Web browser exits. A zero value causes the cookie to be deleted.

Parameters:

`expiry` - an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie is not stored; if zero, deletes the cookie

See Also:

getMaxAge()

---

getMaxAge

```
public int getMaxAge()
```

Gets the maximum age in seconds of this Cookie.

By default, `-1` is returned, which indicates that the cookie will persist until browser shutdown.

Returns:

an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie persists until browser shutdown

See Also:

setMaxAge(int)

---

setPath

```
public void setPath(java.lang.String uri)
```

Specifies a path for the cookie to which the client should return the cookie.

The cookie is visible to all the pages in the directory you specify, and all the pages in that directory's subdirectories. A cookie's path must include the servlet that set the cookie, for example, */catalog*, which makes the cookie visible to all directories on the server under */catalog*.

Consult RFC 2109 (available on the Internet) for more information on setting path names for cookies.

Parameters:

`uri` - a `String` specifying a path

See Also:

`getPath()`

---

getPath

```
public java.lang.String getPath()
```

Returns the path on the server to which the browser returns this cookie. The cookie is visible to all subpaths on the server.

Returns:

a `String` specifying a path that contains a servlet name, for example, */catalog*

See Also:

`setPath(java.lang.String)`

---

setSecure

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

Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.

The default value is `false`.

Parameters:

`flag` - if `true`, sends the cookie from the browser to the server only when using a secure protocol; if `false`, sent on any protocol

See Also:

`getSecure()`

---

getSecure

```
public boolean getSecure()
```

Returns `true` if the browser is sending cookies only over a secure protocol, or `false` if the browser can send cookies using any protocol.

Returns:

`true` if the browser uses a secure protocol, `false` otherwise

See Also:

`setSecure(boolean)`

---

getName

```
public java.lang.String getName()
```

Returns the name of the cookie. The name cannot be changed after creation.

Returns:
the name of the cookie

---

setValue

```
public void setValue(java.lang.String newValue)
```

Assigns a new value to this Cookie.

If you use a binary value, you may want to use BASE64 encoding.

With Version 0 cookies, values should not contain white space, brackets, parentheses, equals signs, commas, double quotes, slashes, question marks, at signs, colons, and semicolons. Empty values may not behave the same way on all browsers.

Parameters:
`newValue` - the new value of the cookie
See Also:
`getValue()`

---

getValue

```
public java.lang.String getValue()
```

Gets the current value of this Cookie.

Returns:
the current value of this Cookie
See Also:
`setValue(java.lang.String)`

---

getVersion

```
public int getVersion()
```

Returns the version of the protocol this cookie complies with. Version 1 complies with RFC 2109, and version 0 complies with the original cookie specification drafted by Netscape. Cookies provided by a browser use and identify the browser's cookie version.

Returns:

0 if the cookie complies with the original Netscape specification; 1 if the cookie complies with RFC 2109

See Also:

`setVersion(int)`

---

setVersion

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

Sets the version of the cookie protocol that this Cookie complies with.

Version 0 complies with the original Netscape cookie specification. Version 1 complies with RFC 2109.

Since RFC 2109 is still somewhat new, consider version 1 as experimental; do not use it yet on production sites.

Parameters:

`v` - 0 if the cookie should comply with the original Netscape specification; 1 if the cookie should comply with RFC 2109

See Also:

`getVersion()`

---

clone

```
public java.lang.Object clone()
```

Overrides the standard `java.lang.Object.clone` method to return a copy of this Cookie.

Overrides:

`clone` in class `java.lang.Object`

---

setHttpOnly

```
public void setHttpOnly(boolean isHttpOnly)
```

Marks or unmarks this Cookie as *HttpOnly*.

If `isHttpOnly` is set to `true`, this cookie is marked as *HttpOnly*, by adding the `HttpOnly` attribute to it.

HttpOnly cookies are not supposed to be exposed to client-side scripting code, and may therefore help mitigate certain kinds of cross-site scripting attacks.

Parameters:

`isHttpOnly` - true if this cookie is to be marked as *HttpOnly*, false otherwise

Since:

Servlet 3.0

isHttpOnly

```
public boolean isHttpOnly()
```

Checks whether this Cookie has been marked as *HttpOnly*.

Returns:

true if this Cookie has been marked as *HttpOnly*, false otherwise

Since:

Servlet 3.0

Submit a bug or feature

Generated on 10-February-2011 12:41

javax.servlet.http

Class HttpServlet

```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
```

All Implemented Interfaces:

java.io.Serializable, Servlet, ServletConfig

```
public abstract class HttpServlet
extends GenericServlet
implements java.io.Serializable
```

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of `HttpServlet` must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

There's almost no reason to override the `service` method. `service` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the `do`*XXX* methods listed above).

Likewise, there's almost no reason to override the `doOptions` and `doTrace` methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. See the Java Tutorial on Multithreaded Programming for more information on handling multiple threads in a Java program.

Author:

Various

See Also:

Serialized Form

Constructor Summary

HttpServlet() Does nothing, because this is an abstract class.

Method Summary

protected void	**doDelete**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void	**doGet**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	**doHead**(HttpServletRequest req, HttpServletResponse resp) Receives an HTTP HEAD request from the protected service method and handles the request.
protected void	**doOptions**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void	**doPost**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a POST request.
protected void	**doPut**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void	**doTrace**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a TRACE request.
protected long	**getLastModified**(HttpServletRequest req) Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	**service**(HttpServletRequest req, HttpServletResponse resp) Receives standard HTTP requests from the public service method and dispatches them to the do*XXX* methods defined in this class.
void	**service**(ServletRequest req, ServletResponse res) Dispatches client requests to the protected service method.

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

HttpServlet

```
public HttpServlet()
```

Does nothing, because this is an abstract class.

Method Detail

doGet

```
protected void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
              throws ServletException,
                     java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a GET request.

Overriding this method to support a GET request also automatically supports an HTTP HEAD request. A HEAD request is a GET request that returns no body in the response, only the request header fields.

When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data. It's best to include content type and encoding. When using a `PrintWriter` object to return the response, set the content type before accessing the `PrintWriter` object.

The servlet container must write the headers before committing the response, because in HTTP the headers must be sent before the response body.

Where possible, set the Content-Length header (with the `ServletResponse.setContentLength(int)` method), to allow the servlet container to use a persistent connection to return its response to the client, improving performance. The content length is automatically set if the entire response fits inside the response buffer.

When using HTTP 1.1 chunked encoding (which means that the response has a Transfer-Encoding header), do not set the Content-Length header.

The GET method should be safe, that is, without any side effects for which users are held responsible. For example, most form queries have no side effects. If a client request is intended to change stored data, the request should use some other HTTP method.

The GET method should also be idempotent, meaning that it can be safely repeated. Sometimes making a method safe also makes it idempotent. For example, repeating queries is both safe and idempotent, but buying a product online or modifying data is neither safe nor idempotent.

If the request is incorrectly formatted, `doGet` returns an HTTP "Bad Request" message.

Parameters:

`req` - an `HttpServletRequest` object that contains the request the client has made of the servlet

`resp` - an `HttpServletResponse` object that contains the response the servlet sends to the client

Throws:

`java.io.IOException` - if an input or output error is detected when the servlet handles the GET request

`ServletException` - if the request for the GET could not be handled

See Also:

`ServletResponse.setContentType(java.lang.String)`

---

getLastModified

```
protected long getLastModified(HttpServletRequest req)
```

Returns the time the `HttpServletRequest` object was last modified, in milliseconds since midnight January 1, 1970 GMT. If the time is unknown, this method returns a negative number (the default).

Servlets that support HTTP GET requests and can quickly determine their last modification time should override this method. This makes browser and proxy caches work more effectively, reducing the load on server and network resources.

Parameters:

`req` - the `HttpServletRequest` object that is sent to the servlet

Returns:

a `long` integer specifying the time the `HttpServletRequest` object was last modified, in milliseconds since midnight, January 1, 1970 GMT, or -1 if the time is not known

---

doHead

```
protected void doHead(HttpServletRequest req,
                      HttpServletResponse resp)
               throws ServletException,
                      java.io.IOException
```

Receives an HTTP HEAD request from the protected `service` method and handles the request. The client sends a HEAD request when it wants to see only the headers of a response, such as Content-Type or Content-Length. The HTTP HEAD method counts the output bytes in the response to set the Content-Length header accurately.

If you override this method, you can avoid computing the response body and just set the response headers directly to improve performance. Make sure that the `doHead` method you write is both safe and idempotent (that is, protects itself from being called multiple times for one HTTP HEAD request).

If the HTTP HEAD request is incorrectly formatted, `doHead` returns an HTTP "Bad Request" message.

Parameters:

`req` - the request object that is passed to the servlet

`resp` - the response object that the servlet uses to return the headers to the clien

Throws:

`java.io.IOException` - if an input or output error occurs

`ServletException` - if the request for the HEAD could not be handled

---

doPost

```
protected void doPost(HttpServletRequest req,
                      HttpServletResponse resp)
               throws ServletException,
                      java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a POST request. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.

When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data. It's best to include content type and encoding. When using a `PrintWriter` object to return the response, set the content type before accessing the `PrintWriter` object.

The servlet container must write the headers before committing the response, because in HTTP the headers must be sent before the response body.

Where possible, set the Content-Length header (with the `ServletResponse.setContentLength(int)` method), to allow the servlet container to use a persistent connection to return its response to the client, improving performance. The content length is automatically set if the entire response fits inside the response buffer.

When using HTTP 1.1 chunked encoding (which means that the response has a Transfer-Encoding header), do not set the Content-Length header.

This method does not need to be either safe or idempotent. Operations requested through POST can have side effects for which the user can be held accountable, for example, updating stored data or buying items online.

If the HTTP POST request is incorrectly formatted, `doPost` returns an HTTP "Bad Request" message.

Parameters:

`req` - an `HttpServletRequest` object that contains the request the client has made of the servlet

`resp` - an `HttpServletResponse` object that contains the response the servlet sends to the client

Throws:

`java.io.IOException` - if an input or output error is detected when the servlet handles the request

`ServletException` - if the request for the POST could not be handled

See Also:

ServletOutputStream, ServletResponse.setContentType(java.lang.String)

---

doPut

```
protected void doPut(HttpServletRequest req,
                     HttpServletResponse resp)
              throws ServletException,
                     java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a PUT request. The PUT operation allows a client to place a file on the server and is similar to sending a file by FTP.

When overriding this method, leave intact any content headers sent with the request (including Content-Length, Content-Type, Content-Transfer-Encoding, Content-Encoding, Content-Base, Content-Language, Content-Location, Content-MD5, and Content-Range). If your method cannot handle a content header, it must issue an error message (HTTP 501 - Not Implemented) and discard the request. For more information on HTTP 1.1, see RFC 2616 .

This method does not need to be either safe or idempotent. Operations that `doPut` performs can have side effects for which the user can be held accountable. When using this method, it may be useful to save a copy of the affected URL in temporary storage.

If the HTTP PUT request is incorrectly formatted, `doPut` returns an HTTP "Bad Request" message.

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet
`resp` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the PUT request
`ServletException` - if the request for the PUT cannot be handled

---

doDelete

```
protected void doDelete(HttpServletRequest req,
                        HttpServletResponse resp)
                 throws ServletException,
                        java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a DELETE request. The DELETE operation allows a client to remove a document or Web page from the server.

This method does not need to be either safe or idempotent. Operations requested through DELETE can have side effects for which users can be held accountable. When using this method, it may be useful to save a copy of the affected URL in temporary storage.

If the HTTP DELETE request is incorrectly formatted, `doDelete` returns an HTTP "Bad Request" message.

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet

`resp` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the DELETE request

`ServletException` - if the request for the DELETE cannot be handled

---

doOptions

```
protected void doOptions(HttpServletRequest req,
                         HttpServletResponse resp)
                  throws ServletException,
                         java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a OPTIONS request. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header. For example, if a servlet overrides `doGet`, this method returns the following header:

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

There's no need to override this method unless the servlet implements new HTTP methods, beyond those implemented by HTTP 1.1.

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet

`resp` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the OPTIONS request

`ServletException` - if the request for the OPTIONS cannot be handled

---

doTrace

```
protected void doTrace(HttpServletRequest req,
                       HttpServletResponse resp)
                throws ServletException,
                       java.io.IOException
```

Called by the server (via the `service` method) to allow a servlet to handle a TRACE request. A TRACE returns the headers sent with the TRACE request to the client, so that they can be used in debugging. There's no need to override this method.

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet

`resp` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the TRACE request

`ServletException` - if the request for the TRACE cannot be handled

---

service

```
protected void service(HttpServletRequest req,
                       HttpServletResponse resp)
                throws ServletException,
                       java.io.IOException
```

Receives standard HTTP requests from the public `service` method and dispatches them to the `do`*XXX* methods defined in this class. This method is an HTTP-specific version of the `Servlet.service(javax.servlet.ServletRequest, javax.servlet.ServletResponse)` method. There's no need to override this method.

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet

`resp` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the HTTP request

`ServletException` - if the HTTP request cannot be handled

See Also:

`Servlet.service(javax.servlet.ServletRequest, javax.servlet.ServletResponse)`

---

service

```
public void service(ServletRequest req,
                    ServletResponse res)
             throws ServletException,
                    java.io.IOException
```

Dispatches client requests to the protected `service` method. There's no need to override this method.

Specified by:

`service` in interface `Servlet`

Specified by:

`service` in class `GenericServlet`

Parameters:

`req` - the `HttpServletRequest` object that contains the request the client made of the servlet

`res` - the `HttpServletResponse` object that contains the response the servlet returns to the client

Throws:

`java.io.IOException` - if an input or output error occurs while the servlet is handling the HTTP request

`ServletException` - if the HTTP request cannot be handled

See Also:

`Servlet.service(javax.servlet.ServletRequest, javax.servlet.ServletResponse)`

Submit a bug or feature

Generated on 10-February-2011 12:41

javax.servlet.http

Class HttpServletRequestWrapper

```
java.lang.Object
  └javax.servlet.ServletRequestWrapper
      └javax.servlet.http.HttpServletRequestWrapper
```

All Implemented Interfaces:

HttpServletRequest, ServletRequest

```
public class HttpServletRequestWrapper
extends ServletRequestWrapper
implements HttpServletRequest
```

Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.

This class implements the Wrapper or Decorator pattern. Methods default to calling through to the wrapped request object.

Since:

Servlet 2.3

See Also:

HttpServletRequest

Field Summary

Fields inherited from interface javax.servlet.http.HttpServletRequest
BASIC_AUTH, CLIENT_CERT_AUTH, DIGEST_AUTH, FORM_AUTH

Constructor Summary

HttpServletRequestWrapper(HttpServletRequest request) Constructs a request object wrapping the given request.

Method Summary

boolean	**authenticate**(HttpServletResponse response)

	The default behavior of this method is to call authenticate on the wrapped request object.
java.lang.String	**getAuthType**() The default behavior of this method is to return getAuthType() on the wrapped request object.
java.lang.String	**getContextPath**() The default behavior of this method is to return getContextPath() on the wrapped request object.
Cookie[]	**getCookies**() The default behavior of this method is to return getCookies() on the wrapped request object.
long	**getDateHeader**(java.lang.String name) The default behavior of this method is to return getDateHeader(String name) on the wrapped request object.
java.lang.String	**getHeader**(java.lang.String name) The default behavior of this method is to return getHeader(String name) on the wrapped request object.
java.util.Enumeration<java.lang.String>	**getHeaderNames**() The default behavior of this method is to return getHeaderNames() on the wrapped request object.
java.util.Enumeration<java.lang.String>	**getHeaders**(java.lang.String name) The default behavior of this method is to return getHeaders(String name) on the wrapped request object.
int	**getIntHeader**(java.lang.String name) The default behavior of this method is to return getIntHeader(String name) on the wrapped request object.
java.lang.String	**getMethod**() The default behavior of this method is to return getMethod() on the wrapped request object.
Part	**getPart**(java.lang.String name) The default behavior of this method is to call getPart on the wrapped request object.
java.util.Collection<Part>	**getParts**() The default behavior of this method is to call getParts on the wrapped request object.
java.lang.String	**getPathInfo**() The default behavior of this method is to return getPathInfo() on the wrapped request object.
java.lang.String	**getPathTranslated**() The default behavior of this method is to return

	getPathTranslated() on the wrapped request object.
java.lang.String	**getQueryString**() The default behavior of this method is to return getQueryString() on the wrapped request object.
java.lang.String	**getRemoteUser**() The default behavior of this method is to return getRemoteUser() on the wrapped request object.
java.lang.String	**getRequestedSessionId**() The default behavior of this method is to return getRequestedSessionId() on the wrapped request object.
java.lang.String	**getRequestURI**() The default behavior of this method is to return getRequestURI() on the wrapped request object.
java.lang.StringBuffer	**getRequestURL**() The default behavior of this method is to return getRequestURL() on the wrapped request object.
java.lang.String	**getServletPath**() The default behavior of this method is to return getServletPath() on the wrapped request object.
HttpSession	**getSession**() The default behavior of this method is to return getSession() on the wrapped request object.
HttpSession	**getSession**(boolean create) The default behavior of this method is to return getSession(boolean create) on the wrapped request object.
java.security.Principal	**getUserPrincipal**() The default behavior of this method is to return getUserPrincipal() on the wrapped request object.
boolean	**isRequestedSessionIdFromCookie**() The default behavior of this method is to return isRequestedSessionIdFromCookie() on the wrapped request object.
boolean	**isRequestedSessionIdFromUrl**() The default behavior of this method is to return isRequestedSessionIdFromUrl() on the wrapped request object.
boolean	**isRequestedSessionIdFromURL**() The default behavior of this method is to return isRequestedSessionIdFromURL() on the wrapped request object.

boolean	**isRequestedSessionIdValid**() The default behavior of this method is to return isRequestedSessionIdValid() on the wrapped request object.
boolean	**isUserInRole**(java.lang.String role) The default behavior of this method is to return isUserInRole(String role) on the wrapped request object.
void	**login**(java.lang.String username, java.lang.String password) The default behavior of this method is to call login on the wrapped request object.
void	**logout**() The default behavior of this method is to call login on the wrapped request object.

Methods inherited from class javax.servlet.ServletRequestWrapper

getAsyncContext, getAttribute, getAttributeNames, getCharacterEncoding, getContentLength, getContentType, getDispatcherType, getInputStream, getLocalAddr, getLocale, getLocales, getLocalName, getLocalPort, getParameter, getParameterMap, getParameterNames, getParameterValues, getProtocol, getReader, getRealPath, getRemoteAddr, getRemoteHost, getRemotePort, getRequest, getRequestDispatcher, getScheme, getServerName, getServerPort, getServletContext, isAsyncStarted, isAsyncSupported, isSecure, isWrapperFor, isWrapperFor, removeAttribute, setAttribute, setCharacterEncoding, setRequest, startAsync, startAsync

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface javax.servlet.ServletRequest

getAsyncContext, getAttribute, getAttributeNames, getCharacterEncoding, getContentLength, getContentType, getDispatcherType, getInputStream, getLocalAddr, getLocale, getLocales, getLocalName, getLocalPort, getParameter, getParameterMap, getParameterNames, getParameterValues, getProtocol, getReader, getRealPath, getRemoteAddr, getRemoteHost, getRemotePort, getRequestDispatcher, getScheme, getServerName, getServerPort, getServletContext, isAsyncStarted, isAsyncSupported, isSecure, removeAttribute, setAttribute, setCharacterEncoding, startAsync, startAsync

Constructor Detail

HttpServletRequestWrapper

public **HttpServletRequestWrapper**(HttpServletRequest request)

Constructs a request object wrapping the given request.

Throws:

`java.lang.IllegalArgumentException` - if the request is null

Method Detail

getAuthType

```
public java.lang.String getAuthType()
```

The default behavior of this method is to return getAuthType() on the wrapped request object.

Specified by:

`getAuthType` in interface `HttpServletRequest`

Returns:

one of the static members BASIC_AUTH, FORM_AUTH, CLIENT_CERT_AUTH, DIGEST_AUTH (suitable for == comparison) or the container-specific string indicating the authentication scheme, or `null` if the request was not authenticated.

---

getCookies

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

The default behavior of this method is to return getCookies() on the wrapped request object.

Specified by:

`getCookies` in interface `HttpServletRequest`

Returns:

an array of all the `Cookies` included with this request, or `null` if the request has no cookies

---

getDateHeader

```
public long getDateHeader(java.lang.String name)
```

The default behavior of this method is to return getDateHeader(String name) on the wrapped request object.

Specified by:

`getDateHeader` in interface `HttpServletRequest`

Parameters:

`name` - a `String` specifying the name of the header

Returns:

a `long` value representing the date specified in the header expressed as the number of milliseconds since January 1, 1970 GMT, or -1 if the named header was not included with the request

---

getHeader

```
public java.lang.String getHeader(java.lang.String name)
```

The default behavior of this method is to return getHeader(String name) on the wrapped request object.

Specified by:

`getHeader` in interface `HttpServletRequest`

Parameters:

`name` - a `String` specifying the header name

Returns:

a `String` containing the value of the requested header, or `null` if the request does not have a header of that name

---

getHeaders

```
public java.util.Enumeration<java.lang.String> getHeaders(java.lang.String name)
```

The default behavior of this method is to return getHeaders(String name) on the wrapped request object.

Specified by:

`getHeaders` in interface `HttpServletRequest`

Parameters:

`name` - a `String` specifying the header name

Returns:

an `Enumeration` containing the values of the requested header. If the request does not have any headers of that name return an empty enumeration. If the container does not allow access to header information, return null

---

getHeaderNames

```
public java.util.Enumeration<java.lang.String> getHeaderNames()
```

The default behavior of this method is to return getHeaderNames() on the wrapped request object.

Specified by:

`getHeaderNames` in interface `HttpServletRequest`

Returns:

an enumeration of all the header names sent with this request; if the request has no headers, an empty enumeration; if the servlet container does not allow servlets to use this method, `null`

---

getIntHeader

```
public int getIntHeader(java.lang.String name)
```

The default behavior of this method is to return getIntHeader(String name) on the wrapped request object.

Specified by:

getIntHeader in interface HttpServletRequest

Parameters:

`name` - a `String` specifying the name of a request header

Returns:

an integer expressing the value of the request header or -1 if the request doesn't have a header of this name

---

getMethod

```
public java.lang.String getMethod()
```

The default behavior of this method is to return getMethod() on the wrapped request object.

Specified by:

getMethod in interface HttpServletRequest

Returns:

a `String` specifying the name of the method with which this request was made

---

getPathInfo

```
public java.lang.String getPathInfo()
```

The default behavior of this method is to return getPathInfo() on the wrapped request object.

Specified by:

getPathInfo in interface HttpServletRequest

Returns:

a `String`, decoded by the web container, specifying extra path information that comes after the servlet path but before the query string in the request URL; or `null` if the URL does not have any extra path information

---

getPathTranslated

`public java.lang.String` **`getPathTranslated`**`()`

The default behavior of this method is to return getPathTranslated() on the wrapped request object.

Specified by:

`getPathTranslated` in interface `HttpServletRequest`

Returns:

a `String` specifying the real path, or `null` if the URL does not have any extra path information

---

getContextPath

`public java.lang.String` **`getContextPath`**`()`

The default behavior of this method is to return getContextPath() on the wrapped request object.

Specified by:

`getContextPath` in interface `HttpServletRequest`

Returns:

a `String` specifying the portion of the request URI that indicates the context of the request

See Also:

`ServletContext.getContextPath()`

---

getQueryString

`public java.lang.String` **`getQueryString`**`()`

The default behavior of this method is to return getQueryString() on the wrapped request object.

Specified by:

`getQueryString` in interface `HttpServletRequest`

Returns:

a `String` containing the query string or `null` if the URL contains no query string. The value is not decoded by the container.

---

getRemoteUser

`public java.lang.String` **`getRemoteUser`**`()`

The default behavior of this method is to return getRemoteUser() on the wrapped request object.

Specified by:

getRemoteUser in interface HttpServletRequest

Returns:

a String specifying the login of the user making this request, or null if the user login is not known

---

isUserInRole

```
public boolean isUserInRole(java.lang.String role)
```

The default behavior of this method is to return isUserInRole(String role) on the wrapped request object.

Specified by:

isUserInRole in interface HttpServletRequest

Parameters:

role - a String specifying the name of the role

Returns:

a boolean indicating whether the user making this request belongs to a given role; false if the user has not been authenticated

---

getUserPrincipal

```
public java.security.Principal getUserPrincipal()
```

The default behavior of this method is to return getUserPrincipal() on the wrapped request object.

Specified by:

getUserPrincipal in interface HttpServletRequest

Returns:

a java.security.Principal containing the name of the user making this request; null if the user has not been authenticated

---

getRequestedSessionId

```
public java.lang.String getRequestedSessionId()
```

The default behavior of this method is to return getRequestedSessionId() on the wrapped request object.

Specified by:

getRequestedSessionId in interface HttpServletRequest

Returns:

a String specifying the session ID, or null if the request did not specify a session ID

See Also:

HttpServletRequest.isRequestedSessionIdValid()

---

getRequestURI

```
public java.lang.String getRequestURI()
```

The default behavior of this method is to return getRequestURI() on the wrapped request object.

Specified by:

getRequestURI in interface HttpServletRequest

Returns:

a `String` containing the part of the URL from the protocol name up to the query string

See Also:

`HttpUtils#getRequestURL`

---

getRequestURL

```
public java.lang.StringBuffer getRequestURL()
```

The default behavior of this method is to return getRequestURL() on the wrapped request object.

Specified by:

getRequestURL in interface HttpServletRequest

Returns:

a `StringBuffer` object containing the reconstructed URL

---

getServletPath

```
public java.lang.String getServletPath()
```

The default behavior of this method is to return getServletPath() on the wrapped request object.

Specified by:

getServletPath in interface HttpServletRequest

Returns:

a `String` containing the name or path of the servlet being called, as specified in the request URL, decoded, or an empty string if the servlet used to process the request is matched using the "/*" pattern.

---

getSession

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

The default behavior of this method is to return getSession(boolean create) on the wrapped request object.

Specified by:

getSession in interface HttpServletRequest

Parameters:

create - true to create a new session for this request if necessary; false to return null if there's no current session

Returns:

the HttpSession associated with this request or null if create is false and the request has no valid session

See Also:

HttpServletRequest.getSession()

---

getSession

```
public HttpSession getSession()
```

The default behavior of this method is to return getSession() on the wrapped request object.

Specified by:

getSession in interface HttpServletRequest

Returns:

the HttpSession associated with this request

See Also:

HttpServletRequest.getSession(boolean)

---

isRequestedSessionIdValid

```
public boolean isRequestedSessionIdValid()
```

The default behavior of this method is to return isRequestedSessionIdValid() on the wrapped request object.

Specified by:

isRequestedSessionIdValid in interface HttpServletRequest

Returns:

true if this request has an id for a valid session in the current session context; false otherwise

See Also:

HttpServletRequest.getRequestedSessionId(), HttpServletRequest.getSession(boolean), HttpSessionContext

---

isRequestedSessionIdFromCookie

```
public boolean isRequestedSessionIdFromCookie()
```

The default behavior of this method is to return isRequestedSessionIdFromCookie() on the wrapped request object.

Specified by:

isRequestedSessionIdFromCookie in interface HttpServletRequest

Returns:

`true` if the session ID came in as a cookie; otherwise, `false`

See Also:

HttpServletRequest.getSession(boolean)

---

isRequestedSessionIdFromURL

```
public boolean isRequestedSessionIdFromURL()
```

The default behavior of this method is to return isRequestedSessionIdFromURL() on the wrapped request object.

Specified by:

isRequestedSessionIdFromURL in interface HttpServletRequest

Returns:

`true` if the session ID came in as part of a URL; otherwise, `false`

See Also:

HttpServletRequest.getSession(boolean)

---

isRequestedSessionIdFromUrl

```
public boolean isRequestedSessionIdFromUrl()
```

The default behavior of this method is to return isRequestedSessionIdFromUrl() on the wrapped request object.

Specified by:

isRequestedSessionIdFromUrl in interface HttpServletRequest

---

authenticate

```
public boolean authenticate(HttpServletResponse response)
                    throws java.io.IOException,
                           ServletException
```

The default behavior of this method is to call authenticate on the wrapped request object.

Specified by:

authenticate in interface HttpServletRequest

Parameters:

`response` - The `HttpServletResponse` associated with this `HttpServletRequest`

Returns:

`true` when non-null values were or have been established as the values returned by `getUserPrincipal`, `getRemoteUser`, and `getAuthType`. Return `false` if authentication is incomplete and the underlying login mechanism has committed, in the response, the message (e.g., challenge) and HTTP status code to be returned to the user.

Throws:

`java.io.IOException` - if an input or output error occurred while reading from this request or writing to the given response

`ServletException` - if the authentication failed and the caller is responsible for handling the error (i.e., the underlying login mechanism did NOT establish the message and HTTP status code to be returned to the user)

Since:

Servlet 3.0

---

login

```
public void login(java.lang.String username,
                  java.lang.String password)
           throws ServletException
```

The default behavior of this method is to call login on the wrapped request object.

Specified by:

login in interface HttpServletRequest

Parameters:

`username` - The `String` value corresponding to the login identifier of the user.

`password` - The password `String` corresponding to the identified user.

Throws:

`ServletException` - if the configured login mechanism does not support username password authentication, or if a non-null caller identity had already been established (prior to the call to login), or if validation of the provided username and password fails.

Since:

Servlet 3.0

---

logout

```
public void logout()
            throws ServletException
```

The default behavior of this method is to call login on the wrapped request object.

Specified by:

logout in interface HttpServletRequest

Throws:

ServletException - if logout fails

Since:

Servlet 3.0

---

getParts

```
public java.util.Collection<Part> getParts()
                                  throws java.io.IOException,
                                         ServletException
```

The default behavior of this method is to call getParts on the wrapped request object.

Any changes to the returned Collection must not affect this HttpServletRequestWrapper.

Specified by:

getParts in interface HttpServletRequest

Returns:

a (possibly empty) Collection of the Part components of this request

Throws:

java.io.IOException - if an I/O error occurred during the retrieval of the Part components of this request

ServletException - if this request is not of type multipart/form-data

Since:

Servlet 3.0

See Also:

MultipartConfig.maxFileSize(), MultipartConfig.maxRequestSize()

---

getPart

```
public Part getPart(java.lang.String name)
             throws java.io.IOException,
                    ServletException
```

The default behavior of this method is to call getPart on the wrapped request object.

Specified by:

getPart in interface HttpServletRequest

Parameters:

name - the name of the requested Part

Returns:

The Part with the given name, or `null` if this request is of type `multipart/form-data`, but does not contain the requested Part

Throws:

`java.io.IOException` - if an I/O error occurred during the retrieval of the requested Part

`ServletException` - if this request is not of type `multipart/form-data`

Since:

Servlet 3.0

See Also:

`MultipartConfig.maxFileSize()`, `MultipartConfig.maxRequestSize()`

---

javax.servlet.http

Class HttpServletResponseWrapper

```
java.lang.Object
  └ javax.servlet.ServletResponseWrapper
      └ javax.servlet.http.HttpServletResponseWrapper
```

All Implemented Interfaces:

HttpServletResponse, ServletResponse

```
public class HttpServletResponseWrapper
extends ServletResponseWrapper
implements HttpServletResponse
```

Provides a convenient implementation of the HttpServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet. This class implements the Wrapper or Decorator pattern. Methods default to calling through to the wrapped response object.

Since:

Servlet 2.3

Author:

Various

See Also:

HttpServletResponse

Field Summary

Fields inherited from interface javax.servlet.http.HttpServletResponse

SC_ACCEPTED, SC_BAD_GATEWAY, SC_BAD_REQUEST, SC_CONFLICT, SC_CONTINUE, SC_CREATED, SC_EXPECTATION_FAILED, SC_FORBIDDEN, SC_FOUND, SC_GATEWAY_TIMEOUT, SC_GONE, SC_HTTP_VERSION_NOT_SUPPORTED, SC_INTERNAL_SERVER_ERROR, SC_LENGTH_REQUIRED, SC_METHOD_NOT_ALLOWED, SC_MOVED_PERMANENTLY, SC_MOVED_TEMPORARILY, SC_MULTIPLE_CHOICES, SC_NO_CONTENT, SC_NON_AUTHORITATIVE_INFORMATION, SC_NOT_ACCEPTABLE, SC_NOT_FOUND, SC_NOT_IMPLEMENTED, SC_NOT_MODIFIED, SC_OK, SC_PARTIAL_CONTENT, SC_PAYMENT_REQUIRED, SC_PRECONDITION_FAILED, SC_PROXY_AUTHENTICATION_REQUIRED, SC_REQUEST_ENTITY_TOO_LARGE, SC_REQUEST_TIMEOUT, SC_REQUEST_URI_TOO_LONG, SC_REQUESTED_RANGE_NOT_SATISFIABLE, SC_RESET_CONTENT, SC_SEE_OTHER, SC_SERVICE_UNAVAILABLE, SC_SWITCHING_PROTOCOLS, SC_TEMPORARY_REDIRECT, SC_UNAUTHORIZED, SC_UNSUPPORTED_MEDIA_TYPE, SC_USE_PROXY

Constructor Summary

HttpServletResponseWrapper(HttpServletResponse response) Constructs a response adaptor wrapping the given response.

Method Summary

void	**addCookie**(Cookie cookie) The default behavior of this method is to call addCookie(Cookie cookie) on the wrapped response object.
void	**addDateHeader**(java.lang.String name, long date) The default behavior of this method is to call addDateHeader(String name, long date) on the wrapped response object.
void	**addHeader**(java.lang.String name, java.lang.String value) The default behavior of this method is to return addHeader(String name, String value) on the wrapped response object.
void	**addIntHeader**(java.lang.String name, int value) The default behavior of this method is to call addIntHeader(String name, int value) on the wrapped response object.
boolean	**containsHeader**(java.lang.String name) The default behavior of this method is to call containsHeader(String name) on the wrapped response object.
java.lang.String	**encodeRedirectUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeRedirectURL(String url) instead*
java.lang.String	**encodeRedirectURL**(java.lang.String url) The default behavior of this method is to return encodeRedirectURL(String url) on the wrapped response object.
java.lang.String	**encodeUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeURL(String url) instead*
java.lang.String	**encodeURL**(java.lang.String url) The default behavior of this method is to call encodeURL(String url) on the wrapped response object.
java.lang.String	**getHeader**(java.lang.String name) The default behaviour of this method is to call

	HttpServletResponse#getHeader on the wrapped response object.
java.util.Collection<java.lang.String>	**getHeaderNames**() The default behaviour of this method is to call HttpServletResponse#getHeaderNames on the wrapped response object.
java.util.Collection<java.lang.String>	**getHeaders**(java.lang.String name) The default behaviour of this method is to call HttpServletResponse#getHeaders on the wrapped response object.
int	**getStatus**() The default behaviour of this method is to call HttpServletResponse#getStatus on the wrapped response object.
void	**sendError**(int sc) The default behavior of this method is to call sendError(int sc) on the wrapped response object.
void	**sendError**(int sc, java.lang.String msg) The default behavior of this method is to call sendError(int sc, String msg) on the wrapped response object.
void	**sendRedirect**(java.lang.String location) The default behavior of this method is to return sendRedirect(String location) on the wrapped response object.
void	**setDateHeader**(java.lang.String name, long date) The default behavior of this method is to call setDateHeader(String name, long date) on the wrapped response object.
void	**setHeader**(java.lang.String name, java.lang.String value) The default behavior of this method is to return setHeader(String name, String value) on the wrapped response object.
void	**setIntHeader**(java.lang.String name, int value) The default behavior of this method is to call setIntHeader(String name, int value) on the wrapped response object.
void	**setStatus**(int sc) The default behavior of this method is to call setStatus(int sc) on the wrapped response object.
void	**setStatus**(int sc, java.lang.String sm)

	Deprecated. *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use* `setStatus(int)`*, to send an error with a description use* `sendError(int, String)`

Methods inherited from class javax.servlet.ServletResponseWrapper
flushBuffer, getBufferSize, getCharacterEncoding, getContentType, getLocale, getOutputStream, getResponse, getWriter, isCommitted, isWrapperFor, isWrapperFor, reset, resetBuffer, setBufferSize, setCharacterEncoding, setContentLength, setContentType, setLocale, setResponse

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface javax.servlet.ServletResponse
flushBuffer, getBufferSize, getCharacterEncoding, getContentType, getLocale, getOutputStream, getWriter, isCommitted, reset, resetBuffer, setBufferSize, setCharacterEncoding, setContentLength, setContentType, setLocale

Constructor Detail

HttpServletResponseWrapper

```
public HttpServletResponseWrapper(HttpServletResponse response)
```

Constructs a response adaptor wrapping the given response.

Throws:

`java.lang.IllegalArgumentException` - if the response is null

Method Detail

addCookie

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

The default behavior of this method is to call addCookie(Cookie cookie) on the wrapped response object.

Specified by:

`addCookie` in interface `HttpServletResponse`

Parameters:

`cookie` - the Cookie to return to the client

---

containsHeader

```
public boolean containsHeader(java.lang.String name)
```

The default behavior of this method is to call containsHeader(String name) on the wrapped response object.

Specified by:

`containsHeader` in interface `HttpServletResponse`

Parameters:

`name` - the header name

Returns:

`true` if the named response header has already been set; `false` otherwise

---

encodeURL

```
public java.lang.String encodeURL(java.lang.String url)
```

The default behavior of this method is to call encodeURL(String url) on the wrapped response object.

Specified by:

`encodeURL` in interface `HttpServletResponse`

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

encodeRedirectURL

```
public java.lang.String encodeRedirectURL(java.lang.String url)
```

The default behavior of this method is to return encodeRedirectURL(String url) on the wrapped response object.

Specified by:

`encodeRedirectURL` in interface `HttpServletResponse`

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

See Also:

HttpServletResponse.sendRedirect(java.lang.String), HttpServletResponse.encodeUrl(java.lang.String)

---

encodeUrl

```
public java.lang.String encodeUrl(java.lang.String url)
```

Deprecated. *As of version 2.1, use encodeURL(String url) instead*

The default behavior of this method is to call encodeUrl(String url) on the wrapped response object.

Specified by:

encodeUrl in interface HttpServletResponse

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

encodeRedirectUrl

```
public java.lang.String encodeRedirectUrl(java.lang.String url)
```

Deprecated. *As of version 2.1, use encodeRedirectURL(String url) instead*

The default behavior of this method is to return encodeRedirectUrl(String url) on the wrapped response object.

Specified by:

encodeRedirectUrl in interface HttpServletResponse

Parameters:

`url` - the url to be encoded.

Returns:

the encoded URL if encoding is needed; the unchanged URL otherwise.

---

sendError

```
public void sendError(int sc,
                      java.lang.String msg)
               throws java.io.IOException
```

The default behavior of this method is to call sendError(int sc, String msg) on the wrapped response object.

Specified by:

sendError in interface HttpServletResponse

Parameters:

`sc` - the error status code

`msg` - the descriptive message

Throws:

`java.io.IOException` - If an input or output exception occurs

---

sendError

```
public void sendError(int sc)
               throws java.io.IOException
```

The default behavior of this method is to call sendError(int sc) on the wrapped response object.

Specified by:

sendError in interface HttpServletResponse

Parameters:

`sc` - the error status code

Throws:

`java.io.IOException` - If an input or output exception occurs

---

sendRedirect

```
public void sendRedirect(java.lang.String location)
                  throws java.io.IOException
```

The default behavior of this method is to return sendRedirect(String location) on the wrapped response object.

Specified by:

sendRedirect in interface HttpServletResponse

Parameters:

`location` - the redirect location URL

Throws:

`java.io.IOException` - If an input or output exception occurs

---

setDateHeader

```
public void setDateHeader(java.lang.String name,
                          long date)
```

The default behavior of this method is to call setDateHeader(String name, long date) on the wrapped response object.

Specified by:

setDateHeader in interface HttpServletResponse

Parameters:

name - the name of the header to set

date - the assigned date value

See Also:

HttpServletResponse.containsHeader(java.lang.String), HttpServletResponse.addDateHeader(java.lang.String, long)

---

addDateHeader

```
public void addDateHeader(java.lang.String name,
                          long date)
```

The default behavior of this method is to call addDateHeader(String name, long date) on the wrapped response object.

Specified by:

addDateHeader in interface HttpServletResponse

Parameters:

name - the name of the header to set

date - the additional date value

See Also:

HttpServletResponse.setDateHeader(java.lang.String, long)

---

setHeader

```
public void setHeader(java.lang.String name,
                      java.lang.String value)
```

The default behavior of this method is to return setHeader(String name, String value) on the wrapped response object.

Specified by:

setHeader in interface HttpServletResponse

Parameters:

name - the name of the header

value - the header value If it contains octet string, it should be encoded according to RFC 2047 (http://www.ietf.org/rfc/rfc2047.txt)

See Also:

HttpServletResponse.containsHeader(java.lang.String),
HttpServletResponse.addHeader(java.lang.String, java.lang.String)

---

addHeader

```
public void addHeader(java.lang.String name,
                      java.lang.String value)
```

The default behavior of this method is to return addHeader(String name, String value) on the wrapped response object.

Specified by:

addHeader in interface HttpServletResponse

Parameters:

name - the name of the header

value - the additional header value If it contains octet string, it should be encoded according to RFC 2047 (http://www.ietf.org/rfc/rfc2047.txt)

See Also:

HttpServletResponse.setHeader(java.lang.String, java.lang.String)

---

setIntHeader

```
public void setIntHeader(java.lang.String name,
                         int value)
```

The default behavior of this method is to call setIntHeader(String name, int value) on the wrapped response object.

Specified by:

setIntHeader in interface HttpServletResponse

Parameters:

name - the name of the header

value - the assigned integer value

See Also:

HttpServletResponse.containsHeader(java.lang.String),
HttpServletResponse.addIntHeader(java.lang.String, int)

---

addIntHeader

```
public void addIntHeader(java.lang.String name,
                         int value)
```

The default behavior of this method is to call addIntHeader(String name, int value) on the wrapped response object.

Specified by:

addIntHeader in interface HttpServletResponse

Parameters:

name - the name of the header

value - the assigned integer value

See Also:

HttpServletResponse.setIntHeader(java.lang.String, int)

---

setStatus

```
public void setStatus(int sc)
```

The default behavior of this method is to call setStatus(int sc) on the wrapped response object.

Specified by:

setStatus in interface HttpServletResponse

Parameters:

sc - the status code

See Also:

HttpServletResponse.sendError(int, java.lang.String)

---

setStatus

```
public void setStatus(int sc,
                      java.lang.String sm)
```

Deprecated. *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use setStatus(int), to send an error with a description use sendError(int, String)*

The default behavior of this method is to call setStatus(int sc, String sm) on the wrapped response object.

Specified by:

setStatus in interface HttpServletResponse

Parameters:

sc - the status code

sm - the status message

---

getStatus

```
public int getStatus()
```

The default behaviour of this method is to call HttpServletResponse#getStatus on the wrapped

response object.

Specified by:

getStatus in interface HttpServletResponse

Returns:

the current status code of the wrapped response

---

getHeader

```
public java.lang.String getHeader(java.lang.String name)
```

The default behaviour of this method is to call HttpServletResponse#getHeader on the wrapped response object.

Specified by:

getHeader in interface HttpServletResponse

Parameters:

name - the name of the response header whose value to return

Returns:

the value of the response header with the given name, or null if no header with the given name has been set on the wrapped response

Since:

Servlet 3.0

---

getHeaders

```
public java.util.Collection<java.lang.String> getHeaders(java.lang.String name)
```

The default behaviour of this method is to call HttpServletResponse#getHeaders on the wrapped response object.

Any changes to the returned Collection must not affect this HttpServletResponseWrapper.

Specified by:

getHeaders in interface HttpServletResponse

Parameters:

name - the name of the response header whose values to return

Returns:

a (possibly empty) Collection of the values of the response header with the given name

Since:

Servlet 3.0

---

getHeaderNames

```
public java.util.Collection<java.lang.String> getHeaderNames()
```

The default behaviour of this method is to call `HttpServletResponse#getHeaderNames` on the wrapped response object.

Any changes to the returned `Collection` must not affect this `HttpServletResponseWrapper`.

Specified by:

`getHeaderNames` in interface `HttpServletResponse`

Returns:

a (possibly empty) `Collection` of the names of the response headers

Since:

Servlet 3.0

---

javax.servlet.http

Class HttpSessionBindingEvent

```
java.lang.Object
  └java.util.EventObject
      └javax.servlet.http.HttpSessionEvent
          └javax.servlet.http.HttpSessionBindingEvent
```

All Implemented Interfaces:

java.io.Serializable

```
public class HttpSessionBindingEvent
extends HttpSessionEvent
```

Events of this type are either sent to an object that implements `HttpSessionBindingListener` when it is bound or unbound from a session, or to a `HttpSessionAttributeListener` that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.

The session binds the object by a call to `HttpSession.setAttribute` and unbinds the object by a call to `HttpSession.removeAttribute`.

Author:

Various

See Also:

`HttpSession`, `HttpSessionBindingListener`, `HttpSessionAttributeListener`, Serialized Form

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary

HttpSessionBindingEvent(HttpSession session, java.lang.String name) Constructs an event that notifies an object that it has been bound to or unbound from a session.
HttpSessionBindingEvent(HttpSession session, java.lang.String name, java.lang.Object value)

Constructs an event that notifies an object that it has been bound to or unbound from a session.

Method Summary

java.lang.String	**getName**() Returns the name with which the attribute is bound to or unbound from the session.
HttpSession	**getSession**() Return the session that changed.
java.lang.Object	**getValue**() Returns the value of the attribute that has been added, removed or replaced.

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

HttpSessionBindingEvent

```
public HttpSessionBindingEvent(HttpSession session,
                               java.lang.String name)
```

Constructs an event that notifies an object that it has been bound to or unbound from a session. To receive the event, the object must implement `HttpSessionBindingListener`.

Parameters:

`session` - the session to which the object is bound or unbound

`name` - the name with which the object is bound or unbound

See Also:

`getName()`, `getSession()`

HttpSessionBindingEvent

```
public HttpSessionBindingEvent(HttpSession session,
                               java.lang.String name,
                               java.lang.Object value)
```

Constructs an event that notifies an object that it has been bound to or unbound from a session. To receive the event, the object must implement `HttpSessionBindingListener`.

Parameters:

`session` - the session to which the object is bound or unbound

`name` - the name with which the object is bound or unbound

See Also:

`getName()`, `getSession()`

Method Detail

getSession

```
public HttpSession getSession()
```

Return the session that changed.

Overrides:

`getSession` in class `HttpSessionEvent`

---

getName

```
public java.lang.String getName()
```

Returns the name with which the attribute is bound to or unbound from the session.

Returns:

a string specifying the name with which the object is bound to or unbound from the session

---

getValue

```
public java.lang.Object getValue()
```

Returns the value of the attribute that has been added, removed or replaced. If the attribute was added (or bound), this is the value of the attribute. If the attribute was removed (or unbound), this is the value of the removed attribute. If the attribute was replaced, this is the old value of the attribute.

Since:

Servlet 2.3

---

javax.servlet.http

Class HttpSessionEvent

```
java.lang.Object
  └java.util.EventObject
      └javax.servlet.http.HttpSessionEvent
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

HttpSessionBindingEvent

```
public class HttpSessionEvent
extends java.util.EventObject
```

This is the class representing event notifications for changes to sessions within a web application.

Since:

Servlet 2.3

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary

HttpSessionEvent(HttpSession source) Construct a session event from the given source.

Method Summary

HttpSession	**getSession**() Return the session that changed.

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

HttpSessionEvent

```
public HttpSessionEvent(HttpSession source)
```

Construct a session event from the given source.

Method Detail

getSession

```
public HttpSession getSession()
```

Return the session that changed.

Generated on 10-February-2011 12:41