Servlet API

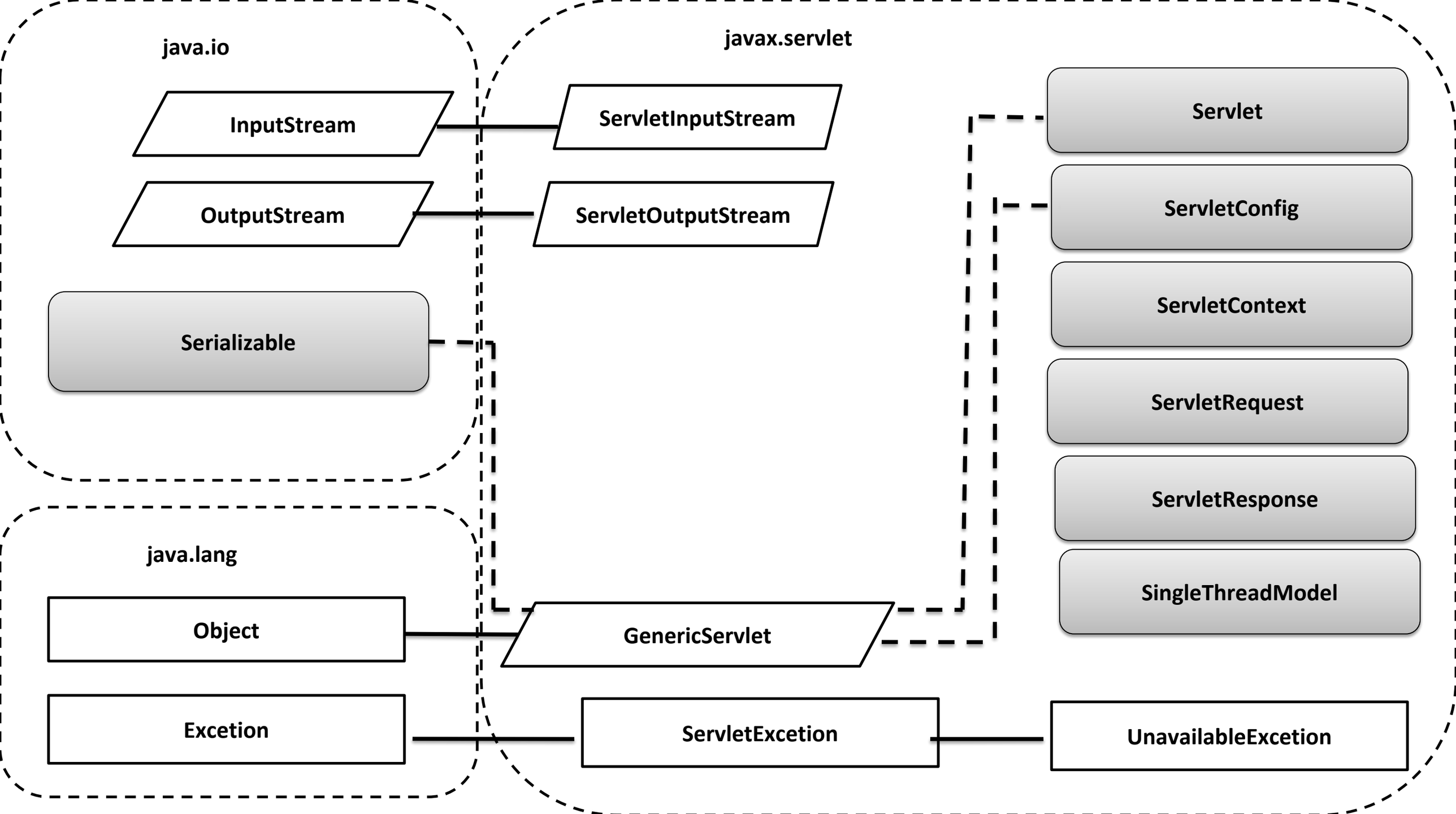
- In the servlet basically two packages contains the classes and interfaces which are used to develop servlet programs. They are as follows:
- **Javax.servlet** package
- **Javax.servlet.http** package

Javax.servlet package

- It is having following classes and interfaces. all these classes and interfaces are shown in fig. 1.
- **Servlet interface:**
  - It is having the methods which define the life cycle of servlet.
- **ServletConfig interface:**
  - It provides basic or initialization parameter of the servlet.
- **ServletContext interface:**
  - It provides the runtime environment to the servlet. It also logs the events using log().

- **GenericServlet class:**
  - It implements the Servlet, ServletConfig and Serialization interface.
- **ServletRequest interface:**
  - It is used to read the client request.
- **ServletResponse interface:**
  - It is used to write the response data.
- **ServletException class:**
  - It defines that servlet occurred errors.
- **UnavailableException class:**
  - It defines that servlet is permanently or temporarily not available.
- **ServletInputStream class:**
  - It provides input stream to read the data from the client request.
- **ServletOutputStream class:**
  - It provides output stream to writing the data as a response to client.

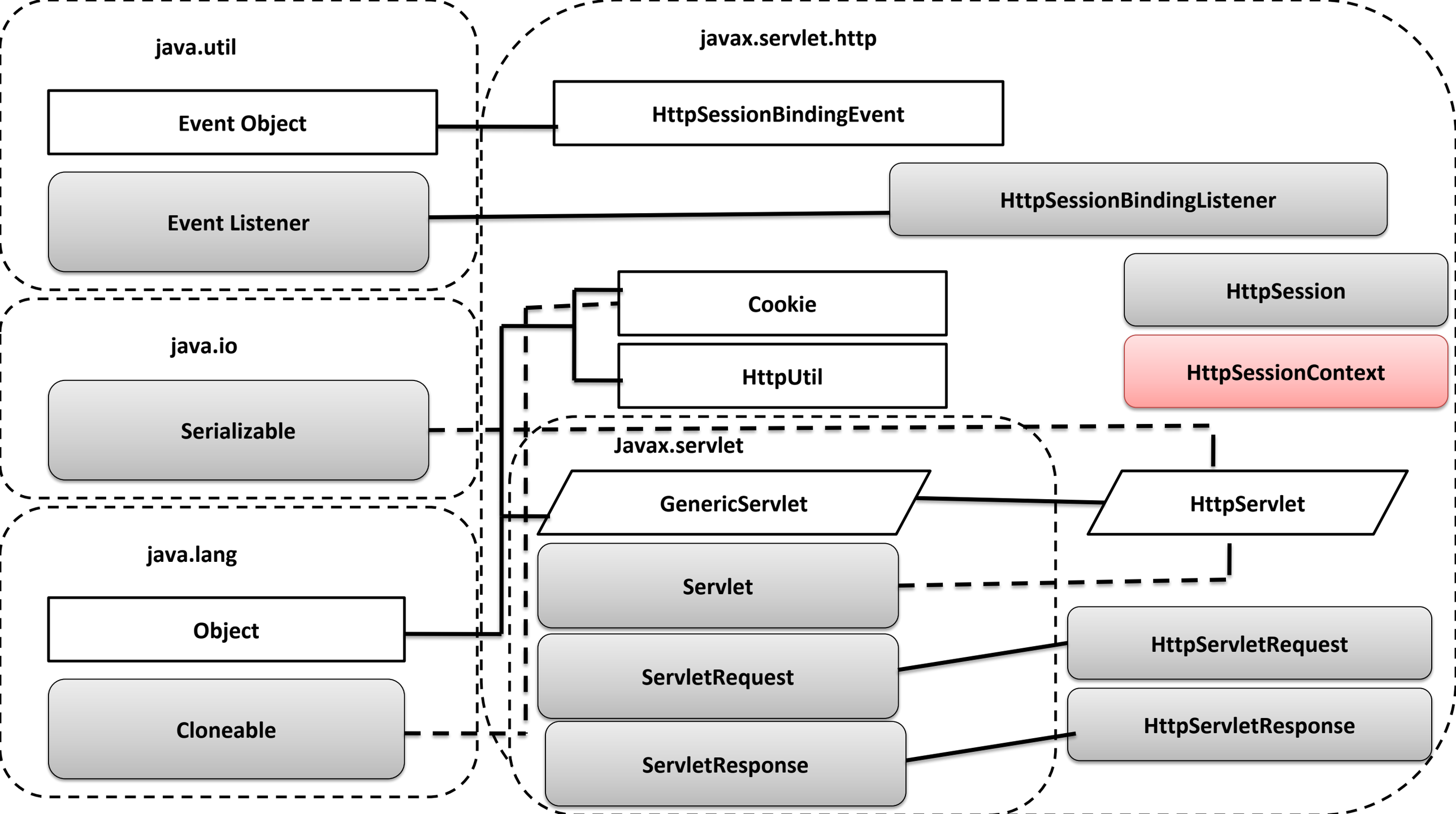
- **SingleThreadModel interface:**
  - It provides the mechanism to make servlet thread safe.


java.io
javax.servlet
InputStream
ServletInputStream
OutputStream
ServletOutputStream
Serializable
Servlet
ServletConfig
ServletContext
ServletRequest
ServletResponse
SingleThreadModel
java.lang
Object
GenericServlet
Excetion
ServletExcetion
UnavailableExcetion

Javax.servlet.http package

- It is having following classes and interfaces.
- Following classes and interfaces commonly used by programmers and it provides the facility to the servlet to work with HttpRequest and HttpResponses. All these classes and interfaces are shown in fig 2.
- **HttpServlet class:**
  - It provides the methods of handling HTTP request and HTTP response.
- **HttpServletRequest interface:**
  - It allows the servlet to read the data from HTTP request.
- **HttpServletResponse interface:**
  - It allows the servlet to write the data to an HTTP response.

- **Cookie class:**
  - It is used to store the state information at client side or on the client machine.
- **HttpSession interface:**
  - It is used to create session and provides the way of reading and writing the session and accessing the information related to session.
- **HttpSessionBindingListener interface:**
  - It informs the object that it is bound or unbound to the session.
- **HttpSessionEvent class:**
  - It wrap ups the session change events.
- **HttpSessionBindingEvent class:**
  - It defines that when session should be bound or unbound with the object value and its attributes.


java.util
Event Object
Event Listener
javax.servlet.http
HttpSessionBindingEvent
HttpSessionBindingListener
java.io
Serializable
Cookie
HttpUtil
HttpSession
HttpSessionContext
Javax.servlet
GenericServlet
HttpServlet
java.lang
Object
Cloneable
Servlet
ServletRequest
ServletResponse
HttpServletRequest
HttpServletResponse

Servlet implementation

- To implement the servlet in the program there are basically 4 ways means we can use any one from following four classes or interfaces to create a servlet program.
  - Servlet interface
  - GenericServlet class
  - HttpServlet class
  - SingleThreadModel interface

Servlet Interface

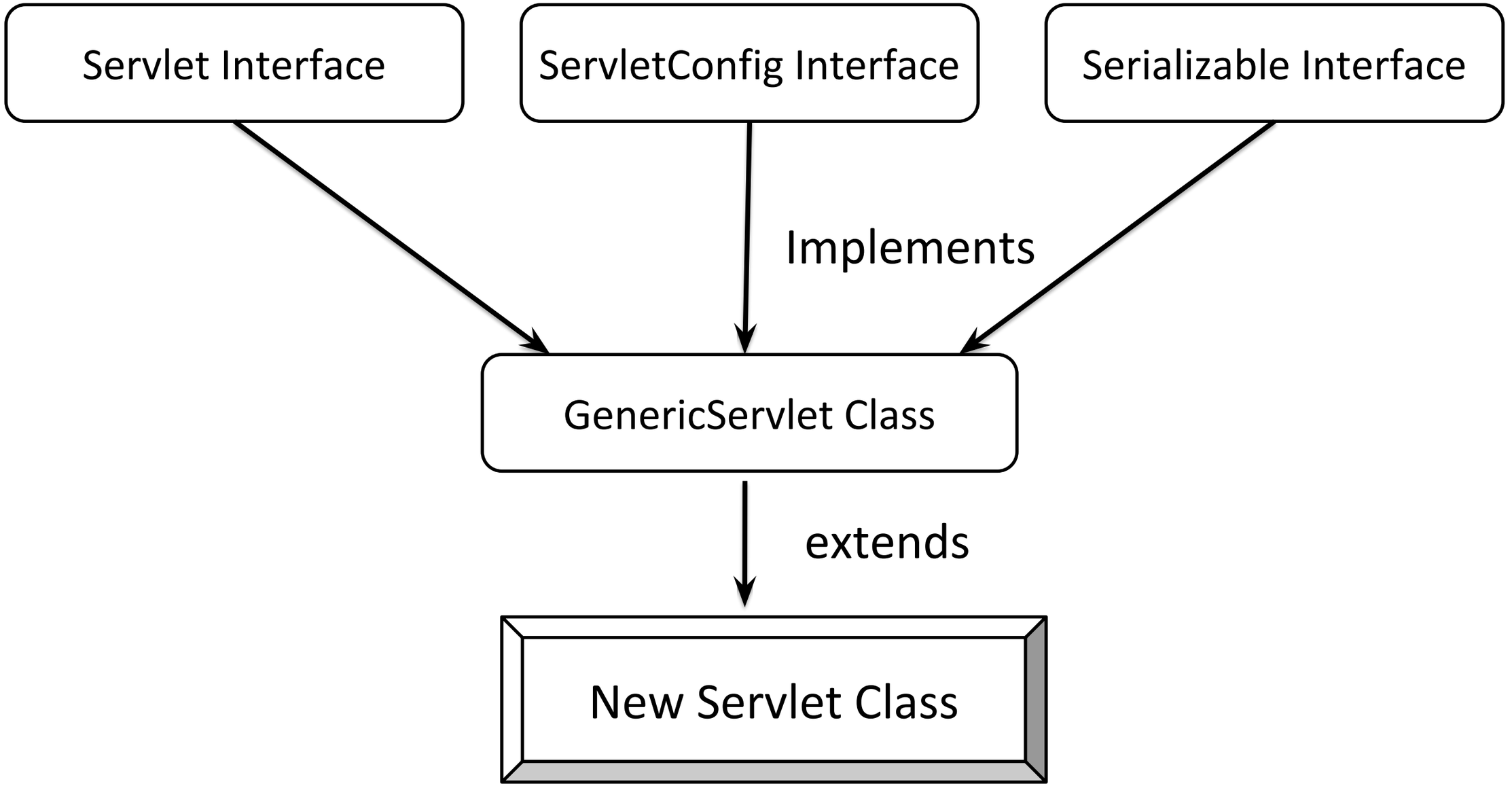
- All the user defined servlets should inherit the servlet interface. It is having following methods which defines the life cycle of servlet.
- Init():
  - It is called by servlet container to initialize the servlet itself. It is called once only when servlet engine loads the servlet. It finishes its work before the service method starts for request and response. It's having one parameter with it, which is an object of ServletConfig interface for start up configuration and initialization parameters which returned through getServletConfig(). If any error occurs with initialization(fatal error) it throws an Unavailable Exception.
- Syntax:
  - Public abstract void init(ServletConfig config)throws ServletException

- getServletConfig():
  - It returns the ServletConfig object which contains initialization parameter and start up configuration of servlet. It is passes as a parameter in the init().
- Syntax:
  - Public abstract void getServletConfig()
- Service():
  - It is used to read the client request and write the response data to the client. It passes two parameters with it first object of ServletRequest interface and second the object of ServletResponse interface. The requests object having information related to client request and parameters provided by the client and response object contains the information which is sent back to the client based on request. This method is called after the initialization is completed. If any error occurs during the request and response or input and output it throws an IOException and ServletException.

- Syntax:
  - Public abstract void service(ServletRequest req,ServletResponse res)throws IOException,ServletException
- getServletInfo():
  - It returns the basic information related to the servlet such as author name, version, copyright etc.
- Syntax:
  - Public abstract String getServletInfo()
- destroy():
  - This method is called by servlet container automatically when the objects or resources like memory, thread etc. needs to be destroyed or servlet stops its execution.
- Syntax:
  - Public abstract void destroy()

GenericServlet Class

- The GenericServlet class belongs to javax.servlet package. It implements the Servlet,ServletConfig and Serializable interface in it. It defines the generic independent servlet it means that it is extended to provide the protocol based things like FTP, SMTP protocols but it is not having HTTP protocol facility. So most probably with web application we need to extend the HttpServlet class.
- GenericServlet class provides the implementation of Servlet interface because in the most of the classes they are required to use service() to handle the requests and responses. It is having most of the methods which are there with the super interfaces except log().


Servlet Interface
ServletConfig Interface
Serializable Interface
Implements
GenericServlet Class
extends
New Servlet Class

Methods of GenericServlet class

- Init()
- Service()
- Destroy():
- Log():
  - GenericServlet class provides the log() method to write the server log file. It is an overloaded method which we can use in two ways, first way log(String str)method writes servlet name and message to web containers log file and other method log(String str,Throwable t), writes servlet name, string str and the exception stack trace of given throwable exception to web containers log file.
- Syntax:
  - Public void log(String str)
  - Public void log(String str, Throwable t)

- getInitParameter():
  - It is used to get the initialization parameter. If the parameter does not exist it returns null.
- Syntax:
  - Public String getInitParameter(String str)
- getInitParameterNames():
  - It returns the name of the initialization parameter names in the form of Enumeration.
- Syntax:
  - Public Enumeration getParameterNames()
- getServletContext():
  - It returns the object of ServletContext.
- Syntax:
  - Public ServletContext getServletContext()

- getServletName():
  - It returns name of the servlet.
- Syntax:
  - Public String getServletName()

Single Thread Model

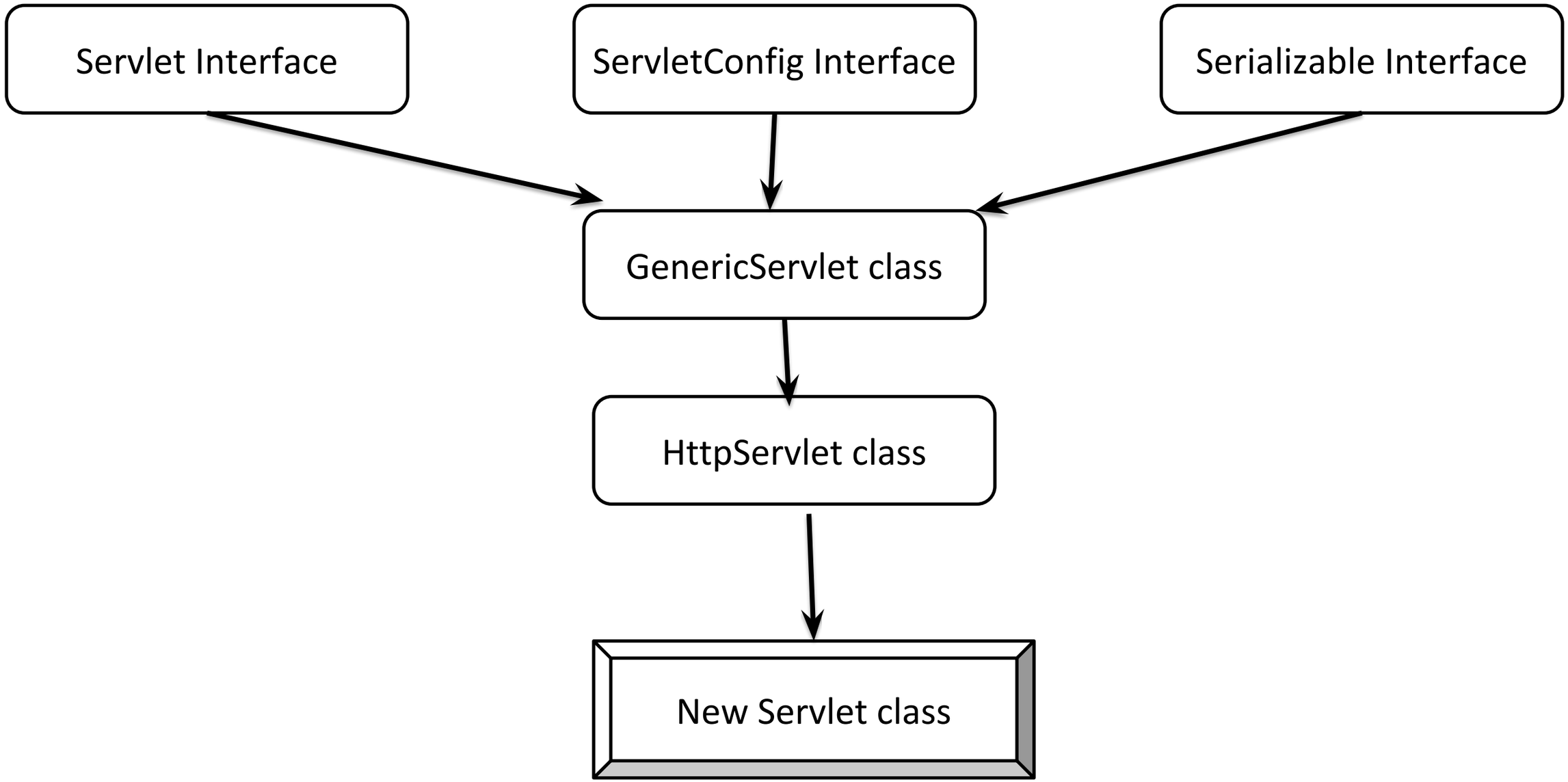
- In the CGI a single instance creates multiple threads in order to process multiple request and response.
- In the past also a single instance of a servlet creates multiple threads for multiple request and response.
- In this process the shared non local variables must be synchronized in the order to prevent data inconsistency.
- But synchronization is implemented at the cost of system performance because thread waits in a queue for the current thread to complete its job.
- Therefore synchronizing the code increases time to perform a single task, it downs the performance of the system.
- In certain cases synchronization may not be appropriate method to implement thread safely. So, in such case we need to implement the SingleThreadModel.

- SingleThreadModel is implemented by SingleThread interface. It ensures that servlets handle only one request at a time.
- This interface has no methods.
- If a servlet implements this interface, you are guaranteed that no two threads will execute concurrently in the servlet's service method.
- The servlet container can make this guarantee by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet.
- Note that SingleThreadModel does not solve all thread safety issues. For example, session attributes and static variables can still accessed by multiple requests on multiple threads at the same time, even when SingleThreadModel servlets are used.

HttpServlet Class

- The HttpServlet class extends GenericServlet class.
- It is commonly used by programmers when developing servlets that receive and process HTTP requests, because it is having HTTP functionality.

Hierarchy of HttpServlet class



Methods of HttpServet class

- doGet():
  - doGet() is called by the server via the service() to handle an HTTP GET request. A GET request allows a client to send form data to a server. With the GET request, the form data is attached to the end of the URL sent by the browser to the server as a query string. The amount of form data that can be sent is limited to the maximum length of the URL.
- Syntax:
  - Public void doGet(HttpServeltRequest req,HttpServletResponse res)throws IOException,ServletException

- doPost():
  - doPost() is called by the server via the service() to handle an HTTP POST request. A POST request allows a client to send form data to a server. With the POST request, the form data is sent to the server separately instead of being appended to the URL. This allows a large amount of form data to be sent.
- Syntax:
  - Public void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException, IOException
- doDelete():
  - doDelete() is called by the server via the service() to handle an HTTP DELETE request. A DELETE request allows a client to remove a document of web page from a server.
- Syntax:
  - Public void doDelete(HttpServletRequest req,HttpServletResponse res)throws ServletException, IOException

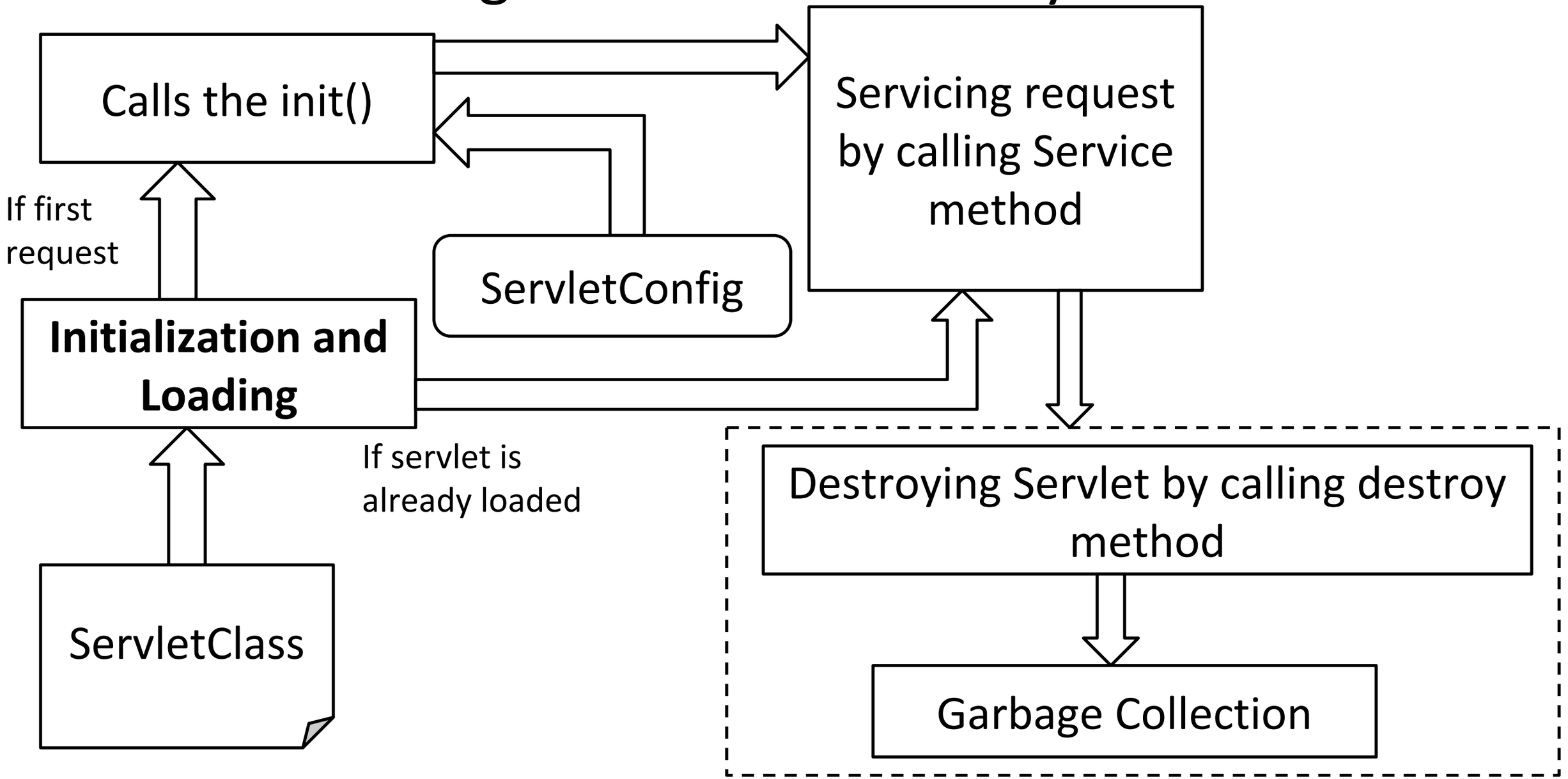
- doOption():
  - doOption() is called by the server via the service() to handle an HTTP OPTIONS request. An OPTIONS request determines which HTTP methods the server supports and sends the information back to the client by way of header.
- Syntax:
  - Public void doOption(HttpServletRequest req,HttpServletResponse res)throws ServletException, IOException
- doPut():
  - doPut() is called by the server via service() to handle an HTTP PUT request. A PUT request allows a client to place a file on the server and is conceptually similar to sending the file to the server via FTP.
- Syntax:
  - Public void doPut(HttpServletRequest req,HttpServletResponse res)throws ServletException, IOException

- doTrace():
  - doTrace() is called by the server via the service() to handle an HTTP TRACE request. A TRACE request returns the headers sent with the TRACE request back to the client. This can be useful for debugging purposes. This method is rarely overridden.
- Syntax:
  - Public void doTrace(HttpServletRequest req,HttpServletResponse res)throws ServletException, IOException

Servlet Life Cycle

- To execute the servlet program with servlet life cycle servlet container calls different methods of servlet interface on different stage.
- Three methods are central to the life cycle of a servlet.
- Init()
- Service()
- Destroy()

Figure of Servlet Life Cycle



- These three methods are implemented by every servlet and are invoked at specific times by server. Let’s take a look how servlet performs its life cycle.
- Assume user enters a URL to a browser. The browser generates an HTTP Request for the URL and send to the server.
- Then, Server receives the HTTP Request. The server maps this request to a particular servlet. The server is dynamically retrieved and loaded into the address space of the server.
- The server invokes the init() of the servlet. This method invoked only when the servlet is first loaded into memory. You will see that initialization parameters can be passed to the servlet so that it may configure itself.
- The server invokes the servlet’s service(), which is called by the HTTP request. You will see that the servlet can read data that has been provided in the HTTP request, and may also formulate an HTTP response for the client.

- The servlet remains in the server’s address space and is available to process any other HTTP requests received from clients. The service() is called for each HTTP request. Finally, the server may decide to unload the servlet from its memory. The servlet calls destroy().
- The memory allocated for the servlet and its objects can then be garbage collected.

Servlet Exceptions

- Java provides various classes for a servlet to handle any exception occurred during execution of a servlet. A servlet throws an exception when it finds difficulties. The javax.servlet defines exceptions.
- ServletException
- UnavailableException

ServletException

- ServletException indicates that a servlet problem has occurred. The general syntax of ServletException class is:
- Public class ServletException extends Exception
- It has following constructors:
- Public ServletException()
- Public ServletException(String msg)
- Public ServletException(String msg, Throwable rootCause)
- Public ServletException(Throwable rootCause)
- getRootCause() method of the ServletException class returns the exception that caused this servlet exception.

UnavailableException class

- UnavailableException extends ServletException.
- It shows that a servlet is temporarily or permanently unavailable.
- When it throws unavailable exception, something is wrong with it, it can not requests until some action is taken.
- A servlet is temporarily unavailable if it can not handle requests momentarily due to some system wide problem.
- Servlet container can safely treat both types of unavailable exceptions in the same way.
- However, treating temporary unavailability effectively makes the servlet container more robust.
- Specifically, the servlet container might block requests to the servlet or filter for a period of time suggested by the exception, rather than rejecting them until the servlet container restarts.

- General syntax:
- Public class UnavailableException extends ServletException
- Constructors:
- UnavailableException()
- UnavailableException(String msg)
- UnavailableException(String msg,int second)
- Methods:
- getUnavailableSecond(): it returns the no of seconds the servlet expects to be temporarily available.
- getServlet(): it returns the servlet that threw this exception or null if the servlet instance was not provided to the constructor.
- isPermanent(): it indicates that the servlet is permanently unavailable or not.

ServletRequest Interface

- ServletRequest interface is a member javax.servlet package. Public interface ServletRequest defines an object to provide client request information to a servlet.
- The servlet container creates a ServletRequest object and passes it as an argument to the **service()**.
- An object of ServletRequest provides data including parameter name, attributes, an input stream and values.
- There are many interfaces that extends ServletRequest interface and can provide additional protocol specific data. For example, HTTP data is provided by HttpServletRequest.

HttpServletRequest Interface

- A public interface HttpServletRequest extends ServletRequest interface and it is a member of javax.servlet.http package.
- It is used to provide request information for HTTP servlets. The servlet container creates an HttpServletRequest object and passes it as an argument to the service method.
- Method of ServletRequest interface and HttpServletRequest interface.
- setAttribute():
  - It is a method of ServletRequest interface and it stores an attribute in this request. Attributes are reset between requests. This methods is most often used in conjunction with RequestDispatcher.
  - Syntax:
    - Void setAttribute(String nm,Object obj)

- getParameter():
  - It is a method of ServletRequest interface and it returns the value of a request parameter as a String, or null if the parameter does not exist.
  - Request parameters are extra information sent with the request. For HTTP servlet, parameters are contained in the query string or posted form data.
  - We can only use this method when we are sure the parameter has only one value.
  - Syntax:
    - String getParameter(String nm)
- getParameterNames():
  - It is a method of ServletRequest interface and it returns Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.
  - Syntax:
    - Enumeration getParameterNames()

- getParameterValues():
  - It is a method of ServletRequest interface and it returns an array of String objects with all the values that the given request parameter has, or null if the parameter does not exist.
  - If the parameter has a single value, a length of array is 1.
  - Syntax:
    - String[] getParameterValues(String nm)
- getAttribute():
  - It is a method of ServletRequest interface and it returns the value of the named attrubute as an object, or null if no attribute of the given name exists.
  - Syntax:
    - Object getAttribute(String nm)

- getAttributeNames():
  - It is a method of ServletRequest interface and it returns an Enumeration containing the names of the attribute available to this request. This method returns an empty Enumeration if the request has no attributes available to it.
  - Syntax:
    - Enumeration getAttributeNames()
- getRemoteHost():
  - It is a method of ServletRequest interface and it returns the fully qualified name of the client or the last proxy that sent the request.
  - If the engine can not resolve the host name to improve performance, this method returns the dotted – string in the form of IP address.
  - Syntax:
    - String getRemoteHost()

- getRemoteAddr():
  - It is a method of ServletRequest interface and it returns the IP address of the client or last proxy that sent the request.
  - Syntax:
    - String getRemoteAddr()
- getHeaders():
  - The HttpServletRequest interface provides to return all the values of the specified request header as an Enumeration of String objects. If the request did not include any header 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.
  - Syntax:
    - Enumeration getHeaders()

- getCookies():
  - The HttpServletRequest interface provides the getCookies() to obtain an array of cookies that are present in the request. This method returns null if no cookies were sent.
  - The cookies are data sent from the client to the server on every request that the client makes. So, getCookies() returns the contents of the Cookie header, passed and stored in an array of Cookie object.
  - Syntax:
    - Cookies[] getCookies()
- getQueryString():
  - The HttpServletRequest interface 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.
  - Syntax:
    - String getQueryString()

- getSession():
  - The HttpServletRequest interface returns the current session associated with this request or if the request does not have a session, creates one.
  - Syntax:
    - HttpSession getSession()

ServletResponse Interface

- A public interface ServletResponse is a member of javax.servlet package.
- It defines an object to assist a servlet in sending a response to the client.
- The container of the servlet creates a ServletResponse object and passes it as an argument to its service method.
- For binary data in a MIME(Multipurpose Internet Mail Extension) body response ServletOutputStream is returned by getOutputStream().
- But for Character data, use the PrintWriter object returned by getWriter().
- The ServletResponse interface is implemented by the server.
- It enables a servlet to create a response for a client.
- There are many interfaces that extend ServletResponse interface and can provide additional protocol specific data.

HttpServletResponse Interface

- A public interface HttpServletResponse extends ServletResponse interface.
- HttpServletResponse interface provides HTTP specific functionalities in sending response.

Methods of ServletResponse & HttpServletResponse

- setContentType():
  - It is a method of ServletResponse interface, which is used to set the content type of the response being sent to the client.
  - The content type may include the type of character encoding used.
  - Syntax:
    - Void setContentType(String type)
- setHeader():
  - It is a method of HttpServletResponse interface and is used to set a response header with the given name and value.
  - If the header had already been set, the new value overwrites the previous one.
  - We can use containsHeader() to test for the presence of a header before setting its value.
  - Syntax:
    - Void setHeader(String nm,String value)

- setStatus():
  - It is a method of HttpServletResponse interface and it sets the status code for HTTP response.
  - This method is used to set the return status when there is no error for example SC_OK or SC_MOVED_TEMPORARILY.
  - But suppose if there is an error, and the caller wishes to invoke an error page defined in the web application.
  - The sendError() should be used instead of setStatus(). The container clears the buffer and sets the location header, preserving cookies and other headers.
  - Syntax:
    - Void setStatus(int sc)

- getWriter():
  - It is a method of ServletResponse interface and returns a PrintWriter object that can be used to send character data to the client. Before calling this method we must call setContentType(String msg) for the character encoding type(charset property).
  - It throws java.io.UnsupportedEncodingException if the charset specified in the setContentType() cannot be used.
  - Either getWriter() or getOutputSteram() may be called to write the body but not both.
  - If the getOutputStream() has already been called for this response then java.lang.IllegalStateException is thrown. But if an intput or output exception occurred then java.io.IOException is thrown.
  - Syntax:
    - PrintWriter getWriter()throws IOExcepiton

- sendRedirect():
  - It is a method of HttpServletResponse interface and it sends a temporary redirect response to the client using the specified redirect location URL.
  - This method can accept relative URLs, and 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 the response has already been committed.
  - After using this method, the response should be considered to be committed and should not be written to.
  - Syntax:
    - Void sendRedirect(String location)throws IOException

- addCookies():
  - It is a method of HttpServletResponse interface it adds the specified cookie to the HTTP response. We can call this method multiple times to set more than one cookie.
  - Syntax:
    - Void addCookie(Cookie cookie)
- encodeURL():
  - It is a method of HttpServletResponse interface and it encodes the specified URL by including the session ID in it. 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 the URLs emitted by a servlet should be run through this method.
  - Otherwise, URL rewriting cannot be used with browsers which do not support cookies.
  - Syntax: String encodeURL(String url)

Session Tracking

- First of all, let's discuss **what is Session?**
- When a user makes a page request to the server, the server creates a temporary session to identify that user. So, when that same user goes to another page on that site the server can recognize that user easily.
- So a session is a temporary small unique connection between a server and the user or client enabling it to identify that user across multiple page request or visit to that site.
- There is still a big question that **why Session Mgt. is needed?**
- As we all know that there are lots of users of the website.
- And they simultaneously use the website at a time. So how can we identify each of them separately?

- Customization:
  - We can allow site visitors to customize view of site. Each user can set different view of our site.
  - For example, you can change the theme, layout, font etc. for your gmail or yahoo account.
- Security:
  - We can allow access to our site according the level of user thus making sure that only some member get to see special content on our site.
  - After logging in we can identify members from non-members by setting an attribute on the user session to some value. so, no need to login again.
- User Behavior:
  - We can control user behavior using session mgt. we can log user behavior like how many ad views have been shown to the user.

- HTTP is a stateless protocol. It provides no built in way for a server to recognize that a sequence of request all originated from the same user.
- Robust web application need to interact back and forth with the user, remembering information about the user between requests.
- There are two important activities in session mgt:
- **Tracking the identity of a user:**
  - As the user makes multiple requests to the same web application over a period of time, we need a mechanism that links these requests together.
  - Effectively, this means that we need to associate each request with a client identifier, so that we can identify requests from the same user.
- **Managing user state:**
  - Since there is often data associated with each request, we will need a way to associate the request data with the user that made the request, and a way to preserve that data across request.
- So the ability to associate a request with the client that made the request is known as maintaining a session.

- We must preserve the identity of the client and the data associated with the client across requests.
- Since, HTTP is a stateless protocol, just how do web applications manage to keep track of users?
- URL – rewriting
- Hidden – form fields
- Cookies

URL Rewriting

- URL rewriting is based on the idea of inserting a unique ID(generated by the server) in each URL of the response from the server.
- That is, while generating the response to the first request the server inserts this id in each URL.
- When the client submits a request to one such URL, the browser sends this ID back to the server.
- The server can therefore identify the id with all requests.
- Advantages of URL rewriting are user remains anonymous and they are universally supported.
- But the major disadvantages are it is tedious to rewrite all the URLs and it only work with dynamically created documents.

Hidden Form Field

- Other way to support session tracking is to use hidden form fields. As the name implies, these are fields added to an HTML form that are not displayed in the client's browser.
- They are sent back to the server when the form that contains them is submitted. Hidden form fields define constant variables for a form.
- To a servlet receiving a submitted form, there is no difference between a hidden form field and a visible field.
- Hidden fields do not affect the appearance of the page that is presented to the user. Instead, they store fixed names and values that are sent unchanged to the server, regardless of user input.
- Hidden fields are typically used for three purpose.

- For tracking session as users moved around within site.
- Hidden fields are used to provide predefined input to a server – side program when a variety of static HTML pages act as front ends to the same program on the server.
- Hidden form fields are used to store background information in pages that are dynamically generated.
- **Advantages of hidden form field are that it is universally supported and allow anonymous users. But major problems with hidden form field are:**
- It only works for a sequence of dynamically generated forms.
- It breaks down with static documents, emailed documents, and bookmarked documents.
- There are no browsers shutdowns.

Session Tracking Using Cookies

- Cookies provide a better alternative to explicit URL rewriting, because cookies are not sent as query strings but are exchanged within the bodies of HTTP requests and response.
- Since there is no need to rewrite URLs, session handling via cookies does not depend on whether the content is static or dynamic.
- All modern browsers can recognize and receive cookies from web servers, and then send them back along with requests. However, there is a limitation with cookies. All browsers allow users to disable this functionality, which leads to browsers not recognizing cookies.
- This is because cookies have a bad press – they have sometimes been used to gather information about consumers without their knowledge. Given this, it is worth simultaneously supporting an alternative technique such as URL rewriting.

- A cookie is a string sent via the HTTP request and HTTP response headers. A cookie has following parameter.
- Name – Name of Cookie
- Value – Value of Cookie
- Comment – comment explaining the purpose of the cookie
- Max – Age – Maximum age of the cookie (a time in seconds after which the client should not send the cookie back to the server)
- Domain – domain to which the cookie should be sent
- Path – path to which the cookie should be sent
- Secure – specifies if the cookie should be sent securely via HTTPS
- Version – version of the cookie protocol. Version 0 is meant for the original Netscape version of cookies. Version 1 meant for cookies standardized via RFC 2109.

- In order to send a cookie to a client, a server creates a cookie header and attaches it to the HTTP response. The client receives the request and extracts the cookie data is usually stored in a file on the client's system.
- Some methods:
- setComment():
  - It is used to set 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.
  - Syntax: public void setComment(String msg)
- setDomain():
  - It is used to set the domain within which this cookie should be presented.
  - The form of domain name is specified by RFC 2109. A domain name begins with a dot(.abc.com) and means that the cookie is visible to server in a specified domain name system(DNS) zone. By default, cookies are only returned to the server that sent them.
  - Syntax: public void setDomain(String domain)

- setMaxAge():
  - It used to set the maximum age of the cookie in seconds. 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.
  - Syntax: public void setMaxAge(int expiry)
- setPath():
  - If 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, "/cookie", which makes the cookie visible to all directories on the server under "/cookie". If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.

  - This method can be used to specify something more general.
  - For example, cookieObj.setPath(“/cookies”) specifies that all the pages on the server should receive the cookie.
  - Note that the path specified must include the cookies directory.
  - Syntax: public void setPath(String url)
- setSecure():
  - It is used to set boolean flag, which 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.
  - Syntax: public void setSecure(boolean flag)
- setVersion():
  - It is used to set the version of the cookie protocol this cookie complies with. version 0 complies with original Netscape cookie specification. Version 1 complies with RFC 2109.
  - Syntax: public void setVersion(int version)

- setValue():
  - It assigns a new value to a cookie after the cookie is created. If you use a binary value. You may want to use base64 encoding.
  - With version 0 cookies, values should not contain white space, brackets, parenthesis, equal signs, commas, double quotes, slashes, question marks, at signs, colons and semicolons.
  - Empty values may not behave the same way on all browsers.
  - Syntax: public void setValue(String value)
- **Get Methods:**
- getComment():
  - It returns the comment as a string, which describes the purpose of this cookie, or null id the cookie has no comment.
  - Syntax: public String getComment()

- getDomain():
  - It returns the domain name set for this cookie. The form of the domain name is set by RFC 2109.
  - Syntax: public String getDomain()
- getMaxAge():
  - It returns the maximum age of the cookie, specified in seconds, by default -1 indicating the cookie will persist until browser shutdown.
  - Syntax: public int getMaxAge()
- getName():
  - It returns the name of the cookie. The name cannot be changed after creation.
  - Syntax: public String getName()
- getPath():
  - It returns the path on the server to which browser returns this cookie. The cookie is visible to all sub paths on the server.
  - Syntax: public String getPath()

- getSecure():
  - It returns true if the browser is sending cookies only over a secure protocol, or false if the browser can sent cookies using any protocol.
  - Syntax: public boolean getSecure()
- getValue():
  - It returns the value of cookie.
  - Syntax: public String getValue()
- getVersion:
  - It returns the version of the protocol this cookie compiles with.
  - Syntax: public int getVersion()

Introduction of Servlet Session API

- The java servlet framework provides an implementation of the Session API to solve various problems. Such as,
- How to transmit the session id from server to client and vice versa?
- How to select secure session id?
- How to store associated object/data with each session physically and check for session expiry?
- Generally, transmitting the session id to and from the server is handled via HTTP cookies, though other means are possible and could be chosen by servlet implementations for example, URL rewriting contains session id as a parameter. But we can not solve the problem using this way, for that we have to use Session API.

- Session management is also done using the APIs provided by the application server to identify the user across multiple page requests.
- Here note that every server has its own set of API for session mgt.
- In java servlet servletAPI 2.2 is used and which is supported by almost all of the java application servers.

Session tracking using session API

- The servlet API provides several methods and classes specifically designed to handle short term session tracking on behalf of servlets.
- In other words, servlets have built – in session tracking.
- The HttpSession interface provides core session – management functionality.
- Servlet use HttpSession object to store or relative information about the user.

HttpSession Interface

- A public interface HttpSession is a member of javax.servlet.http.
- It 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.
- 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.
- An implementation of HttpSession represents the server's view of the session.
- The server considers a session to be new until it has been joined by the client until the client joins the session, the isNew() returns true. A true value indicate below cases:
  - The client does not yet know about session
  - The session has not yet begun
  - The client chooses not to join the session.

- This case will occur if the client supports only cookies and chooses to reject any cookies sent by the server.
- If the server supports URL rewriting, this case will not commonly occur.
- If the client chooses not to join session, getSession() will return different session on which 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.
- HttpSession defines methods which store following types of data:
- Standard session properties, like an identifier for the session, and the context for the session.
- Application layer data accessed using this interface and stored using a dictionary – like interface.

- Several methods of HttpSession interface are given below.
- All these methods throw an IllegalStateException if an attempt is made to access session data after the session has been invalidate.
- Methods of HttpSession interface:
- setMaxInactiveInterval():
  - It sets the maximum interval between requests that this session will be kept by the host server.
  - Syntax: void setMaxInactiveInterval(int interval)
- **Get methods:**
- getCreationTime():
  - It returns the time at which this session representation was created, in milliseconds since midnight, Jan. 1, 1970 UTC.
  - Syntax: long getCreationTime()

- getId():
  - It returns the identifier assigned to this session which is known as Session ID.
  - An HttpSession’s Session ID is a unique string that is created and maintained by HttpSessionContext.
  - Syntax: String getId()
- getLastAccessedTime():
  - It returns the last time the client sent a request carrying the identifier assigned to the session. Time is expressed as milliseconds since midnight, Jan. 1, 1970 UTC.
  - Application level operations, such as getting or setting a value associated with the session, does not affect the access time.
  - This information is particularly useful in session management policies. For example,
  - A session manager could leave all sessions which have not been used in a long time in a given context.
  - The session can be stored according to age to optimize some task.
  - Syntax: long getLastAccessedTime()

- getMaxInactiveInterval():
  - It 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(). A negative time indicates the session should never timeout.
  - Syntax: int getMaxInactiveInterval()
- getValue():
  - It returns the object bound to the given name in the session's application layer data. Returns null if there is no such binding.
  - Syntax: Object getValue(String name)
- getValueNames():
  - It returns an array of String objects specifying the names of all the objects bound to this session.
  - Syntax: String[] getValueNames()

- Invalidate():
  - It invalidates this session and unbinds or removes it from the context.
  - Syntax: void invalidate()
- isNew():
  - It returns true if the server created the session and it has not yet been accessed by the client. A session is considered to be “new” if it has been created by the server, but the client has not yet acknowledged joining the session. For example, if the server supported only cookie – based sessions and the client had completely disabled the use of cookies, then calls to HttpServletRequest.getSession() would always return “new” sessions.
  - Syntax: boolean isNew()
- putValue():
  - It binds the specified object into the session’s application layer data with the given name. Any existing binding with the same name is replaced. New (or existing) values that implement the HttpSessionBindingListener interface will call its valueBound().
  - Syntax: void putValue(String name,Object value)

- removeValue():
  - It 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.
  - Syntax: void removeValue(String name)

HttpSessionContext interface

- HttpSessionContext provides access to all of the currently active sessions on the server. This is to be useful for servlets that weed out inactive sessions, display statistics, or otherwise share information. A servlet obtains an HttpSessionContext object from the getSessionContext() of HttpSesison.
- getIds():
  - It returns an Enumeration that contains the session IDs for all the currently valid sessions in this context. It returns an empty Enumeration if there are no valid sessions. The session IDs returned by getIds() should be held as a server secret because any client with knowledge of another client's session ID can, with a forged cookie or URL, join the second client's session.
  - Syntax: Enumeration getIds()

- getSession():
  - It returns the session associated with the given session identifier. A list of valid session IDs can be obtained from the getIds().
  - Syntax: HttpSession getSession(String sessionId)

Servlet Collaboration

- A servlet can call another servlet's public methods directly, if the two servlets run within the same server. To call another servlet's public methods directly, we must know the name of the servlet that we want to call. Gain access to that servlet's Servlet object. Call the servlet's public method.
- Servlets running together in the same server have several ways to communicate with one another. This communication can be called as sort servlet collaboration. The servlets which are collaborating can pass the shared information directly from one servlet to another. They can do it through method invocations.
- A powerful feature of the servlet API is request delegation. A single client's request can pass through many servlets and/or to any other resources in the web application. The entire process is done on the server – side.

- Request delegation does not require any action from a client or extra information sent between the client and server. Through the javax.servlet.RequestDispatcher object, the required delegation is available.
- An object implementing the RequestDispatcher interface may be obtained from the ServletContext via the following methods:
- getRequestDispatcher(String arg)
  - This method takes a String arg describing a path within the scope of the ServletContext. This path must be relative to the root of the ServletContext. This path is used to look up a servlet, wrap it with a RequestDispatcher object, and return it.
- getNamedDispatcher(String arg)
  - This method takes a String arg indicating the name of a servlet known to the ServletContext. If a servlet is known to the ServletContext by the given name, it is wrapped with a RequestDispatcher object and returned.

RequestDispatcher Interface

- A RequestDispatcher is a public interface which defines an object that receives requests from the client and sends them to any resource such as a servlet, HTML file, or JSP file on the server.
- The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server – resources located at a particular path or given by a particular name.
- This interface is intended to wrap servlets, but a servlet container can create RequestDispatcher objects to wrap any type of resource.
- There are two methods in the interface, first one is **forward()** and second one is **include()**.
- To use a request dispatcher, a developer needs to call either include or forward method of the **RequestDispatcher** interface using the request and response arguments that were passed in via the service() of the **servlet** interface.

Methods of RequestDispatcher interface

- Forward():
  - It forwards a client request to another resources such as, servlet, HTML, JSP etc.). It allows a servlet to serve as a “request processor”. Forward may not be called if data has been sent to the client.
  - Forward() calls the response.reset() to clear the output buffer. Query parameters are added to the original query parameters. The new URI values are based on the RequestDispatcher URI. So, getRequestURI(), getServletPath() and getPathInfo() will reflect the request dispatcher URI.
  - The forward() can be used if the servlet has not already opened a PrintWriter or ServletOutputStream back to the client machine.
  - If an output stream has been created, use the include() instead of forward(). The request and response must be either the same objects that were passed to this servlet’s service(), or ServletRequestWrapper or ServletResponseWrapper subclass instances that wrap them.
  - Syntax: public void forward(ServletRequest,ServletResponse)throws IOException, ServletException

- Include():
  - It includes the content of a resource like servlet, JSP, HTML etc in the response.
  - In essence, this method enables programmatic server – side includes.
  - This method is used to include some content to the response after the response has been initiated by opening a PrintWriter or ServletOutputStream back to the client machine.
  - The request and response must be either the same objects that were passed to this servlet's service(), or ServletRequestWrapper or ServletResponseWrapper subclass instances that wrap them.
  - Syntax: public void include(ServletRequest,ServletResponse)throws ServletException, IOException

ServletConfig Interface

- ServletConfig interface belongs to javax.servlet package.
- It is used by servlet container to get the configuration and initial parameter information of servlet.
- Its object passed with init method to provide the basic information mentioned in the <init-param> element of the Deployment Descriptor(web.xml) for that particular servlet.
- If no initialization parameters are provided then a default ServletConfig object is built by the Container and passed to the servlet.

- Syntax:

```
<servlet>
    <servlet-name>servlet name</servlet-name>
    <servlet-class>class file name</servlet-class>
    <init-param>
        <param-name>parameter 1 name</param-name>
        <param-value>parameter 1 value</param-value>
    </init-param>
    <init-param>
        <param-name>parameter 2 name</param-name>
        <param-value>parameter 2 value</param-value>
    </init-param>
    …….
</servlet>
```

- It is implemented with the servlet through GenericServlet interface. When we extend GenericServlet or HttpServlet with our user defined servlet it provides configuration. Following methods are used with ServletConfig interface to get the configuration and initial parameters.
- Get methods:
- getInitParameter():
  - It is used to get the initialization parameter if the parameter does not exist it returns null.
  - Syntax: public String getInitParameter(String nm)
- getInitParameterNames():
  - It reurns the name of the initialization parameter names in the form of Enumeration.
  - Syntax: public Enumeration getParameterNames()

- getServletContext():
  - It returns the object of ServletContext.
  - Syntax: public ServletContext getServletContext()s

ServletContext Interface

- ServletContext interface belongs to javax.servlet package.
- It provides methods to communicate servlet with the servlet container.
- For example fetching the MIME type of a file, writing to a log file, calling request dispatcher, etc.
- <context – param> element can be used by the container to build the ServletContext objects.
- These ServletContext objects will be unique per JVM per web application.

- Syntax:

```
<context-param>
    <param-name>parameter 1 name</param-name>
    <param-value>parameter 1 value</param-value>
</context-param>
```

- It is implemented with each servlet through ServletConfig object because every ServletConfig object contains a ServletContext object as well.
- Both the objects are created by the Servlet container and the ServletConfig object is passed to every servlet while its initialization which we hava seen with ServletConfig interface.

- Methods:
- setAttribute():
  - It is used to bind an object to a given attribute name in the servlet context.
  - Syntax: public void setAttribute(String atrNm,Object objVal)
- getAttribute():
  - It is used to return the attribute of servlet context with given name, it returns null if that attribute name has not been set.
  - Syntax: public Object getAttribute(String atrNm)
- getAttributeNames():
  - It is used to return the Enumeration of attribute names with servlet context. It returns empty Enumeration if no attribute names is there.
  - Syntax: public Enumeration getAttributeNames()

- getContext():
  - It is used to return a ServletContext object that corresponds to a URL on the server.
  - Syntax: public ServletContext getContext(String URLPath)
- getMajorVersion():
  - It returns the major version of servlet API in the integer format.
  - Syntax: public int getMajorVersion()
- getMimeType():
  - It returns MIME type of the specified file, or null if the MIME type is not known.
  - Syntax: public String getMimeType(String file)
- getMinorVersion():
  - It returns the minor version of the Servlet API that this servlet container supports.
  - Syntax: public int getMinorVersion()

- getRealPath():
  - It returns a String containing the real path for a given virtual path, or it returns null if the servlet container cannot translate the virtual path to a real path for any reason.
  - Syntax: public String getRealPath(String path)
- getRequestDispatcher():
  - It returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. This method returns null if the ServletContext cannot return a RequestDispatcher.
  - Syntax: public RequestDispatcher getRequestDispatcher(String path)
- getResource():
  - This method allows the servlet container to make resource available to servlets from any source. It returns null if no resource is mapped to the pathname.
  - Syntax: public URL getResource(String path)throws MalformedURLException

- getResourceAsStream():
  - It returns the resource located at the named path as an InputStream object. This method returns null if no resource exists at the specified path.
  - Syntax: public InputStream getResourceAsStream(String path)
- getServerInfo():
  - It returns the name and version of the servlet container on which the servlet is running.
  - Syntax: public String getServerInfo()
- Log():
  - It is used to write the log event of servlet to the log file.
  - Syntax: public void log(String msg)
    public void log(String msg, throwable exception)
- removeAttribute():
  - It is used to remove the given attribute with the parameter from the servlet context.
  - Syntax: public void removeAttribute(String atrNm)