MVC Pattern

Prof. Vipul Dabhi
Dept. of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad

MVC (Model, View, Controller)

- Main objective of MVC architecture is to separate the data (model) and the user interface (view)
  - Helps the developer : changing to the user interface can be made without affecting the underlying data handling logic
  - The data can be reorganized without changing the user interface
- Separation is achieved by introducing an controller component
  - Controller is an intermediate component
  - Controller defines as how the user interface should react to a user input
- Logical Layers in a Web Application
  - Model [Business Process Layer]
  - View[Presentation Layer]
  - Controller[Control Layer]

Model

- Model contains only the pure application data
  - Models data and behavior behind business process
  - Responsible for actually doing
    - Performing DB queries
    - Calculating the business process
    - Processing orders
  - Encapsulate of data and behavior which are independent of presentation

View (Presentation Layer)

- Display information according to client types
- Display result of business logic (Model)
- Not concerned with how the information was obtained, or from where (since that is the responsibility of Model)

Controller

- Serves logical connection between user's interaction and the business process (backside of application)
- Responsible for making decision among multiple presentation
- A request enters the application through the control layer, it will decide how the request should be handled and what information should be returned (response)
- Responsible for calling methods on the model that changes the state of the model

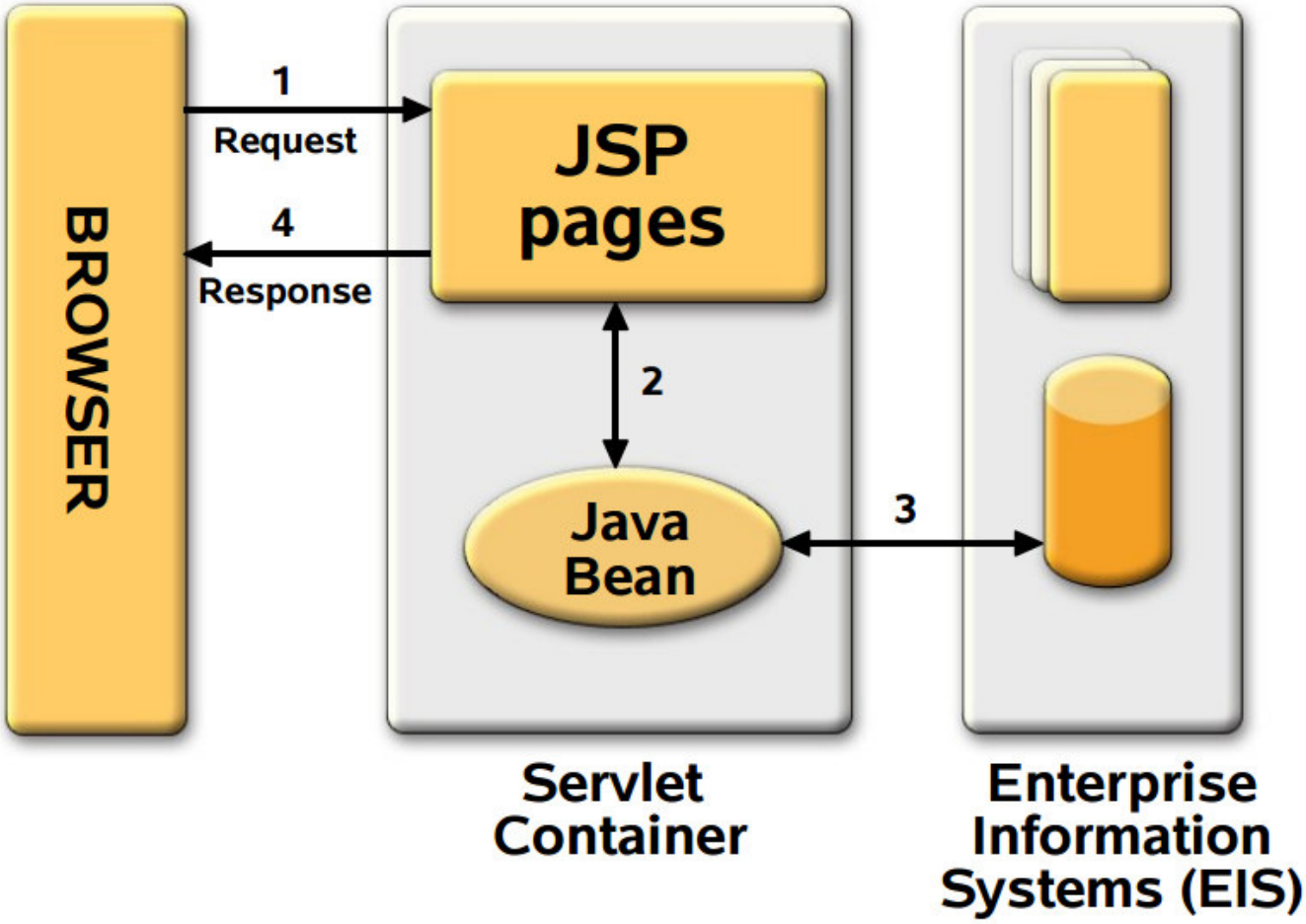
Relationship between components

- View and Controller
  - controller is responsible for creating or selecting view
- Model and View
  - View depends on Model
  - if a change is made to the model then there might be required to make parallel changes in the view
- Model and Controller
  - If a change is made to the model then there might be required to make parallel changes in the Controller

Evolution of MVC Architectures

1. No MVC
2. MVC Model 1 (Page-centric)
3. MVC Model 2 (Servlet-centric)
4. Web application frameworks
   - Struts
5. Standard-based Web application framework
   - JavaServer Faces (JSR-127)

Model 1 (Page Centric) Architecture



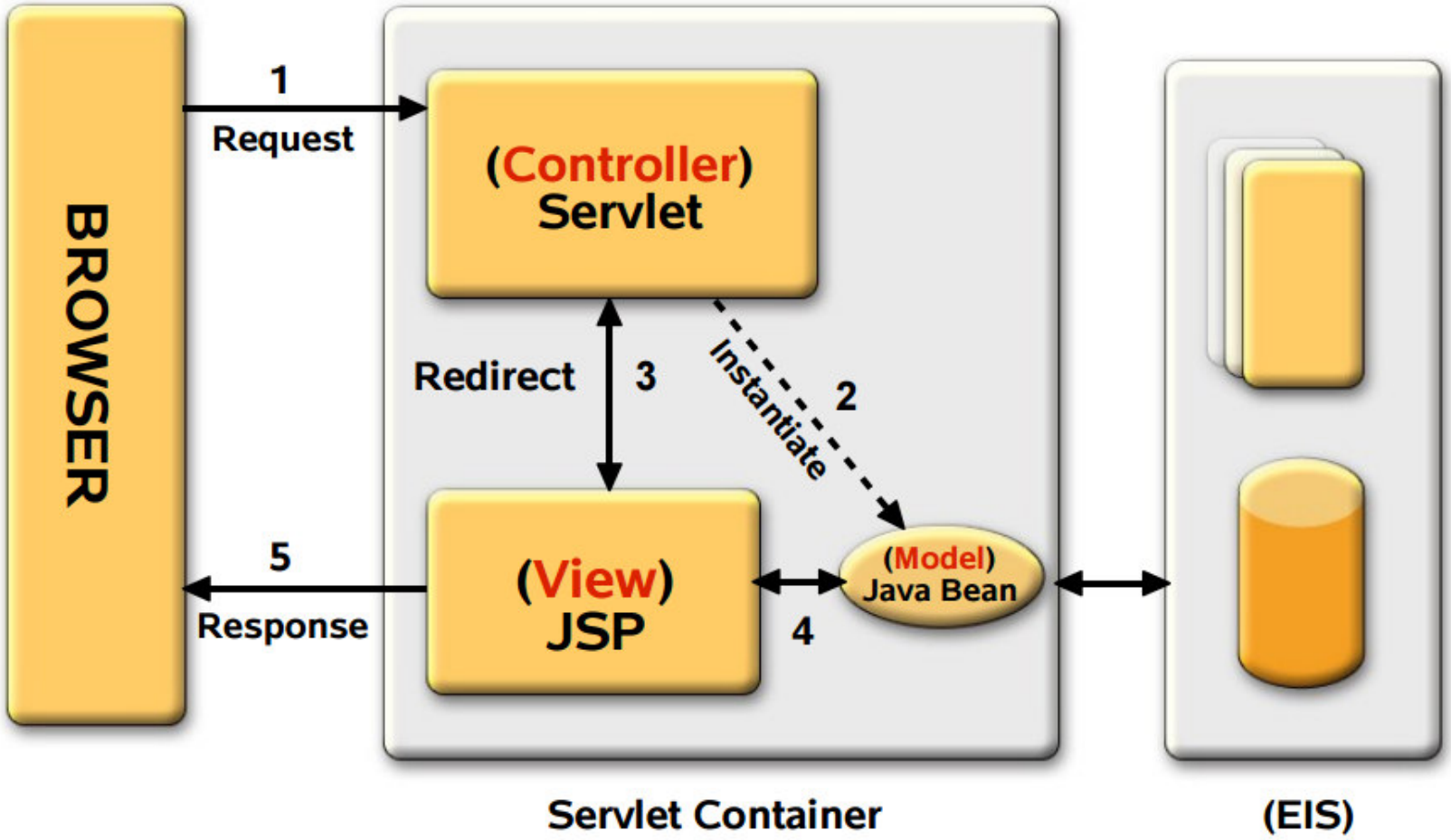
Model 1 (Page Centric) Architecture

- Composed of a series of interrelated JSP pages
  - JSP pages handle all aspects of the application: presentation, control, and business process
- Business process logic and control decisions are hard coded inside JSP pages
  - in the form of JavaBeans, scriptlets, expression
- Next page selection is determined by
  - A user clicking on a hyper link, e.g. <a href="find.jsp>
  - Through the action of submitting a form, e.g. <form action="search.jsp">

Limitations of MVC Model - 1

- Navigation Problem – Changing name of JSP file must change in many location
- Difficult to maintain an application – large java code being embedded in JSP page
- Performance is not high and difficult to extend
  - It is better for small applications but not for large application
- What if you want to present different JSP pages depending on the data you receive?
- Solution:
  - Use Servlet and JSP together (Model 2) Servlet handles initial request, partially process the data, set up beans, then forward the results to one of a number of different JSP pages

Model 2 (Servlet Centric) Architecture



Model 2 (Servlet Centric) Architecture

- JSP pages are used only for presentation
  - Control and application logic handled by a servlet (or set of servlets)
- Servlet serves as a gatekeeper
  - Provides common services, such as authentication, authorization, login, error handling, and etc
- Servlet serves as a central controller
  - Act as a state machine or an event dispatcher to decide upon the appropriate logic to handle the request
  - Performs redirecting

Difference between Model 1 and Model 2

- Model 1 and Model 2 simply refer to the absence or presence (respectively) of a controller servlet that dispatches requests from the client tier and selects views.

Web Application Framework

- Based on MVC Model 2 architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Web Application Frameworks

- Web Application Frameworks
  - Apache Struts
  - Java Server Faces (JSR-127)
    - A server side user interface component framework for Java technology-based web applications
  - Echo
  - Tapestry

Including another web resource

- When to include another web resource?
  - When it is useful to add static or dynamic contents already created by another web resource
    - Adding a banner content or copyright information in the response returned from a web component.
  - When you want to have one web component do preliminary processing of a request and have another component generate the response
  - You might want to partially process a request and then transfer to another component depending on the nature of the request.

Types of Included Web Resource

- Static Resource
  - It is like “programmatic” way of adding the static contents in the response of “including” servlet.
- Dynamic web component (Servlet or JSP)
  - Send the request to “included” web component
  - Execute the “included” web component
  - Include the result of execution from the “included” web component in the response of the “including” servlet.

Can and Can’t: Web resource

- Included web resource has access to the request object, but it is limited to:
  - It can write to the body of response and commit a response
  - It cannot set headers or call any method that affects the headers of the response.

RequestDispatcher

- Request attributes make sense when you want some other component of application to take over all or part of the request.
  - For ex., the controller communicates with the model, and gets back data that the view needs in order to build the response.
- There is no reason to put the data in a context or session attribute, since it applies only to this request, so we put it in the request scope.

RequestDispatcher

- How another component of application take over the request?
  - Using RequestDispatcher.

Code in doGet() or doPost()

```
BeerExpert be = new BeerExpert();
ArrayList result = be.getBrands (c);

Request.setAttribute("styles", result);

RequestDispatcher view =
    request.getRequestDispatcher("result.jsp");
view.forward (request, response);
```

RequestDispatcher

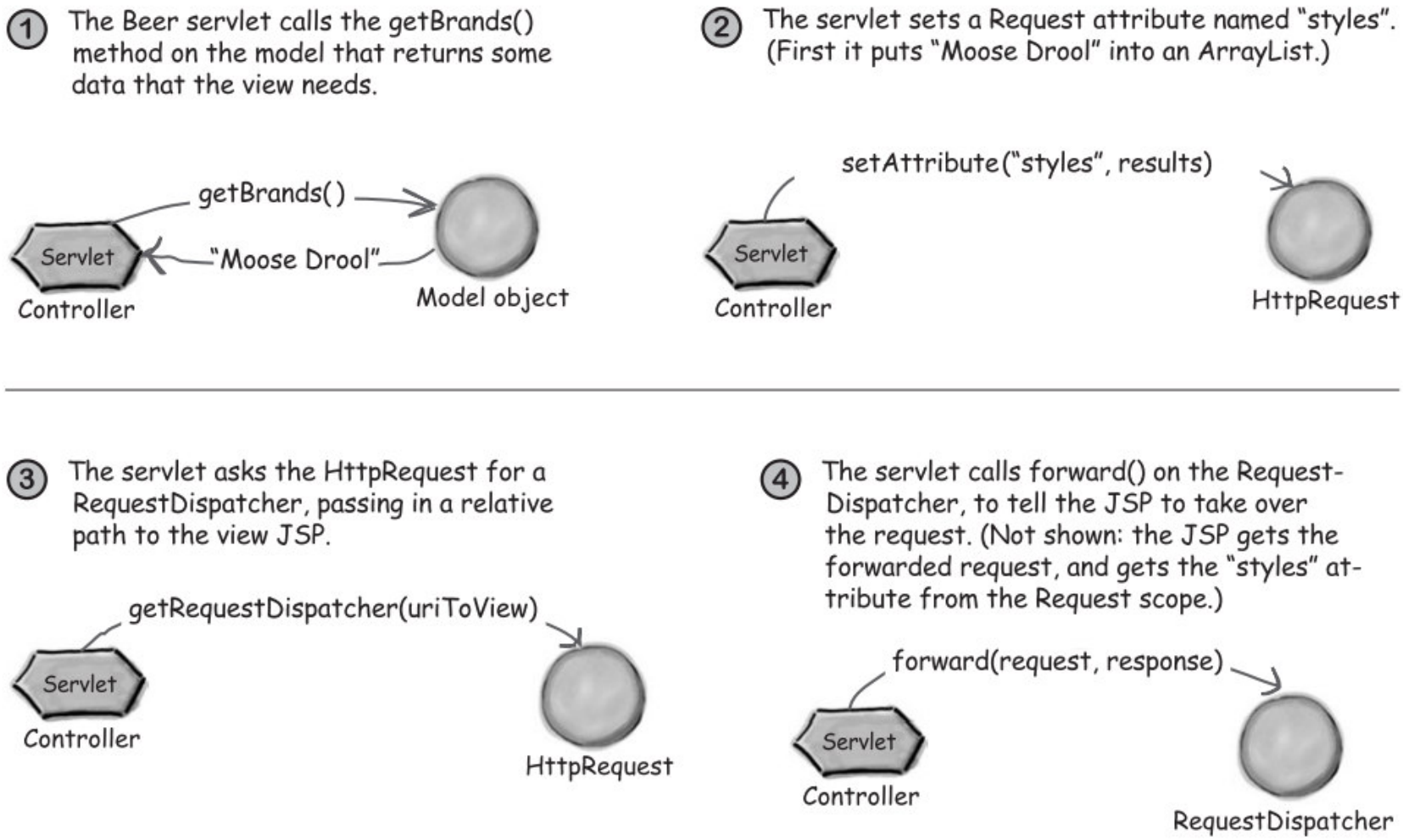


Image Source: Head First Servlets and JSP 2nd Edition - by Bryan Basham, Kathy Sierra & Bert Bates, O'Reilly

Obtaining RequestDispatcher

- Two ways to get RequestDispatcher
  - From the request
  - From the context.
- Obtaining RequestDispatcher from ServletRequest
  - RequestDispatcher view = request.getRequestDispatcher(“result.jsp);
    - Relative path for result.jsp: The container looks for “result.jsp” in same logical location the “request” is in.
- Obtaining RequestDispatcher from ServletContext
  - RequestDispatcher view = getServletContext().getRequestDispatcher(“/result.jsp);
    - Absolute path for result.jsp

RequestDispatcher

- RequestDispatchers have two methods:
  - forward()
  - include()
  - Both these methods take request and response objects as arguments
- In servlets, we use forward() method most of times. Include method is being used by JSPs in <jsp:include> standard action.
- Calling forward() on RequestDispatcher
  - view.forward (request, response)

RequestDispatcher

- You can't forward the request if you have already committed a response.
  - If you try to forward the request after the response is sent to the client, you will get an error.
  - **public void doGet (HttpServletRequest req, HttpServletResponse res)**
    {

```
res.setContentType("application/jar");
OutputStream  os  =  res.getOutputStream();
os.flush();
RequestDispatcher view = req.getRequestDispatcher("result.jsp");
view.forward(req, res);
os.close();
```

}

sendRedirect() of HttpServletResponse

- The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- It accepts relative as well as absolute URL.
- It works at client side because it uses the url bar of the browser to make another request.

Difference between Response.sendRedirect() and RequestDispatcher.forward()

- RequestDispatcher.forward()
  - method pass the control of the request to another servlet or jsp without telling anything about the request dispatch to the client browser.
  - Therefore client browser don't know whether the returned resource is from an another servlet/jsp or not.
- Response.sendRedirect()
  - method stop further processing of the request and send http status code "301" and URL of the location to be redirected to the client browser in the response header.
  - Server does not have control of this request after sending the redirect related HTTP header to the client browser.
  - Client browser sees http status 301 and then it knows it should send a new request to the url in "Location" http header which is set by server.
  - Client browser sends a new request to the new URL and it will be processed by the server as another normal request.

SendRedirect() Vs Forward()

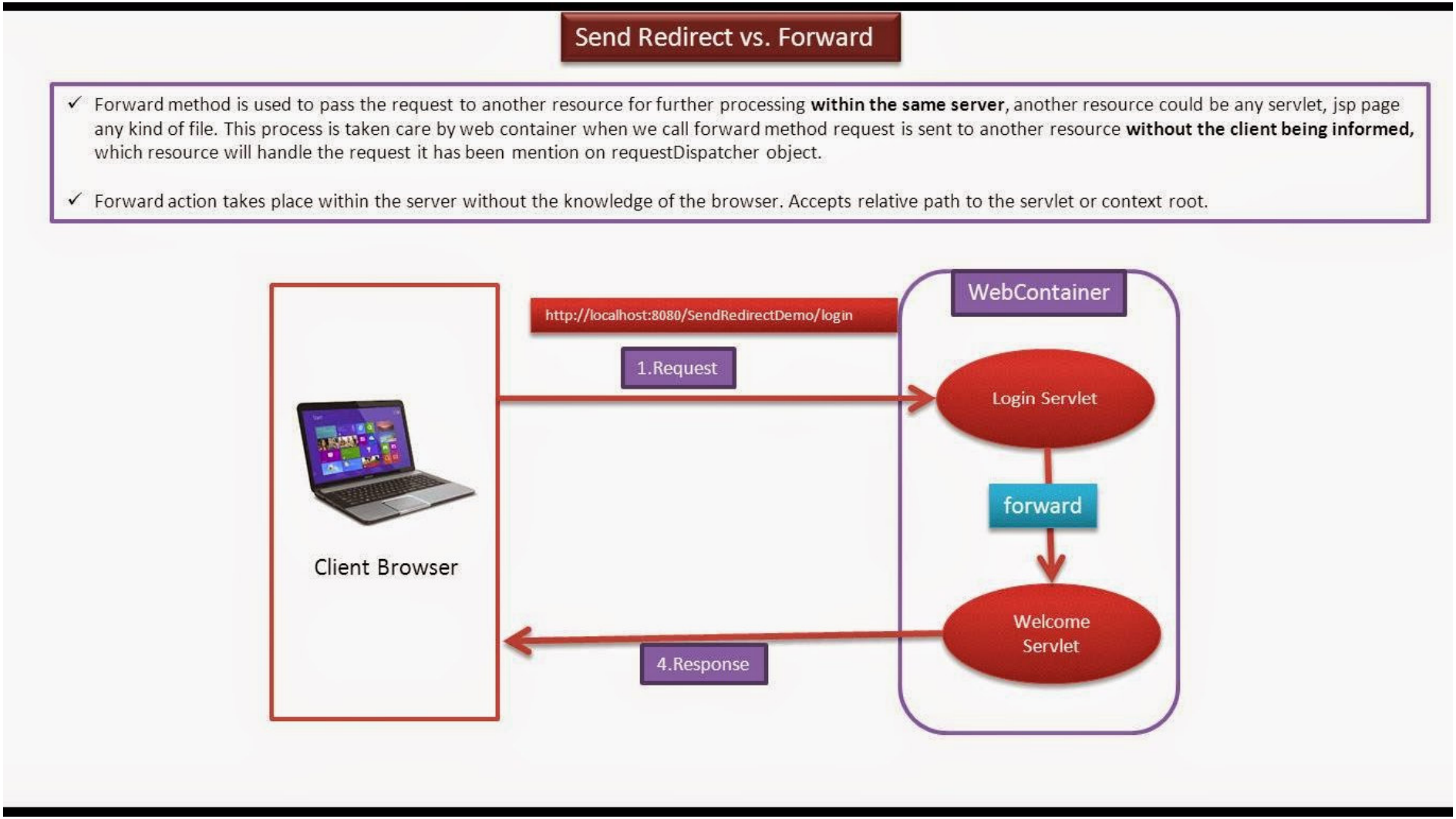


Image source: http://ramj2ee.blogspot.in/2014/04/servlets-send-redirect-vs-forward.html

SendRedirect() Vs Forward()

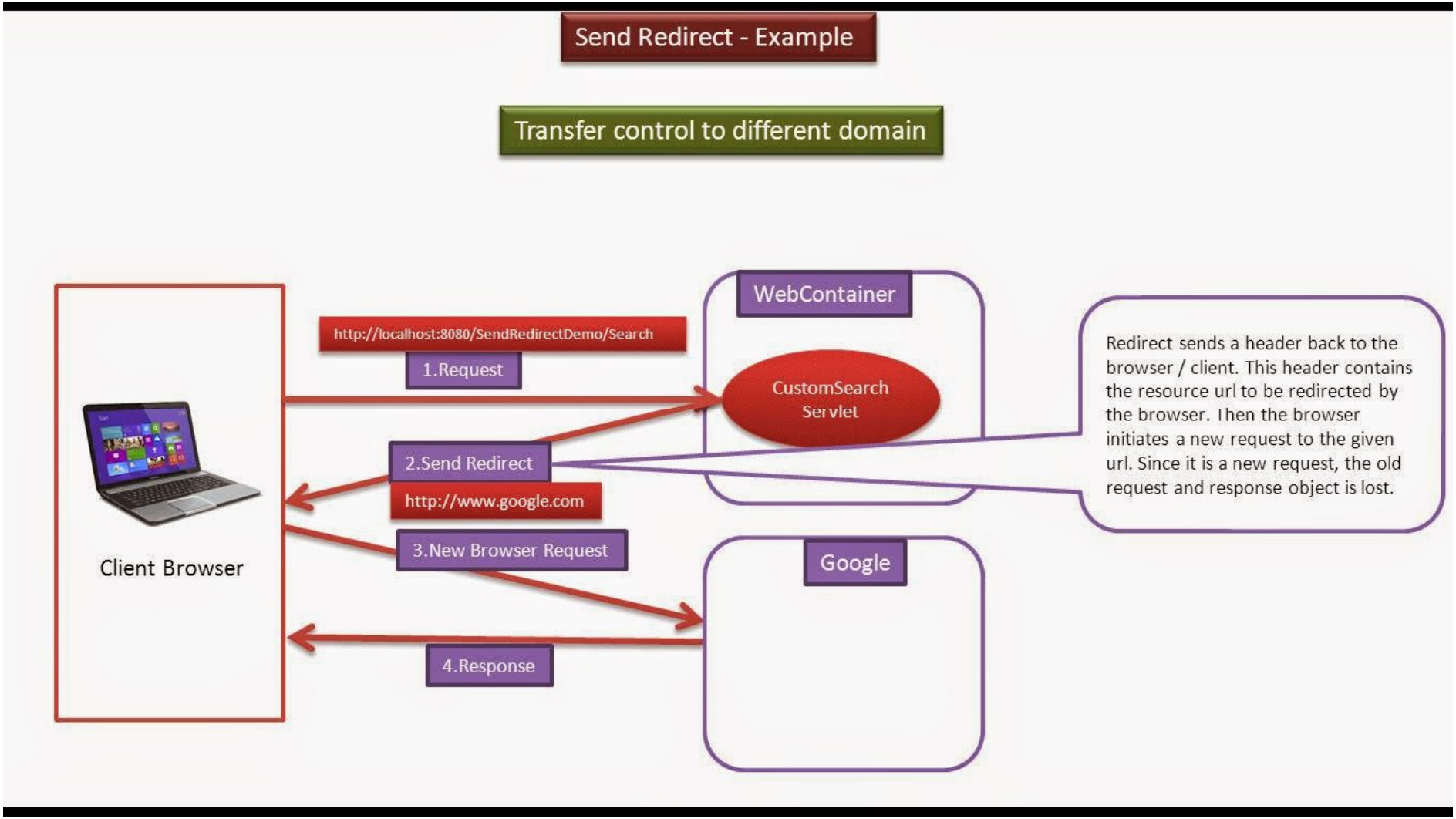


Image Source: http://ramj2ee.blogspot.in/2014/05/ servlets-send-redirect-demotransfer_5.html

Difference between Response.sendRedirect() and RequestDispatcher.forward()

RequestDispatcher: Forward()	HttpServletResponse: SendRedirect()
• When we use forward method request is transfer to other resource within the same server for further processing.	• In case of sendRedirect request is transfer to another resource to different domain or different server for further processing.
• In case of forward Web container handle all process internally and client or browser is not involved.	• When you use SendRedirect container transfers the request to client or browser so url given inside sendRedirect method is visible as a new request to the client.
• When forward is callon **requestdispatcher** object we pass request and response object so our old request object is present on new resource which is going to process our request	• In case of SendRedirect call old request and response object is lost because it's treated as new request by the browser.
• Visually we are not able to see the forwarded address, its is transparent	• In address bar we are able to see the new redirected address it's not transparent.
• When we redirect using forward and we want to use same data in new resource we can use request.setAttribute () as we have request object available.	• But in sendRedirect if we want to use we have to store the data in session or pass along with the URL.