What is Servlet ?

The primary purpose of the Servlet specification is to define a robust mechanism for **sending content to a client as defined by the Client/Server model**. It is **used to create a web application** (resides at server side and generates a dynamic web page).

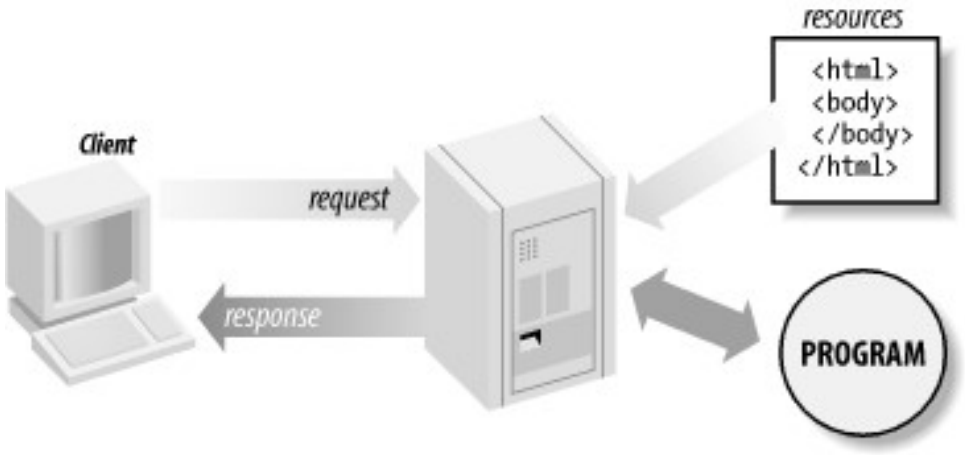
Advantages:

1. **Better performance**: because it creates a thread for each request, not process.
2. **Portability**: because it uses Java language.
3. **Robust**: JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure**: because it uses java language.

The HTTP Request/Response (Client/Server) Model

A client, typically a web browser, sends a request for a resource to a server, and the server sends back a response corresponding to the resource (or a response with an error message if it can't process the request for some reason).

A resource can be a number of things, such as a simple HTML file returned verbatim to the browser or a program that generates the response dynamically.



The HTTP Request/Response Model – Important Facts

Three important facts you need to be aware of :

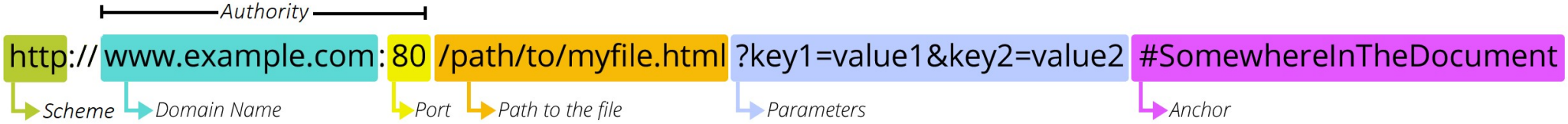
- ✓ **HTTP is a stateless protocol**. This means that the server doesn't keep any information about the client after it sends its response, and therefore **it can't recognize that multiple requests from the same client** may be related.

- ✓ **Web applications can't easily provide the kind of immediate feedback typically found in standalone GUI applications such as word processors or traditional client/server applications.** Every interaction between the client and the server requires a request/response exchange. Performing a request/response exchange when a user selects an item in a list box or fills out a form element is usually too taxing on the bandwidth available to most Internet users.

- ✓ There's nothing in the protocol that tells the server how a request is made; consequently, **the server can't distinguish between various methods of triggering the request on the client.** For example, HTTP doesn't allow a web server to differentiate between an explicit request caused by clicking a link or submitting a form and an implicit request caused by resizing the browser window or using the browser's Back button. In addition, **HTTP doesn't contain any means for the server to invoke client specific functions**, such as going back in the browser history list or sending the response to a certain frame. Also, **the server can't detect when the user closes the browser**

Requests in Detail - URL

A user sends a request to the server by clicking a link on a web page, submitting a form, or typing in a web page address in the browser's address field. To send a request, the **browser needs to know which server to talk to and which resource to ask for**. This information is specified by an **HTTP Uniform Resource Locator (URL)**:



- ✓ **Scheme** - Indicates the protocol that the browser must use to request the resource. Usually for websites the protocol is HTTPS or HTTP (its unsecured version). Addressing web pages requires one of these two
- ✓ **Domain Name** - Indicates which Web server is being requested. Usually this is a domain name, but an IP address may also be used.
- ✓ **Port** - Indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory.
- ✓ **Path to the file** - Identifies the resource that the client is requesting. In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is mostly an abstraction handled by Web servers without any physical reality.

Requests in Detail - Structure

A correctly composed HTTP request contains the following elements:

- ✓ A Request line - The request line starts with the **request method** name, followed by a resource identifier and the protocol version used by the browser. Example :

```
GET /index.html HTTP/1.1
```

  The request line specifies the GET method and asks for the resource named /index.html to be returned using the HTTP/1.1 protocol version.

- ✓ **Request header**(s) - Provide additional information the server may use to process the request.
- ✓ **A Message body** – It is included only in some types of requests if needed.

Requests in Detail – Important Request Method

1. **GET** - The most commonly used request method is named GET. As the name implies, a GET request is used to retrieve a resource from the server. **It's the default request method**, so if you type a URL in the browser's address field, or click on a link, the request is sent as a GET request to the server.
2. **OPTIONS** - An OPTIONS request should return data describing what other methods and operations the server supports at the given URL.
3. **HEAD** - The HEAD method is used to get a response with all headers generated by a GET request but without the body. It can make sure a link is valid or to see when a resource was last modified or to know file size before downloading a large file
4. **POST** - A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
5. **PUT** - Similar to POST, PUT requests are used to send data to the API to update or create a resource.
6. **PATCH** - Similar to POST and PUT. The difference with PATCH is that you only apply partial modifications to the resource.
7. **TRACE** - The TRACE method is used for testing the communication between the client and the server. The server sends back the request message, exactly as it received it, as the body of the response.
8. **DELETE** - The DELETE method is used to delete the resource identified by the URI.

Requests in Detail – POST vs GET vs PUT vs DELETE vs PATCH

Most commonly used request methods are POST, GET, PUT and DELETE which are similar to CRUD operations:

1. Create – POST
2. Read – GET
3. Update – PUT
4. Delete – DELETE

PATCH - Submits a partial modification to a resource. If you only need to update one field for the resource, you may want to use the PATCH method.

Requests in Detail – Request Header

The request headers provide additional information the server may use to process the request. Each request header is made up of a name and a value.

The HTTP protocol specifications define the standard set of headers, and describe how to use them correctly. Request can have more headers, which are not part of the HTTP/1.1 or HTTP/1.0 specifications.

Please refer specification - https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for full list of headers.

Requests in Detail – Message Body

A message body is the one which **carries** the actual HTTP request data, **the actual content of the message**. Message bodies are appropriate for some request methods and inappropriate for others.

If it is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated. Please find every content type here - https://www.iana.org/assignments/media-types/media-types.xhtml

Message bodies are appropriate for some request methods and inappropriate for others. For example, a request with the POST method, which sends input data to the server, has a message body containing the data. A request with the GET method, which asks the server to send a resource, does not have a message body.

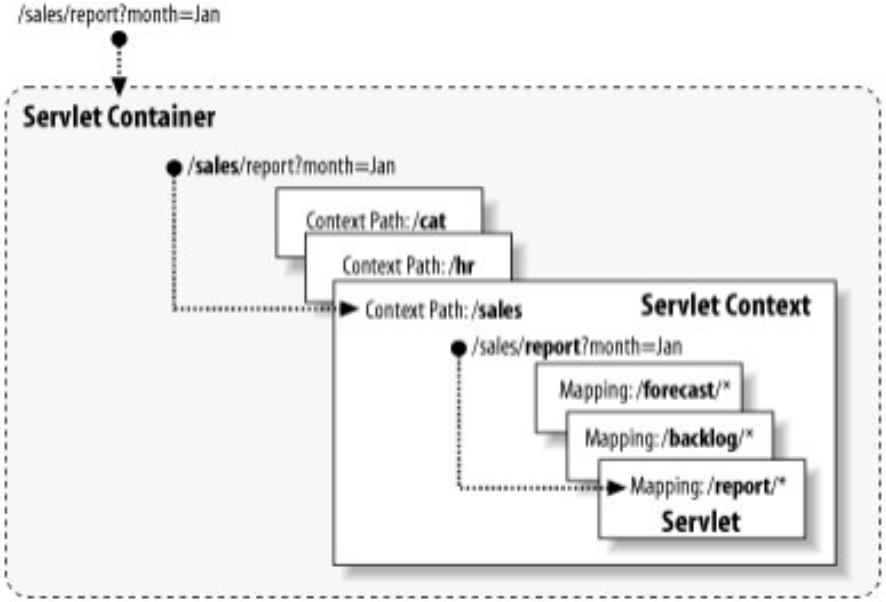
Response

- ✓ Except request method, response also **supports headers, message body** like request.
- ✓ Besides, response **always has HTTP status code** - three-digit integer code **indicates** whether a specific HTTP **request has been successfully completed**. The first digit of the status-code defines the class of response.
  - 1xx (Informational): The request was received, continuing process
  - 2xx (Successful): The request was successfully received, understood, and accepted
  - 3xx (Redirection): Further action needs to be taken in order to complete the request
  - 4xx (Client Error): The request contains bad syntax or cannot be fulfilled

Servlet Containers

A servlet container is the connection between a web server and the servlets. It is responsible for loading and invoking those servlets when the time is right. The container typically loads a servlet class when it receives the first request for the servlet, gives it a chance to initialize itself, and then asks it to process the request. Subsequent requests use the same, initialized servlet until the server is shut down. The container then gives the servlet a chance to release resources and save its state.

The servlet container is responsible for mapping an incoming request to a servlet registered to handle the resource identified by the URI and passing the request message to that servlet. After the request is processed, it's the container's responsibility to convert the response created by the servlet into an HTTP response message and send it back to the client.



Servlet Life Cycle

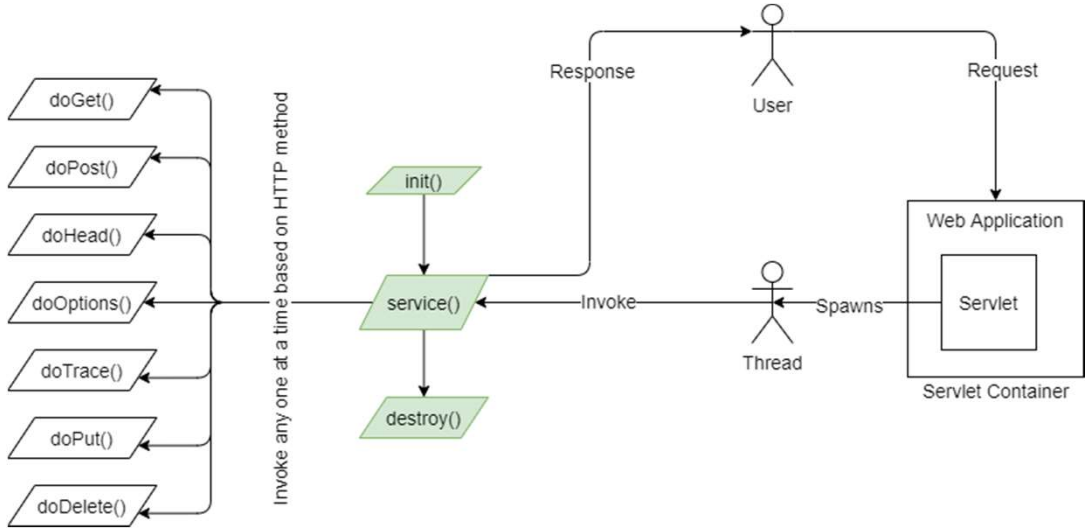
1. **Loading of Servlet** - When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.

2. **Creating instance of Servlet** - Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

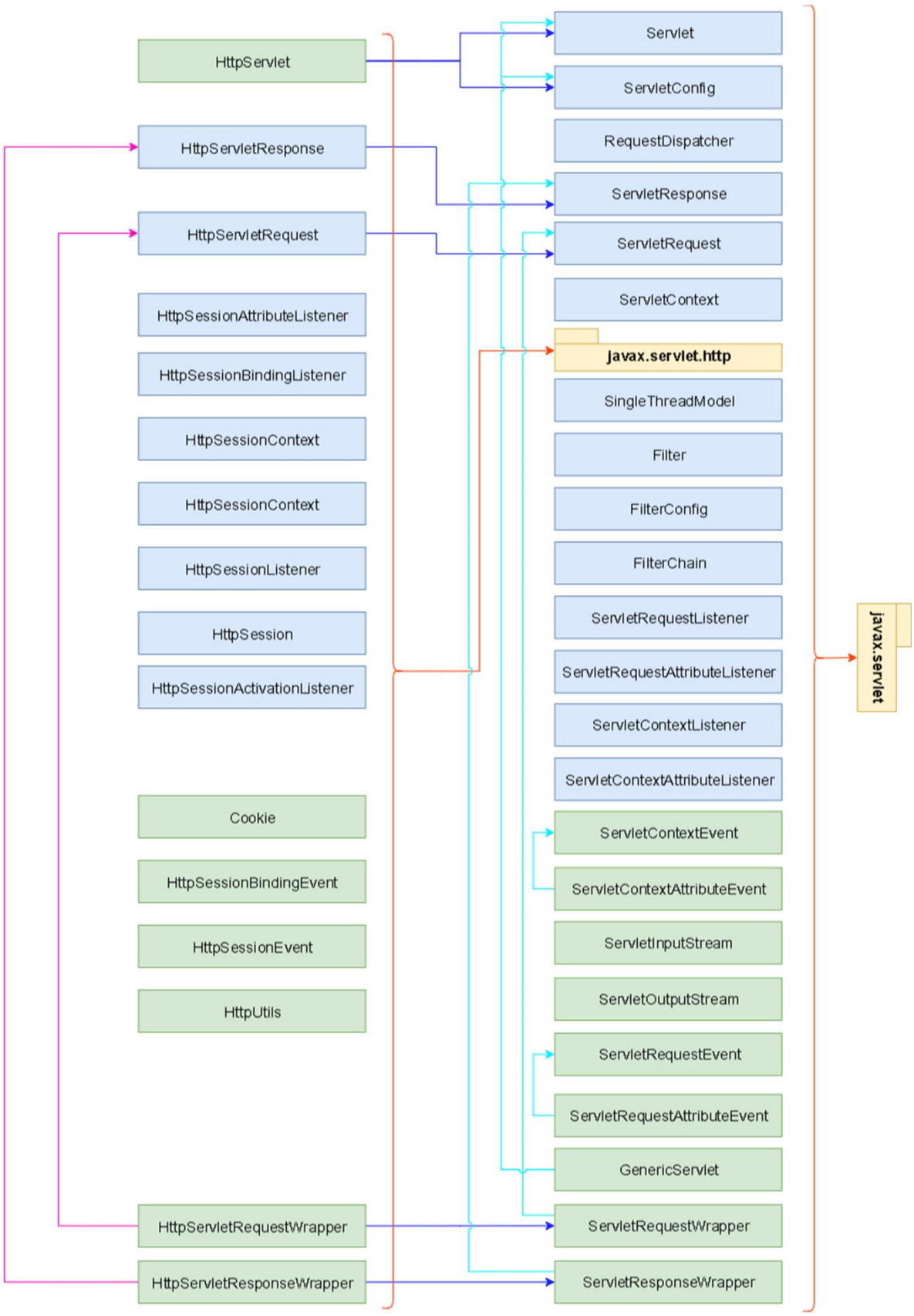
3. **Invoke init() method** - Once all the servlet classes are instantiated, the init() method is invoked for each instantiated servlet. This method initializes the servlet. The init() method is called only once during the life cycle of servlet.

4. **Invoke service() method** - Each time the web server receives a request for servlet, it spawns a new thread that calls service() method. If the servlet is GenericServlet then the request is served by the service() method itself, if the servlet is HttpServlet then service() method receives the request and dispatches it to the correct handler method based on the type of request. Unlike init() and destroy() that are called only once, the service() method can be called any number of times during servlet life cycle. As long as servlet is not destroyed, for each client request the service() method is invoked.

5. **Invoke destroy() method** - When servlet container shuts down(this usually happens when we stop the web server), it unloads all the servlets and calls destroy() method for each initialized servlets.



Servlet API Family



Servlet Vs GenericServlet Vs HttpServlet

	Servlet	GenericServlet	HttpServlet
What it is?	Top level **interface** in the hierarchy of Java servlets API which defines all the necessary methods to be implemented by the servlets.	An **abstract class** which provides methods to write protocol-independent servlets.	An **abstract class** which provides methods to write HTTP-specific servlets.
Package	javax.servlet	javax.servlet	javax.servlet.http
Hierarchy	Top level interface	Implements Servlet interface	Extends GenericServlet
Methods	init(), service(), destroy(), getServletConfig(), getServletInfo()	init(), service(), destroy(), getServletConfig(), getServletInfo(), log(), getInitParameter(), getInitParameterNames(), getServletContext(), getServletName()	doGet(), doPost(), doPut(), doDelete(), doHead(), doOptions(), doTrace(), getLastModified(), service()
Abstract Methods	All methods	Only service() method	Nothing
When to use?	When you want to develop your own Servlet container	Use to write protocol independent servlets	Use to write HTTP specific servlets.