There are three things that a **servlet container** does to service a request for a servlet:

* **Creating a request object** and populate it with information that may be used by the invoked servlet.
  + such as parameters, headers, cookies, query string, URI, etc.
  + A request object is an instance of the javax.servlet.ServletRequest interface or the javax.servlet.http.ServletRequest interface.
* **Creating a response object** that the invoked servlet uses to send the response to the web client.
  + A response object is an instance of the javax.servlet.ServletResponse interface or the javax.servlet.http.ServletResponse interface.
* **Invoking the service method of the servlet**, passing the request and response objects.
  + Here the servlet reads the values from the request object and writes to the response object.

Catalina is most popular servlet container **code** of Tomcat. Catalina consist of two main modules:

* **The connector** connect a request with the container.
  + Its job is to construct a request object and a response object for each HTTP request it receives.
  + It then passes processing to the container.
* **The container** receives the request and response objects from the connector and is responsible for invoking the servlet's service method. Before that it
  + Must load the servlet.
  + Authenticate the user
  + Update the session for that user. Etc.

The Socket class represents a "client" socket(browser in servlet technology), and server (or tomcat) is represented by server socket class.

The role of a server socket is to wait for connection requests from clients. Once the server socket gets a connection request, it creates a Socket instance to handle the communication with the client.

HttpServer is typical Server Socket which waits for client to connect.

* Once there is a connection request(via typing a url in browser) it gets an instance of the Socket class.
* it gets the input and output stream of socket object
* Create the request object and gets uri by calling it's parse method.
  + Uri is relative path of the resource from webroot directory.
  + This uri is used by response object later to get the file provided in URL.
* Create the response object by passing output stream.
* Sets request object to response object.
  + Because response object has to get the uri, earlier evaluated by request object.
* Finally called the sendStaticResource of response object.
  + Which sends the content of the resource to the browser as raw data.
  + This is achieved by putting the file contents into output stream.

Main responsibility of Request object is

* parses the raw data in the HTTP request and
* extracts URI from it
* stores the URI in the uri variable.
  + Later response object invoke getUri method to get the URI of the HTTP request.

To understand how the parse and parseUri methods work, we need to know the structure of an HTTP request.

Here, we are only interested in the first part of the HTTP request - the request line.

A request line begins with a

* method token,
* followed by the request URI then,
* the protocol version, and
* ends with carriage-return linefeed (CRLF) characters.

Elements in a request line are separated by a space character

For Ex, request for the index.html file using the GET method is :

GET /index.html HTTP/1.1

and uri is :

/index.html

Response code-flow:

* HttpServer object pass a Request object and output stream to the Response object.
* Response object uses Request object to get the path of static resource then
* constructs a java.io.FileInputStream object by passing the File object.
* invokes the read method of the FileInputStream and writes the byte array to the OutputStream output.
* content of the static resource is sent to the browser as raw data.

A fully-functional servlet container does the following for each HTTP request for a servlet:

* When the servlet is called for the first time, load the servlet class and call the servlet's init method (once only)
* For each request, construct an instance of javax.servlet.ServletRequest and an instance of javax.servlet.ServletResponse.
* Invoke the servlet's service method, passing the ServletRequest and ServletResponse objects.
* When the servlet class is shut down, call the servlet's destroy method and unload the servlet class.

The first servlet container (Application1) runs very simple servlets and does not call the servlets' init and destroy methods. Instead, it does the following:

* Wait for HTTP requests.
* Construct a ServletRequest object and a ServletResponse object.
* If the request is for a static resource, invoke the process method of the StaticResourceProcessor instance, passing the ServletRequest and ServletResponse objects.
* If the request is for a servlet, load the servlet class and invoke the service method of the servlet, passing the ServletRequest and ServletResponse objects.

Note : In this servlet container, the servlet class is loaded every time the servlet is requested.

**if** (request.getUri().startsWith("/servlet/")) {

ServletProcessor1 processor = **new** ServletProcessor1();

processor.process(request, response);

}

**else** {

StaticResourceProcessor processor = **new** StaticResourceProcessor();

processor.process(request, response);

}

**Security threat:**

There is a serious security threat in the first application. In the ServletProcessor1 class's process method because of exposing request and response class to Servlate (user).

servlet.service((ServletRequest) request, (ServletResponse) response);

Servlet programmers can downcast the ServletRequest/ServletResponse instances back to Request/Response respectively and call their public methods -parse() sendStaticResource().

**public** **void** service(ServletRequest request, ServletResponse response){

Request request1 = (Request)request;

request1.parse();

}

**Solution:**

The security is protected by adding two façade classes: RequestFacade and ResponseFacade.

RequestFacade implements the ServletRequest interface and is instantiated by passing a Request instance that it assigns to a ServletRequest object reference in its constructor.

Implementation of each method in the ServletRequest interface invokes the corresponding method of the Request object.

Again, the ServletRequest object itself is private and cannot be accessed from outside the class.

Now, Instead of upcasting the Request object to ServletRequest and passing it to the service method, we construct a RequestFacade object and pass it to the service method.

Servlet programmers can still downcast the ServletRequest instance back to RequestFacade, however they can only access the methods available in the ServletRequest interface. Now, the parseUri method is safe.