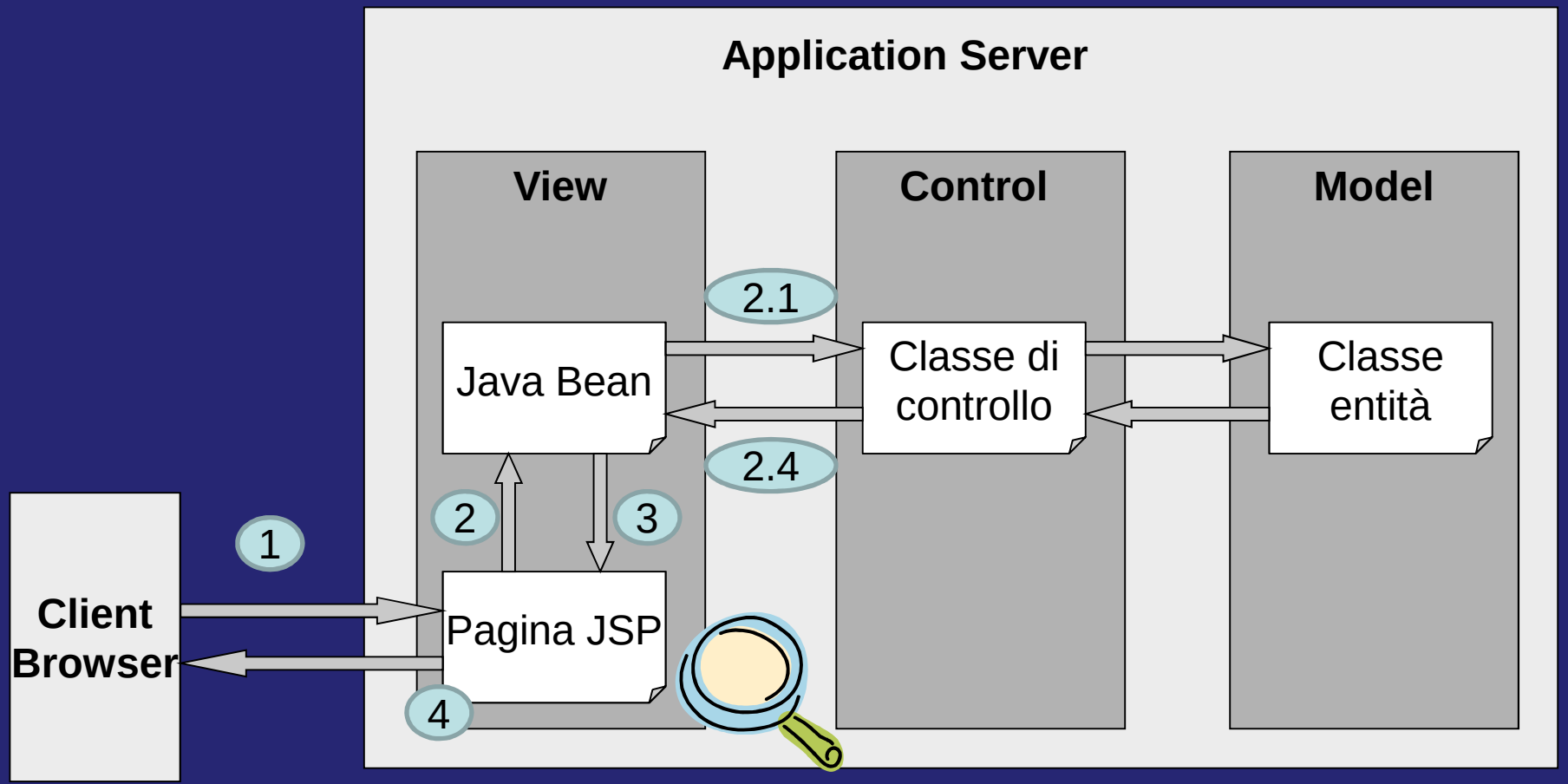
Module 15c

Java Servlets

Contents

Memo: WA-MVC Arch.



Java Servlet

A Java servlet is a Java programming language class used to extend the capabilities of a server.

Although servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers, so they can be thought of as Java Applets that run on servers instead of in web browsers.

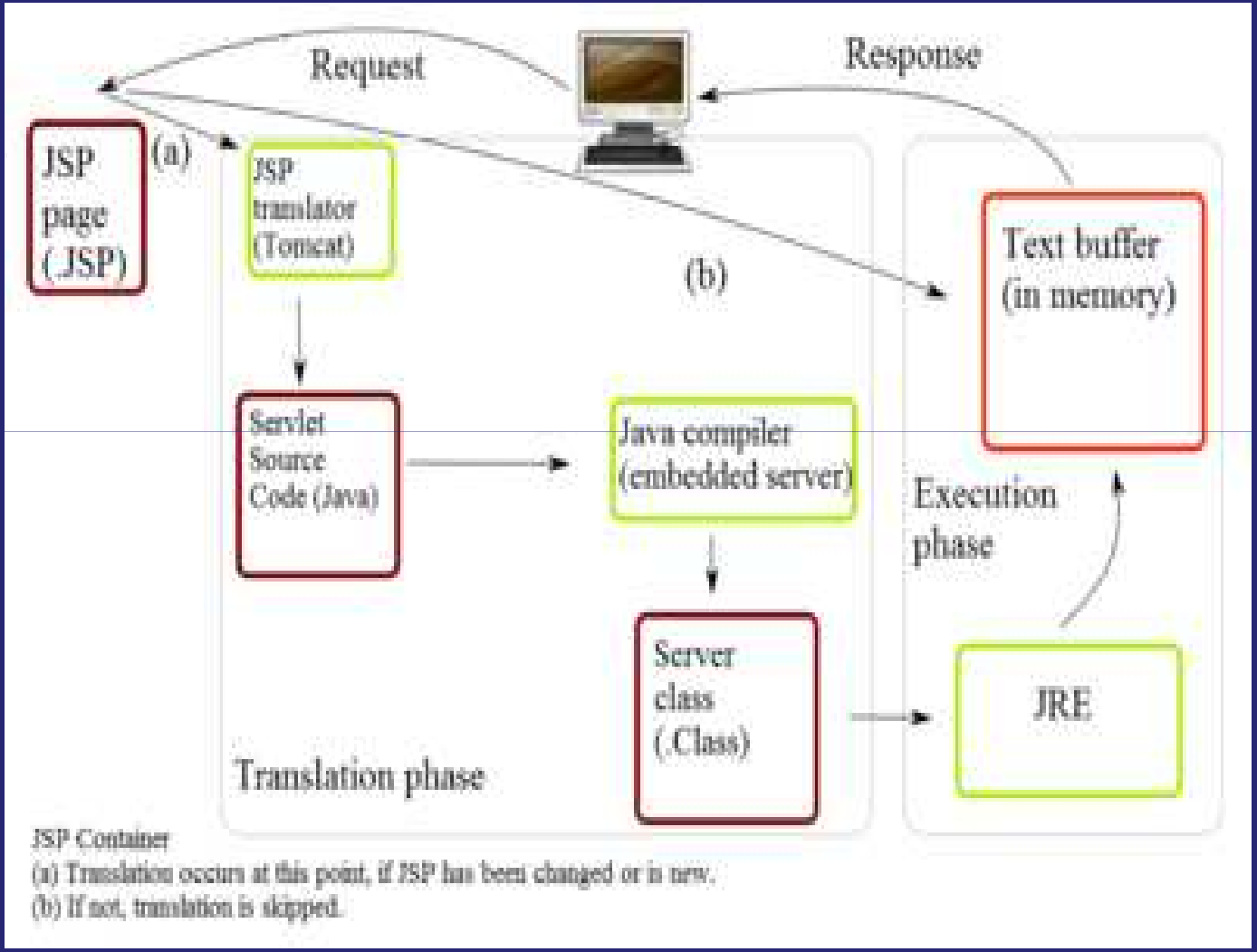
These kinds of servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.

DICTIONARY: Servlet

Servlet: Technically speaking, a "servlet" is a Java class in Java EE that conforms to the *Java Servlet API*, a standard for implementing Java classes which respond to requests.

Servlets could in principle *communicate over any client–server protocol*, but they are most often used with the HTTP protocol. Thus "servlet" is often used as shorthand for "*HTTP servlet*".

Life of a Java Server Page



Why are most often servlet used to?

Servlets are mostly used:

- To process or store data.
  - E.g., data that was submitted from an HTML form.
- To provide dynamic content.
  - E.g., the results of a database query.
- To manage state information, which does not exist in the stateless HTTP protocol.
  - E.g., filling the articles into the shopping cart of the appropriate customer.

What is needed to deploy a Servlet?

To deploy and run a servlet, a web container must be used.

A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets.

The Servlet API, contained in the Java package hierarchy `javax.servlet`, defines the expected interactions of the web container and a servlet.

What are the Responsibilities of a Web Container

The web container is responsible for:

- Managing the lifecycle of servlets;
- Mapping a URL to a particular servlet, and
- Ensuring that the URL requester has the correct access rights.

Servlet Generation

Servlets can be generated automatically from Java Server Pages by the JavaServer Pages compiler.

A difference between servlets and JSP is that servlets typically embed HTML inside Java code, while JSPs embed Java code in HTML.

Additionally, a servlet is longer in size and much more complex to write corectly than the equivalent JSP.

The direct usage of servlets to generate HTML has become rare, less few relevant exceptions.

Comparison with other WA models

The advantages of using servlets are their fast performance and ease of use combined with more power over traditional solutions, like:

- Common Gateway Interfaces
- Server's API[D], such as ISAPI[D] or NSAPI[D].

DICTIONARY: API

API, an abbreviation of ***A****pplication* ***Pr****ogram* ***I****nterface*, is a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer then puts the blocks together.

Most operating environments, such as MS-Windows, provide an API so that programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. This makes it easier for users to learn new programs.

DICTIONARY: ISAPI, NSAPI

- **ISAPI**: Short for ***I**nternet **S**erver **API**,* an API for Microsoft's IIS (Internet Information Server) Web server. ISAPI enables programmers to develop Web-based applications that run much faster than conventional CGI programs because they're more tightly integrated with the Web server. In addition to IIS, several Web servers from companies other than Microsoft support ISAPI.
- **NSAPI**: Short for ***N**etscape **S**erver **API**,* an API for Netscape's Web servers. NSAPI enables programmers to create Web-based applications that are more sophisticated and run much faster than applications based on CGI scripts.

DICTIONARY: CGI

CGI: Short for Common Gateway Interface.
Traditional CGIs written in Java have a number of disadvantages when it comes to performance:

- When an HTTP request is made, a *new process* is created for each call of the CGI script.

This overhead of process creation can be very system-intensive, especially when the script does relatively fast operations. Thus, process creation will take more time for CGI script execution.

What is a CGI

CGI is a specification for transferring information between a World Wide Web server and a CGI program.

A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic.

CGI programs have been a common way for Web servers to interact dynamically with users. Many HTML pages that contain forms, for example, use a CGI program to process the form's data once it's submitted. Another increasingly common way to provide dynamic feedback for Web users is to include scripts or programs that run on the user's machine rather than the Web server. These programs can be Java applets, Java scripts, or ActiveX controls. These technologies are known collectively as *client-side* solutions, while the use of CGI is a *server-side* solution because the processing occurs on the Web server.

One problem with CGI is that each time a CGI script is executed, a new process is started..

Servlets

In contrast with traditional CGI scripts written in Java:

For servlets, each request is handled by a separate Java thread *within the* web server process, omitting separate process forking by the HTTP daemon.

Servlet vs. CGI

Simultaneous CGI request causes the CGI script to be *copied* and *loaded* into memory as many times as there are requests.

With servlets, there is *only one copy* that persists across requests and is shared between threads. Only a single instance answers all requests concurrently. This reduces memory usage and makes the management of persistent data easy.

However, servlets are substantially *much more complex to program* than CGI.

Lifecycle of a Servlet

Three methods are central to the life cycle of a servlet. These are **`init()`**, **`service()`**, and **`destroy()`**. They are implemented by every servlet and are invoked at specific times by the server.
During initialization stage of the servlet life cycle, the web container initializes the servlet instance by calling the **`init()`**method, passing an object implementing the **`javax.servlet.ServletConfig`** interface. This configuration object allows the servlet to access *name-value* initialization parameters from the web application.

Lifecycle of a Servlet

After initialization, the servlet instance can service client requests.

Each request is serviced in its own *separate thread*.

The web container calls the **`service()`** method of the servlet for *every request*.

The **`service()`** method determines the kind of request being made and dispatches it to an appropriate method to handle the request.

The developer of the servlet must provide an implementation for these methods.

Lifecycle of a Servlet

If a request is made for a method that is not implemented by the servlet, the method of the parent class is called, typically resulting in an error being returned to the requester.
Finally, the web container calls the **`destroy()`** method that takes the servlet out of service. The **`destroy()`** method, like **`init()`**, is called only once in the lifecycle of a servlet.

Typical User Scenario

1. Let a user request to visit a URL.
    1. The browser then generates an HTTP request for this URL.
    2. This request is then sent to the appropriate server.
2. The HTTP request is received by the web server and forwarded to the servlet container.
    1. The container maps this request to a particular servlet.
    2. The servlet is dynamically retrieved and loaded into the address space of the container.
3. The container invokes the `init()` method of the servlet.
    1. This method is invoked only when the servlet is first loaded into memory.
    2. It is possible to pass initialization parameters to the servlet so that it may configure itself.

http://en.wikipedia.org/wiki/Servlet

Typical User Scenario

4. The container invokes the **`service()`** method of the servlet.
    1. This method is called to process the HTTP request.
    2. The servlet may read data that has been provided in the HTTP request.
    3. The servlet may also formulate an HTTP response for the client.
5. The servlet remains in the container's address space and is available to process any other HTTP requests received from clients.
    1. The **`service()`** method is called for each HTTP request.

http://en.wikipedia.org/wiki/Servlet

Typical User Scenario

4. The container calls the servlet's **`destroy()`** method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
5. The memory allocated for the servlet and its objects can then be garbage collected.

http://en.wikipedia.org/wiki/Servlet

Example

JavaServletExample