# JAVA SERVLETS

Java Servlet is a java program which runs on web/application server and acts as a middle layer between the requests coming from client browser or any HTTP client applications and the database or any HTTP applications (applications on HTTP server).

**Servlet functionality -**

**Reads the below information -**

Client + Browser 🡪 sends data (explicit data from client which is in html forms) and (implicit data from web browser which contains information like cookies, media related information and compression schemes allowed by the browser) 🡪 To servlet.

**Generate results –**

Servelet need to process the data. In order to do so it talks with Database or RMI (Remote method invocation) or EJB (Enterprise Java beans) call or webservice call.

**RMI –** This is a java application (API) which helps in distributed application model (RMI is a way an object from one JVM can call a method of other object running on a different JVM).

**EJB –** Enterprise JavaBeans (EJB) is one of several [Java APIs](https://en.wikipedia.org/wiki/List_of_Java_APIs) for modular construction of enterprise software. EJB is a [server-side](https://en.wikipedia.org/wiki/Server-side) software component that encapsulates [business logic](https://en.wikipedia.org/wiki/Business_logic) of an application. An EJB web container provides a runtime environment for web related software components, including computer security, Java servlet lifecycle management, transaction processing, and other [web services](https://en.wikipedia.org/wiki/Web_services). The EJB specification is a subset of the [Java EE](https://en.wikipedia.org/wiki/Java_EE) specification.

**POJO** (Plain old Java Object) classes and Beans both are used to define java objects to increase their readability and reusability. POJOs don’t have other restrictions while beans are special POJOs with some restrictions.

**JDBC** – Java Database connectivity (interface which is used to connect java applications to database)

**CORBA** – Common Object Request Broker Architecture ( interface for communication between different applications developed in different languages on different systems)

**JNI** – Java native interface (communication between different applications developed in different languages on different systems).

The data you need may be in a relational database. Relational Database cannot talk directly using http or with web browsers, even it does there is security concerns as your data is exposed. So, we need a middle layer to get the data and transmit back to the client.

**Sending the results**

Again there are 2 types of data we need to send.

1. Explicit data – in the form of html, gzip (actual data requested by client).
2. Implicit data – required by the browser to know what format (html/text) the servlet returned, along with cookie information (setting cookies) , caching parameters etc.

* Java servlets are more efficient, easier to use, more powerful, more portable, safer,

and cheaper than traditional CGI and many alternative CGI-like technologies. (The Common Gateway Interface (CGI) is a standard for writing programs that can interact through a Web server with a client running a Web browser)

**Efficient** – Servlets will be started on java lightweight threads rather than heavy weight CPU processes unlike CGI. Also if a program is called N times, CGI creates N processes while servlet creates N threads and a single process. (less overhead). After the end of program, CGI terminates loosing all cache data but servlet maintains data.

**Convinient** : Lot of infrastructure (libraries) available in servlets unlike CGI where you need to write code for everything.

**Powerful**: Servlets can communicate directly with the webserver and also among themselves, which helps in resource sharing. But in case of CGI they need an additional interface or an API to do so.

**Portable** : Servlets can run on almost all the web servers.

**Inexpensive**: Servlets can run on any free/open source web servers like Tomcat. But CGI needs some initial investment to get a web server.

**Servlet Initialization using Initialization Parameters –**

We use web.xml file of the servlet to configure the init parameters and give deployers/administrators flexibility to config the servlet.

Some important elements in the web.xml file to use – (**servlet, servlet-mapping,init-param**)

Use **servlet** to **name** your servlet.

Use **servlet-mapping** to map (i.e., url ) to access your servlet.

Use **init-param** a sub element in servlet to assign names and values of initialization parameters.

From within your servlet’s init method, call getServletConfig to obtain a reference to the ServletConfig object.

Call the getInitParameter method of ServletConfig with the name of the init parameter. The return value is the value of the init parameter or null if no such init parameter is found in the web.xml file.

**FORM DATA HANDLING –**

One of the nice features of servlets is that all of this form parsing is handled automatically. You call **request.getParameter** to get the value of a form parameter. You can also call **request.getParameterValues** if the parameter appears more than once, or you can call **request.getParameterNames** if you want a complete list of all parameters in the current request. In the rare cases in which you need to read the **raw request data and parse it yourself**, call **getReader or getInputStream.**

**Reading individual request parameters**

Simply call the **getParameter** method of **HttpServletRequest**, supplying the case-sensitive parameter name as an argument

Ex : request.getParameter(“htmlFormInputName”) this returns the field value. If the field value is empty it returns empty string. If the parameter doesnot exist it returns null.

The values supplied to getParameter and getParameterValues are case sensitive.

**Reading the entire set of request parameters**

If the same parameter name might appear in the form data more than once, you should call getParameterValues (which returns an array of strings) instead of getParameter (which returns a single string corresponding to the first occurrence of the parameter). The return value of getParameterValues is null for nonexistent parameter names and is a one-element array when the parameter has only a single value.

Use **getParameterNames** to get the list of parameters, in the form of an Enumeration, each entry of which can be cast to a String and used in a getParameter or getParameterValues call.

If there are no parameters in the current request, getParameterNames returns **an empty Enumeration (not null).**

Don’t count on getParameterNames returning the names in any particular order.

An alternative to getParameterNames is getParameterMap. This method returns a Map: the parameter names (strings) are the table keys and the parameter values (string arrays as returned by getParameterNames) are the table values.

Reading Raw form data using **getReader or getInputStream** methods of HttpServlet request.

The first case in which you might read and parse the data yourself is when **the data comes from a custom client rather than by an HTML form.** The most common custom client is an applet.

The second situation in which you might read the data yourself is when the data is from an uploaded file. HTML supports a element (FORM <INPUT TYPE="FILE"...>) that lets the client upload a file to the server.

**setCharacterEncoding** is used to set the character type (which may not be always english). Also there may be multiple character sets involved.

So, you are left with two choices: read the parameter in one character set and convert it to another, or use an autodetect feature provided with some character sets. For the first option, you would read the parameter of interest, use getBytes to extract the raw bytes, then pass those bytes to the String constructor along with the name of the desired character set. Here is an example that converts a parameter to Japanese:

**String firstNameWrongEncoding = request.getParameter("firstName");**

**String firstName = new String(firstNameWrongEncoding.getBytes(), "Shift\_JIS");**

For the second option, you would use a character set that supports detection and

conversion from the default set. A full list of character sets supported in Java is available

at <http://java.sun.com/j2se/1.4.1/docs/guide/intl/encoding.doc.html>.

For example, to allow input in either English or Japanese, you might use the following.

**request.setCharacterEncoding("JISAutoDetect");**

**String firstName = request.getParameter("firstName");**

**Use form ACTION URLs that are relative, not absolute.**

**Handling missing and malformed data**

Design your servlets to gracefully handle parameters that are missing (null or empty string) or improperly formatted. Test your servlets with missing and malformed data as well as with data in the expected format.

**Filtering special characters out of the request parameters**

**Automatically filling in a data object with request parameter values**

**Dealing with incomplete form submissions**