J2EE Web Component Development

Training Course

Chau Keng Fong
Adegboyega Ojo

e-Macao Report 24

Version 1.0, November 2005

Table of Contents

1. Overview

The Java 2 Enterprise Edition (J2EE) platform is a standard technology for building Internet applications and particularly Enterprise Applications. J2EE is a suite of Application Programming Interfaces (APIs), a distributed computing architecture, and the method of packaging distributable components for deployment [4]. Enterprise Applications are essential for safe storage, retrieval and manipulations of business data. They are characterised by multiple user interfaces, both web- and desktop-oriented, communication between remote systems and data coordination on different machines.

Enterprise applications are multi-tier applications consisting of client, web, business and enterprise tiers. J2EE provides the necessary APIs and services to develop these tiers, with Web and Business Components implementing web and business tiers respectively.

This course introduces the J2EE Technologies, focusing mainly on the Web Component Technologies. It starts by introducing the J2EE Platform including its origin, architecture and components. Next, the course teaches the core technologies for developing web components: how to develop controllers as Servlets, how to view contents using Java Server Pages and how to use Filters for processing requests and responses. Later, support for horizontal technologies like exception handling, database connectivity, security and internationalization are briefly discussed.

The rest of this document explains the objectives, prerequisites and methodology for teaching the course in Sections 2, 3 and 4 respectively. The content of the course is introduced in some detail in Section 5. The assessment and organization of the course are explained in Sections 6 and 7. Following references, Appendix A includes the complete set of slides and Appendix B contains two sets of assessment questions with answers.

2. Objectives

The course has three main objectives:

1) To introduce the J2EE Technology to students.
2) To equip students with essential skills for developing enterprise applications using three J2EE Web Components technologies:
    a) Servlets
    b) JavaServer Pages
    c) Filters
3) To teach techniques for developing multi-lingual, secure web applications.

3. Prerequisites

The course requires that students have a working knowledge of the Java Programming Language. Basic understanding of the TCP/IP network protocol is also essential. In addition, the knowledge of XML and HTML are assumed.

4. Methodology

The course has been designed based on the following didactic principles:

- *Depth versus Breadth* - As foundation, attempt is made to cover the various aspects of web services without loss of depth.
- *Academic Orientation* – A body of concepts is defined rigorously and incrementally to establish a proper foundation for understanding and use of technology.
- *From Definitions to Demonstrations* – All major concepts introduced during the course are illustrated with small-size examples which are also demonstrated on the computer, whenever possible.
- *From Demonstrations to Assignments* – On the basis of demonstrations, students are asked to perform different tasks with increasing level of difficulty and independence.

5. Content

The course consists of 490 slides organized into four major sections: J2EE Introduction, Vertical Concepts, Horizontal Concepts and Case Study. These sections are described below.

5.1. J2EE Introduction

This section comprises the slides 1 through 45. It presents an overview of the Java 2 Enterprise Edition (J2EE). The presentation covers the origin, architecture and design goal of the J2EE platform. The concepts of components, containers and application servers are introduced. Later in the section, standard services and platform roles defined by the J2EE specification are introduced. Finally, setting up the Apache Tomcat Web Server is explained.

5.2. Vertical Concepts

This section comprises the slides 46 through 372. It teaches three major J2EE Web Components technologies: (i) Servlet, (ii) JavaServer Pages (JSP) and (iii) Filter. The three different types of Web Components are presented as follows:

1) *Servlet* – The basic concepts of the architecture and lifecycle of Servlets are introduced. The HTTP protocol is explained before presenting how HTTPServlets are developed and deployed. Communication between Servlet, Client tier and Container are also explained.

2) *JavaServer Pages (JSP)* – The architecture and life cycle of JSPs are introduced. The use of scripting elements for writing JSP is explained. The usage of implicit objects and standard JSP actions like forward, include and useBeans are presented. The J2.0 Expression Language is introduced. Finally, standard and custom tags are discussed.

3) *Filter* - Filters and Filter Chains are presented and their use of explained. Also, request and response wrappers are discussed.

The lifecycles of all components are presented and compared.

5.3. Horizontal Concepts

This section consists of the slides 373 through 483. It presents the supporting technologies for J2EE applications: exception handling, database connectivity, security and internationalization. Each of these topics is explained below:

1) *Exception* – Exceptions in J2EE can be handled in different manners. This section first explains how to develop JSPs and Servlets to handle exceptions thrown by a web page. Subsequently, methods to intercept and handle exceptions thrown by Containers are discussed. The usage of error objects for retrieving error information is also explained. An illustration involving Java Mail to send out error message is presented. Finally, the concept of information logging is introduced.

2) *Database Connectivity* – The concept of data source to establish database connection is introduced. Next, the section introduces connection pools. Finally, configurations for data sources and connection pools are demonstrated.

3) *Security* - The technique for role-based and programmatic authentication in J2EE applications is presented. The role-based security in Apache Tomcat server is demonstrated. Configuration for providing secured communication through HTTPS protocol in Apache Tomcat is introduced as well. In addition, the section describes the programmatic security technique supported in the J2EE environment.

4) *Internationalization* – The concept of internationalisation is introduced, followed by a discussion of difficulties in developing multi-lingual web applications. The development of a multi-lingual website through the use of the Resource Bundle file is demonstrated. Tools for generating the Resource Bundle are presented as well.

5.4. Case Study

This section comprises the slides from 484 through 499. It presents a complete case study involving the development and deployment of a multi-lingual, secure website using the J2EE Web Components Technology.

6. Assessment

The course finishes with an assessment. This comprises 15 multiple-choice questions which cover all major sections and concepts taught.

Two sets of 15 assessment questions and answers are given in Appendices B.1 and B.2. The two sets are different permutation of the same collection of questions. The assessment complements the tasks provided in the various sections.

7. Organization

The course consists of lectures and demonstrations:

- *lectures* – The lectures mainly present the concepts and the use of the J2EE Web Components Technologies.
- *demonstrations* - Demonstrations illustrate the concepts introduced during the lectures with running code and examples. Some examples only provide skeleton codes and require students to complete them by applying the technologies discussed in the lectures.

The full course has been taught for 7 days with 6 hours of lectures per day. A shorter version of the course has also been taught over four days.

References

1. e-Macao Project, The State of Electronic Government in Macao, Volume 2: Agencies, 2005
2. Hans Bergsten, JavaServer Pages, 3rd edition, O'Reilly, 2003
3. Jayson Falkner and Kevin Jones, Servlets and JavaServer Pages: the J2EE Technology Web Tier, Addison-Wesley, 2003
4. James L. Weaver, Kevin Mukhar and Jim Crume, Beginning J2EE 1.4 – From Novice to Professional, Apress, 2004

Appendix

A. Slides

A.1. Introduction

J2EE Web Component Development

Milton, Chau Keng Fong
INESC-Macau

The Course

1) objectives - what do we intend to achieve?
2) outline - what content will be taught?
3) resources - what teaching resources will be available?
4) organization - duration, major activities, daily schedule

e-Macao-16-6-3

Course Objectives

1) explain the concept of J2EE
   a) origin
   b) architecture

2) present the core J2EE Web Components technologies
   a) Servlets
   b) JavaServer Pages
   c) Filters

3) present the techniques to develop a multi-lingual, secured web site

e-Macao-16-6-4

Course Outline

1) J2EE introduction

2) vertical concepts
   a) Servlets
   b) JavaServer Pages
   c) Filters

3) horizontal concepts
   a) exceptions
   b) database connectivity
   c) security
   d) internationalization

4) case study

e-Macao-16-6-5

Outline: J2EE Introduction

An overview of J2EE:

1) origin of J2EE
2) architecture of J2EE

e-Macao-16-6-6

Outline: Vertical Concepts

The main concepts about different types of web components:

1) introduction to Servlets, JavaServer Pages (JSP) and Filters
2) life cycle of different web components
3) development of different types of web components
4) how to deploy a web application
5) criteria for the usage of different web components

e-Macao-16-6-7

Outline: Horizontal Concepts

Supporting technologies to develop web applications:

1) different techniques to handle exceptions and produce logging
2) various strategies to build secure web sites
3) different ways to connect to databases
4) procedures to develop a multi-lingual web site

e-Macao-16-6-8

Outline: Case Study

Build a web site utilizing the J2EE Web Component technologies:

1) enforce the MVC pattern with Filters
2) multi-lingual support with resource bundles
3) utilizing declarative security
4) adopt standard tag libraries in building JSPs

e-Macao-16-6-9

Course Resources

1) Books
   a) JavaServer Pages, Hans Bergsten, 3rd edition, O'Reilly, 2003
   b) Servlets and JavaServer Pages: the J2EE Technology Web Tier, Jayson Falkner, Kevin Jones, Addison-Wesley, 2003
2) Articles
   Links available from the website http://www.emacao.gov.mo.
3) Tools
   a) JDK 1.5.0_01
   b) Eclipse IDE 3.0.1
   c) Jakarta Tomcat 5.5.7

e-Macao-16-6-10

Course Logistics

1) duration - 42 hours
2) activities – lectures and development
3) timing
   a) Monday 09:00–13:00 14:30–17:45
   b) Tuesday 09:00–13:00 14:30–17:45
   c) Wednesday 09:00–13:00
   d) Thursday 09:00–13:00 14:30–17:45
   e) Friday 09:00–13:00
4) sessions - 7 morning, 4 afternoon
5) style - interactive and tutorial

e-Macao-16-6-11

Course Prerequisites

1) basic Java
2) basic understanding of TCP/IP networking concepts
3) basic understanding of XML
4) basic understanding of HTML

A.1.1. J2EE Introduction

J2EE Introduction

e-Macao-16-2-13

Course Outline

1) J2EE introduction

2) vertical concepts
   a) Servlets
   b) JavaServer Pages
   c) Filters

3) horizontal concepts
   a) exceptions
   b) security
   c) internationalization
   d) database connectivity

4) case study

e-Macao-16-2-14

J2EE Introduction Outline

1) Origin of J2EE
2) Architecture of J2EE
3) Summary

e-Macao-16-6-15

Background

The Java platform was first introduced in 1995 to address the programming needs for networks and cross-platform programming.

In order to address different needs, Sun Microsystems soon split the Java Technologies into three editions:

1) Java 2 Platform Micro Edition (J2ME)
2) Java 2 Platform Standard Edition (J2SE)
3) Java 2 Platform Enterprise Edition (J2EE)

e-Macao-16-6-16

Needs

In recent years, the needs for distributed computing in an enterprise made n-tier applications a popular program model.

The needs facing the developers:

a) simplifying the complexity of building n-tier applications
b) easily achieving :
   - availability
   - reliability
   - performance
   - scalability
   - reusability
   - interoperability
c) using a standardized API between components and application servers

e-Macao-16-6-17

J2EE Origin

Sun Microsystems, together with partners such as IBM, designed J2EE to define a multi-tier architecture for developing enterprise information systems (EIS) to answer the needs from the industry.

Goals:

a) reduce the cost and complexity of development
b) allow J2EE applications to be rapidly deployed and easily enhanced

e-Macao-16-6-18

J2EE Major Elements 1

J2EE consists of the following elements to pursue its design goals:

1) J2EE Platform – a standard platform for hosting J2EE applications.

2) J2EE Compatibility Test Suite (CTS) – all J2EE application servers have to pass the CTS test to carry the Java Compatible, Enterprise Edition logo.

e-Macao-16-6-19

J2EE Major Elements 2

3) J2EE Reference Implementation – a reference implementation and operational definition of the J2EE platform.

   A binary version can be downloaded as J2EE Software Development Kits (SDK).

4) J2EE Blue Prints – the standard programming model for developing multi-tier, thin client applications.

A.1.2. Architecture of J2EE

e-Macao-16-6-20

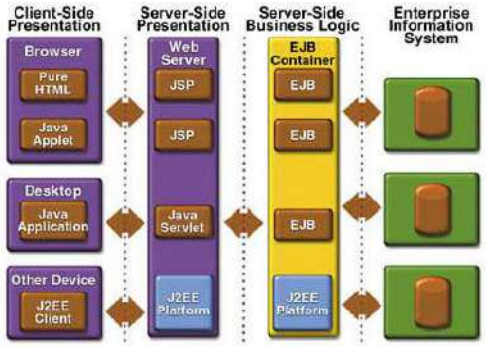
J2EE Introduction Outline

1) Origin of J2EE
2) Architecture of J2EE
3) Summary

e-Macao-16-6-21

J2EE Platform

J2EE platform uses a multi-tiered distributed application model.



courtesy of Sun Microsystems

e-Macao-16-6-22

J2EE Architecture

J2EE architecture is a component architecture.

A J2EE component is a self-contained functional software unit assembled into a J2EE application.

J2EE components are deployed to run on a J2EE server, which executes and manages them.

e-Macao-16-6-23

J2EE Server

J2EE server provides the underlying services, such as:

1) transaction management
2) multithreading
3) resource pooling
4) other complex low-level services

e-Macao-16-6-24

J2EE Containers

Containers are the interface between a component and the low-level platform.

Types of containers:

1) EJB container
2) Web container
3) Application client container
4) Applet container

e-Macao-16-6-25

Task 1: Setup A Web Server

1) Setup and configure your `Tomcat` server for the usage in this course:
   a) copy the folder named `web_componet` to your local hard disk
   b) open the file `Tomcat.bat` and modify the path name if necessary
   c) Configure `Tomcat` to use port 80 as the default port:
      1. edit *<Catalina_Home>/conf/**server**.xml* and change the port attribute of the Connector element from 8080 to 80 as following:

```
<Connector port="80" ... maxThreads="150"
minSpareThreads="25" ...
```

This modification allows us to use the `URL` of the form `http://localhost/myServlet` instead of `http://localhost:8080/myServlet`

e-Macao-16-6-26

Task 2: Setup Web Server

d) configure Tomcat to activate the auto-reload feature:
   1. modify the `<Catalina_Home>/conf/Context.xml` file. Change the tag `<Context>` to `<Context reloadable="true">`. This allows Tomcat to reload the servlet automatically when a modification is made to the servlet. However, this setting may degrade the performance of the server.

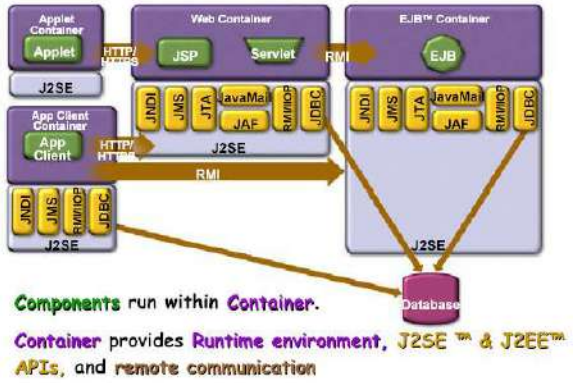
e) start Tomcat by runing `Tomcat.bat` bathc file

f) use a browser to access the following `URL`:

`http://localhost/`

e-Macao-16-6-27

Components and Containers



courtesy of Sun Microsystems

e-Macao-16-6-28

J2EE Components

1) Clients
   a) Web client
   b) Applet
   c) Application client
2) Web Components
   a) Servlet
   b) JavaServer Pages (JSP)
3) Business Components
   a) Enterprise Java Bean (EJB)
4) Enterprise Information System (EIS)
   a) Database

e-Macao-16-6-29

Web Components

In J2EE specification, v1.4, a web component is defined as a collection of:

a) Servlets
b) pages created with the JavaServer Pages™ technology
c) web Filters
d) web Event Listeners

In short, a web component is a software entity that runs in a web container to provide responses to external requests.

e-Macao-16-6-30

Business Components

The Enterprise Java Bean (EJB) architecture is a server-side technology for building object-oriented business application in Java.

There are three types of Enterprise Beans:

a) Session Beans
b) Entity Beans
c) Message-Driven Beans

e-Macao-16-6-31

J2EE Standard Services

J2EE standard services include the following:

- HTTP
- HTTPS
- RMI-IIOP
- JavaIDL
- Java Naming and Directory Interface (JNDI)
- JDBC API
- Java Message Service (JMS)
- Java Transaction API (JTA)
- JavaMail
- JavaBeans Activation Framework (JAF)
- Java API for XML Parsing (JAXP)
- J2EE Connector Architecture
- Security Services
- Web Services
- Management
- Deployment

Some of them are explained as follows.

e-Macao-16-6-32

J2EE Services: JNDI

1) Java naming and directory services (`JNDI`)
   a) applications use `JNDI` to locate objects, such as environment entries, EJBs, datasources or message queues
   b) JNDI is implementation independent
   c) underlying implementation varies: `LDAP, DNS, DBMS`, etc.

e-Macao-16-6-33

J2EE Services: JDBC

2) Java DataBase Connectivity (`JDBC`)
   a) a programming interface that lets Java applications access a database via the `SQL` language
   b) allows the development of platform-independent database applications

e-Macao-16-6-34

J2EE Services: JMS

3) Java Message Service (JMS)
   a) provides standard APIs that Java developers can use to access the common features of an enterprise message system
   b) supports publish/subscribe and point-to-point models
   c) provides support for administration, security, error handling, optimization, distributed transactions, message ordering, message acknowledgments, and more

e-Macao-16-6-35

J2EE Services: JTA

4) Java Transaction API (JTA)
   a) an application-level interface used to define the transaction boundaries
   b) allows applications to perform distributed transactions

e-Macao-16-6-36

J2EE Services: JavaMail

5) JavaMail
   a) a platform-independent Java API allowing to develop email clients or servers using any of the standard email protocols

e-Macao-16-6-37

J2EE Services: JAF

6) JavaBeans Activation Framework (JAF)
   a) used by JavaMail to convert the MIME byte streams into Java objects that can then be handled by assigned JavaBeans

e-Macao-16-6-38

J2EE Services: JAXP

7) Java API for XML Parsing (JAXP)
   a) includes both Simple API for XML (SAX) and Document Object Model (DOM) APIs for manipulating XML documents
   b) enables Extensible Stylesheet Language Transformation (XSLT) engines to be plugged in

e-Macao-16-6-39

J2EE Services: Connector

8) J2EE Connector Architecture
   a) integration with non-J2EE systems, such as mainframes and ERPs (Enterprise Resource Planning)
   b) standard API to access different EIS (Enterprise Information Systems)
   c) vendors implement EIS-specific resource adapters

e-Macao-16-6-40

J2EE Services: Security

9) Security Services
   a) Java Authentication and Authorization Service (JAAS)
   b) authentication via user identification / password or digital certificates
   c) role-based authorization limits access of users to the resources (URLs, EJB methods)

e-Macao-16-6-41

J2EE Platform Roles 1

A set of roles to carry out application development:

1) **J2EE product provider**: implements a J2EE product which provides component containers, J2EE platform APIs, and other features defined in the J2EE specification

2) **application component provider**: produces the building blocks of a J2EE application

3) **application assembler**: takes a set of components developed by application component providers and assembles them into a complete J2EE application.

e-Macao-16-6-42

J2EE Platform Roles 2

4) **deployer**: responsible for deploying J2EE components and applications into a specific operational environment

5) **system administrator**: responsible for configuring and administrating the computing and networking infrastructure of an enterprise

6) **tool provider**: provides tools used for the development and packaging of application components.

e-Macao-16-6-43

J2EE Introduction Outline

1) Origin of J2EE
2) Architecture of J2EE
3) Summary

A.1.3. Summary

e-Macao-16-6-44

Summary 1

J2EE is designed to reduce the cost and complexity of developing distributed cross-platform enterprise applications

J2EE provides a standard platform for hosting J2EE applications

Other than the J2SE standard services, J2EE server also provides:

1) transaction management
2) multithreading
3) resource pooling
4) other low level services

e-Macao-16-6-45

Summary 2

J2EE platform uses a multi-tiered distributed application model.

A J2EE server contains a web component container and a business component container.

Components are executed and managed by the containers.

A.2. Vertical Concepts

Vertical Concepts

e-Macao-16-2-47

Course Outline

1) J2EE introduction

2) vertical concepts
   a) Servlets
   b) JavaServer Pages
   c) Filters

3) horizontal concepts
   a) exceptions
   b) database connectivity
   c) security
   d) internationalization

4) case study

A.2.1. Servlet

e-Macao-16-6-48

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-49

Java Servlets

A servlet is a Java Technology component that executes within the servlet container.

Typically, servlets perform the following functions:

a) process the `HTTP` request
b) generate the `HTTP` response dynamically

e-Macao-16-6-50

Servlet Container

A servlet container

1) is a special JVM (Java Virtual Machine) that is responsible for maintaining the life cycle of servlets

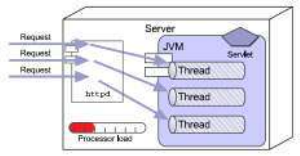
2) must support `HTTP` as a protocol to exchange requests and responses

3) issues threads for each request

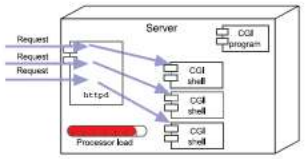
e-Macao-16-6-51

Servlets Versus CGIs

Servlets are lightweight and are scalable.



Each `CGI` is heavyweight and is not scalable.



e-Macao-16-6-52

Servlet Interface

All servlets either:

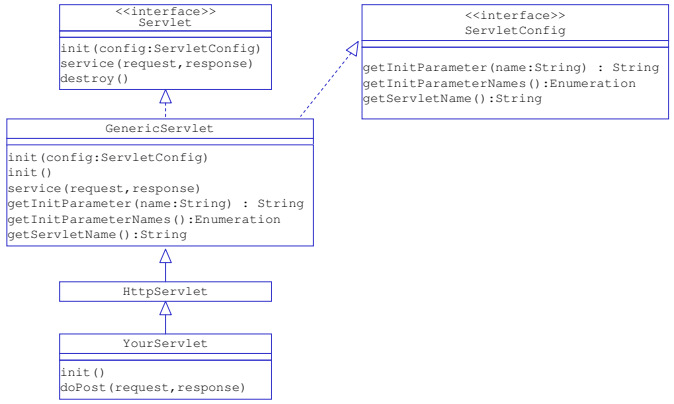
1) implement `javax.servlet.Servlet` interface, or
2) extend a class that implements `javax.servlet.Servlet`

In the Java Servlet API, classes `GenericServlet` and `HttpServlet` implement the `Servlet` interface.

`HttpServlet` is usually extended for Servlet implementation.

e-Macao-16-6-53

Servlet Architecture



e-Macao-16-6-54

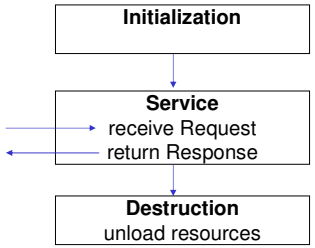
Servlet Life Cycle

Servlets follow a three-phase life cycle:

1) initialization

2) service

3) destruction



e-Macao-16-6-55

Life Cycle: Initialization 1

A servlet container:

a) loads a servlet class during startup, or

b) when the servlet is needed for a request

After the `Servlet` class is loaded, the container will instantiate it.

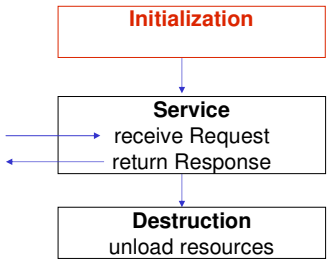
e-Macao-16-6-56

Life Cycle: Initialization 2

Initialization is performed by container before any request can be received.

Persistent data configuration, heavy resource setup (such as `JDBC` connection) and any one-time activities should be performed in this state.

The `init()` method will be called in this stage with a `ServletConfig` object as an argument.



e-Macao-16-6-57

Life Cycle: ServletConfig Object

The `ServletConfig` object allows the servlet to access name-value initialization parameters from the deployment descriptor file using a method such as `getInitParameter(String name)`.

The object also gives access to the `ServletContext` object which contains information about the runtime environment.

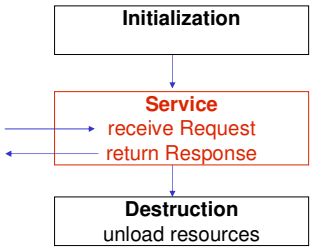
`ServletContext` object is obtained by calling to the `getServletContext()` method.

e-Macao-16-6-58

Life Cycle: Service 1

The service method is defined for handling client request.

The Container of a servlet will call this method every time a request for that specific servlet is received.

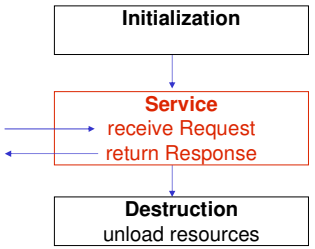


e-Macao-16-6-59

Life Cycle: Service 2

The Container generally handles concurrent requests with multi-threads.

All interactions to response to requests will occur within this phase until the servlet is destroyed.



e-Macao-16-6-60

Life Cycle: Service Method

The `service()` method is invoked to every request and is responsible for generating the response to that request.

The `service()` method takes two parameters:

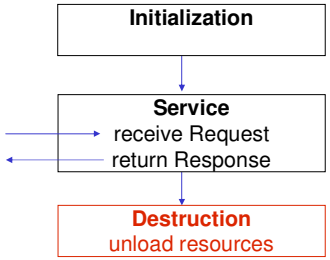
```
javax.servlet.ServletRequest
javax.servlet.ServletResponse

 public void service(ServletRequest request,
 ServletResponse response)throws IOException {
 . . .
}
```

e-Macao-16-6-61

Life Cycle: Destruction

When the servlet container determines that the servlet should be removed, it calls the `destroy` method of the servlet.

The servlet container waits until all threads running in the `service` method have been completed or time out before calling the `destroy` method.



e-Macao-16-6-62

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-63

HTTPServlet

A general servlet knows nothing about the `HyperText Transfer Protocol (HTTP)`, which is the major protocol used for Internet.

A special kind of servlet, `HTTPServlet`, is needed to handle requests from `HTTP` clients such as web browsers.

`HTTPServlet` is included in the package `javax.servlet.http` as a subclass of `GenericServlet`.

e-Macao-16-6-64

Hypertext Transfer Protocol

`Hypertext Transfer Protocol (HTTP)` is the network protocol that underpins the World Wide Web.

For example:

a) when a user enters a `URL` in a Web browser, the browser issues an `HTTP GET` request to the Web server
b) the `HTTP GET` method is used by the server to retrieve a document
c) the Web server then responds with the requested `HTML` document

e-Macao-16-6-65

HTTP Methods

Useful for Web applications:

`GET` - request information from a server

`POST` - sends an unlimited amount of information over a socket connection as part of the `HTTP` request

Not useful for Web applications:

`PUT` - place documents directly to a server

`TRACE` - debugging

`DELETE` - remove documents from a server

`OPTIONS` - ask a server what methods and other options the server supports for the requested resource

`HEAD` - requests the header of a response

e-Macao-16-6-66

Get Versus Post

`GET` request :	`POST` request :
provides a limited amount of information in the form of a query string which is normally up to 255 characters	sends an unlimited amount of information
visible in a `URL`	does not appear as part of a `URL`
must only be used to execute queries in a Web application	able to update data in a Web application

e-Macao-16-6-67

HTTP Request

A valid `HTTP` request may look like this:

```
GET /index.html HTTP/1.0
```

`GET`: is a method defined by `HTTP` to ask a server for a specific resource

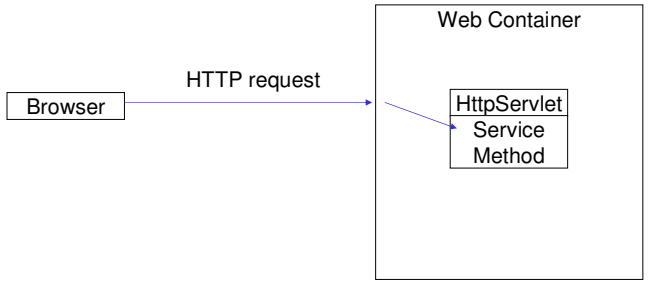
`/index.html`: is the resource being requested from the server

`HTTP/1.0`: is the version of `HTTP` being used

e-Macao-16-6-68

Handling HTTP Requests

A Web container processes `HTTP` requests by executing the `service` method on an `HttpServlet` object.

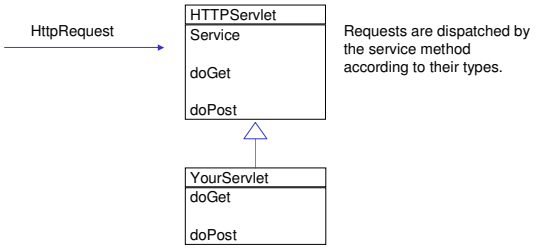


e-Macao-16-6-69

Dispatching HTTP Requests

In the `HttpServlet` class, the `service` method dispatches requests to corresponding methods based on the `HTTP` method such as `Get` or `Post`.

A servlet should extend the `HttpServlet` class and overrides the `doGet()` and/or `doPost()` methods.



e-Macao-16-6-70

HTTP Response

After a request is handled, information should be send back to the client.

In the `HTTP` protocol, an `HTTP` server takes a request from a client and generates a response consisting of

a) a response line

b) headers

c) a body

The response line contains the `HTTP` version of the server, a response code and a reason phrase :

```
HTTP/1.1 200 OK
```

e-Macao-16-6-71

HttpServlet Response

The `HttpServletResponse` object is responsible for sending information back to a client.

An output stream can be obtained by calls to:

```
1)getWriter()

2)getOutputStream()
```

For example:

```
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Hello World!</title>");
```

e-Macao-16-6-72

Task 3: HTTP Servlet

1) Create and deploy a `HelloWorld HTTP servlet` executing the Get method.

a) Declare the package – `com.examples`

b) Import the required clases:

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
import java.io.IOException;
```

c) The body of the servlet may look like this:

```
public class HelloServlet extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)throws IOException {

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
```

e-Macao-16-6-73

Task 4: HTTP Servlet

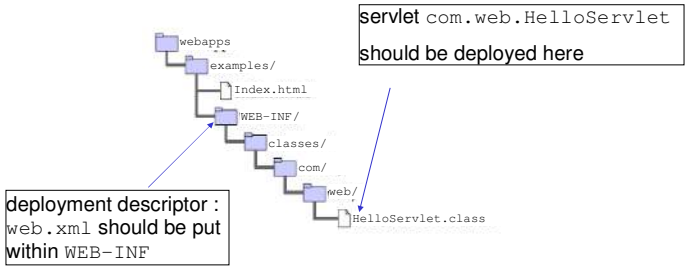
```
    //Generate the HTML response
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Hello Servlet</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY BGCOLOR='white'>");
    out.println("<B>Hello, World</B>");
    out.println("</BODY>");
    out.println("</HTML>");
    out.close();
  }
}
```

e-Macao-16-6-74

Deployment of an HTTP Servlet

The `HTTPServlet` object has to be deployed in the Web server before being used by the server.

A typical structure for deploying a servlet may look as follows:



e-Macao-16-6-75

Deployment Descriptor

In order to deploy a servlet, we also need to put a deployment descriptor file, `web.xml`, under the directory of the `WEB-INF` directory.

Within the `web.xml` file, the definition of the servlet is contained:

1) Define a specific servlet

```
<servlet>
      <servlet-name>name</servlet-name>
      <servlet-class>full_class_name</servlet-
class>
</servlet>
```

2) Map to a `URL` pattern

```
<servlet-mapping>
      <servlet-name>name</servlet-name>
      <url-pattern>pattern</url-pattern>
</servlet-mapping>
```

e-Macao-16-6-76

URL Patterns

There are four types of `URL` patterns:

a) Exact match:
```
<url-pattern>/dir1/dir2/name</url-pattern>
```

b) Path match:
```
<url-pattern>/dir1/dir2/*</url-pattern>
```

c) Extension match:
```
<url-pattern>*.ext</url-pattern>
```

d) Default resource:
```
<url-pattern>/</url-pattern>
```

e-Macao-16-6-77

Mapping Rules 1

When a request is received, the mapping used will be the first servlet mapping that matches the request's path according to the following rules:

a) If the request path exactly matches the mapping, that mapping is used.

1) If the request path starts with one or more prefix mappings (not counting the mapping's trailing "/*"), then the longest matching prefix mapping is used.
For example, `"/foo/*"` will match `"/foo"`, `"/foo/"`, and `"/foo/bar"`, but not `"/foobar"`.

e-Macao-16-6-78

Mapping Rules 2

3) If the request path ends with a given extension mapping, it will be forwarded to the specified servlet.

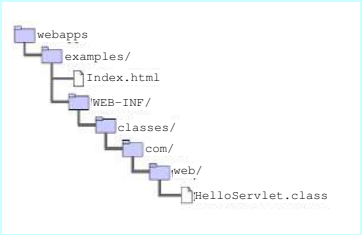
4) If none of the previous rules produce a match, the default mapping is used.

e-Macao-16-6-79

Task 5: Deploying HTTP Servlet

1) Deploy an `HTTP Servlet` in `Tomcat` server.

a) Create a directory for deployment. This directory, say `"examples"`, should be put under `<Tomcat_Home>/webapps`.



e-Macao-16-6-80

Task 6: Deploying HTTP Servlet

b) Refer to the directory structure in previous slide, copy the servlet package to the directory `WEB-INF/classes`.

c) Create a `web.xml` file, if one does not exist, in the directory `WEB-INF`. The file may look like the following:

```
<web-app
xmlns="http://java.sun.com/xml/ns/j2ee"
  version="2.4">
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>
     com.web.HelloServlet
  </servlet-class>
</servlet>
```

e-Macao-16-6-81

Task 7: Deploying HTTP Servlet

```
    <servlet-mapping>
            <servlet-name>HelloWorld</servlet-name>
            <url-pattern>/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>
```

d) Test the output of the servlet by entering the `URL` in the browser:
`http://localhost/examples/HelloWorld`

e-Macao-16-6-82

Task 8: Deploying HTTP Servlet

2) Change the URL address of the servlet to:
   http://localhost/examples/myservelt/HelloWorld

3) Change the URL address of the servlet to:
   http://localhost/examples/Hello

4) Deploy the servlet in a different context, say *admin*.
   The URL may look like this:
   http://localhost/admin/HelloWorld

e-Macao-16-6-83

Request Parameter

Data transmitted from a browser to a servlet is considered the request parameter.

A Web browser can transmit data to a Web server through HTML form.

For example, if the submit button of the following form is pressed, the corresponding data is sent to the Web server:


```
Get /servlet/myForm?name=Bryan HTTP/1.0
. . .
```

e-Macao-16-6-84

POST Method

By using a POST method, data may be contained in the body of the HTTP request:

```
POST /register HTTP/1.0
. . .
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 129
name=Bryan
```

The HTTP POST method can only be activated from a form.

e-Macao-16-6-85

Extracting Request Parameters

Request parameters are stored as a set of name-value pairs.

ServletRequest interface provides the following methods to access the parameters:

1) getParameter(String name)
2) getParameterValues(String name)
3) getParameterNames()
4) getParameterMap()

e-Macao-16-6-86

Task 10: Extract Parameter

1) Parameter value is sent to a servlet through an HTML form. Create a HTTP servlet to retrieve the value of the parameter.

a) Put the following HTML file in the examples folder of your web application, name it form.html and browse it.

```
<html>
 <BODY BGCOLOR='white'>
 <B>Submit this Form</B>
 <FORM ACTION='/examples/myForm' METHOD='POST'>
 Name: <INPUT TYPE='text' NAME='name'><BR><BR>
 <INPUT TYPE='submit'>
 </FORM>
 </BODY>
</html>
```

e-Macao-16-6-87

Task 11: Extract Parameter

b) Methods of the HttpServletRequest are available for extracting parameters from different HTML forms:
String getParameter(name) – get a value from a text field
String getParameterValues(name) – get values from a multiple selection such as a checkbox

c) Create a servlet named myForm and deploy it under the examples context. The servlet will extract the parameter "name" and generate an HTML page showing the name in bold type.

Make sure that your servlet implements the correct method to respond to the request.

e-Macao-16-6-88

Defining Initial Parameters

A servlet can have multiple initial parameters defined in the deployment descriptor (web.xml) as follows:

```
<servlet>
      <servlet-name>EnglishHello</servlet-name>
      <servlet-class>
            com.web.MultiHelloServlet
      </servlet-class>
      <init-param>
            <param-name>greetingText</param-name>
            <param-value>Welcome</param-value>
      </init-param>
      <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
      </init-param>
</servlet>
```

e-Macao-16-6-89

Getting Initial Parameter

There are different ways to obtain servlet initial parameters defined in web.xml. One is to override the init() method, which is defined in the GenericServlet class in your servlet.

The getInitParameter method of the GenericServlet class provides access to the initialization parameters for the servlet instance.

In the init() method, a greeting String may be defined as follows:

```
public void init(){
 . . .
      greeting = getInitParameter("greetingText");

 . . . }
```

e-Macao-16-6-90

Multiple Servlet Definition

Multiple "servlet definitions" can also be defined in a given servlet class. The following could be added to `web.xml` along with the previous slide.

```
<servlet>
      <servlet-name>ChineseHello</servlet-name>
      <servlet-class>
            com.web.MultiHelloServlet
      </servlet-class>
      <init-param>
            <param-name>greetingText</param-name>
            <param-value>     </param-value>
      </init-param>
      <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
      </init-param>
</servlet>
```

e-Macao-16-6-91

Task 12: Servlet Initial Parameter

1) Modify `HelloServlet` to create a `servlet` named `MultiHelloServlet` which will pick up the initial parameters defined earlier and display them on an `HTML` Web page.

Note: Don't forget to define the servlet-mapping tag correctly. You need to define two mappings for a single servlet.

e-Macao-16-6-92

Request Header

A servlet can access the headers of an `HTTP` request with the following methods:

1) `getHeader`

2) `getHeaders`

3) `getHeaderNames`

e-Macao-16-6-93

Request Attributes

Attributes are objects associated with a request. They can be access through the following methods:

1) `getAttribute`

2) `getAttributeNames`

3) `setAttribute`

An attribute name can be associated with only one value.

e-Macao-16-6-94

Reserved Attributes

The following prefixes are reserved for attribute names and cannot be used:

1) `java.`
2) `javax.`
3) `sun.`
4) `com.sun.`

e-Macao-16-6-95

Request Path 1

The request path can be obtained from this method:

`getRequestURI()`

The request path is composed of different sections.

These sections can be obtained through the following methods of the request object :

`getContextPath()`:

If the context of the servlet is the "default" root of the Web server, this call will return an empty string.

Otherwise, the string will starts with a ' / ' character but not end with a ' / ' character

e-Macao-16-6-96

Request Path 2

`getServletPath()`:

The mapping which activates this request:

If the mapping matches with the ' /* ' pattern, returns an empty string

Otherwise, returns a string starts with a ' / ' character.

Example:

Context path: /examples
Servlet mapping :
Pattern: /lec1/ex1
Servlet: exServlet

Request Path: /examples/lec1/ex1
ContextPath: /examples
ServletPath: /lec1/ex1
PathInfo: null

e-Macao-16-6-97

Request Path 3

`getPathInfo()`:

The extra part of the request `URI` that is not returned by the `getContextPath` or `getServletPath` method.

If no extra parts, returns null

otherwise, returns a string starts with a ' / ' character

Example:

Context path: /examples
Servlet mapping :
Pattern: /lec1/*
Servlet: exServlet

Request Path: /examples/lec1/ex/
ContextPath: /examples
ServletPath: /lec1
PathInfo: /ex/

e-Macao-16-6-98

Request Path 4

To sum up:

```
RequestURI = ContextPath + ServletPath + PathInfo
```

e-Macao-16-6-99

Task 13: Request Path

1) Modify the `HelloServlet` class to print out the results of the following methods:
   a) `getRequestURL()`
   b) `getRequestURI()`
   c) `getContextPath()`
   d) `getServletPath()`
   e) `getPathInfo()`

2) What is the result if entering this `URL` in the browser:
   `http://localhost/examples/HelloWorld/`

3) Modify the mapping for `HelloServlet` from `"HelloWorld"` to `"/*"` and check out the result again.

e-Macao-16-6-100

Response Headers

`HttpServletResponse` can manipulate the `HTTP` header of a response with following methods:

```
addHeader(String name, String value)
addIntHeader(String name, int value)
addDateHeader(String name, long date)

setHeader(String name, String value)
setIntHeader(String name, String value)
setDateHeader(String name, long date)
```

For example:
You can make the client's browser cache the common graphic of your web site as following:

```
response.addHeader("Cache-Control",
"max-age=3600");
```

e-Macao-16-6-101

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-102

Servlet Context

A `ServletContext` object is the runtime representation of a Web application.

A servlet has access to the servlet context object through the `getServletContext` method of the `GenericServlet` interface.

The servlet context object provides:

a) read-only access to context initialization parameters
b) read-only access to application-level file resources
c) read-write access to application-level attributes
d) write access to the application-level log file

e-Macao-16-6-103

Context Initial Parameters 1

The application-wide servlet context parameters defined in the deployment descriptor (`web.xml`) can be retrieved through the context object.

The `web.xml` file may look like the following:

```
<web-app>
      <context-param>
            <param-name>admin email</param-name>
            <param-value>admin@servlet.com</param-value>
      </context-param>
. . .
```

e-Macao-16-6-104

Context Initial Parameters 2

In order to obtain a context object, the following can be used:

```
ServletContext context =
getServletConfig().getServletContext();
```

After having the context object, one can access the context initial parameter like this:

```
String adminEmail =
context.getInitParameter("admin email");
```

e-Macao-16-6-105

Access to File Resources

The `ServletContext` object provides read-only access to file resources through the `getResourceAsStream` method that returns a raw `InputStream` object.

After having the servlet context object, one can access the file resources as follows:

```
String fileName = context.getInitParameter("fileName");
InputStream is = null;
BufferedReader reader = null;
try {
is = context.getResourceAsStream(fileName);
reader = new BufferedReader(new inputStreamReader(is));
. . .
```

e-Macao-16-6-106

Access to Attributes 1

The `ServletContext` object provides read-write access to runtime context attributes through the `getAttribute` and `setAttribute` methods.

Setting attributes:

```
ProductList catalog = new ProductList();
catalog.add(new Product("p1",10);
catalog.add(new Product ("p2",50);
context.setAttribute("catalog", catalog);
```

e-Macao-16-6-107

Access to Attributes 2

Getting attributes:

```
catalog =
  (ProductList) context.getAttribute("catalog");
Iterator items = catalog.getProducts();
while ( items.hasNext() ) {
  Product p = (Product) items.next();
  out.println("<TR>");
  out.println("<TD>" + p.getProductCode() + "</TD>");
  out.println(" <TD>" + p.getPrice() + "</TD>");
  out.println("</TR>");
}
```

e-Macao-16-6-108

Write Access to the Log File

The `ServletContext` object provides write access to log file through the `log` method.

The code may look as follows:

```
context.log ("This is a log record");
```

e-Macao-16-6-109

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-110

Servlet Communication

When building a Web application, resource-sharing and communication between servlets are important. This can be achieved through one of the following:

a) servlet context
b) request dispatching
c) session tracking
d) event listening

e-Macao-16-6-111

ServletContext

Servlets can access other servlets within the same servlet context through an instance of :

`javax.servlet.ServletContext`

The `ServletContext` object can be obtained from the `ServletConfig` object by calling the `getServletContext` method.

A list of all other servlets in a given servlet context can be obtained by calling the `getServletNames` method on the `ServletContext` object.

e-Macao-16-6-112

Accessing a Servlet

The following code snippet shows how to access another servlet through the servlet context instance.

```
    ...
     BookDBServlet database = (BookDBServlet)
    getServletConfig().getServletContext().getServlet
    ("bookdb");
     //Obtain a Servlet and call its public method
    // directly
    BookDetails bd = database.getBookDetails(bookId);
       ...
  }
}
```

e-Macao-16-6-113

Request Dispatching 1

A request may need several servlets to cooperate:

`RequestDispatcher` object can be used for redirecting a request from one servlet to another.

An object implementing the `RequestDispather` interface may be obtained from the `ServletContext` via the following methods:

1) `getRequestDispatcher`
2) `getNamedDispatcher`

e-Macao-16-6-114

Request Dispatching 2

The `getRequestDispatcher` method takes a string argument as the path for the located resources.

The pathname must begin with a "/" and is interpreted as relative to the current context root.

The required servlet is obtained and returned as a `RequestDispatcher` object.

e-Macao-16-6-115

Request Dispatching 3

The `getNamedDispatcher` method takes a string argument indicating the name of a servlet known to the `ServletContext`.

Servlets may be given names via server administration or via a web application deployment descriptor.

A servlet instance can be determined through its name by calling `ServletConfig.getServletName()`

If a servlet is known to the `ServletContext` by name, it is wrapped with a `RequestDispatcher` object and returned.

e-Macao-16-6-116

Example: Request Dispatching

```
...
   RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/response");
         if (dispatcher != null) {
             dispatcher.include(request, response);
    }
```

Note:

1) The `RequestDispatcher` object's *include()* method dispatches the request to the "response" servlet path (/response – in the `URL` mapping).

e-Macao-16-6-117

Using a RequestDispatcher

To use a request dispatcher, a developer needs to call either the `include` or `forward` methods of the `RequestDispatcher` interface as follows:

```
...
  dispatcher.include(request, response);
...
```

e-Macao-16-6-118

Include Method

The `include` method of the `RequestDispatcher` interface may be called at any time.

It works like a programmatic server-side include and includes the response from the given resource (`Servlet`, `JSP` page or `HTML` page) within the caller response.

The included servlet cannot set headers or call any method that affects the headers of the response. Any attempt to do so should be ignored.

e-Macao-16-6-119

Forward Method

The `forward` method of the `RequestDispatcher` interface may only be called by the calling servlet if no output has been committed to the client.

If output exists in the response buffer that has not been committed, it must be reset (clearing the buffer) before the target `Servlet's` service method is called.

If the response has been committed, an `IllegalStateException` must be thrown.

e-Macao-16-6-120

Task 14: Request Dispatcher

1) Create a servlet which dispatches its request to another servlet using both the `forward` and `include` methods of the `ServletContext`.
   a) create a servlet name `TestDispatherServlet1` as follows:

```
package com.web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestDispatcherServlet1 extends
  HttpServlet {
  private static final String forwardTo
    = "/DispatcherServlet2";
```

e-Macao-16-6-121

Task 15: Request Dispatcher

```
private static final String includeIn
    = "/DispatcherServlet2";

public void doGet(HttpServletRequest req,
  HttpServletResponse res)
    throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.print("<html><head>");
    out.print("</head><body>");
```

e-Macao-16-6-122

Task 16: Request Dispatcher

```
// Displaying Form
 out.print("<form action=\"");
 out.print( req.getRequestURI() );
 out.print("\" method=\"post\">");
 out.print("<input type=\"hidden\" name=\"mode\" ");
 out.print("value=\"forward\">");
 out.print("<input type=\"submit\" value=\" \"");
 out.print("> ");
 out.print(" Forward to another Servlet ..");
 out.print("</form>");
```

e-Macao-16-6-123

Task 17: Request Dispatcher

```
 out.print("<form action=\"");
 out.print( req.getRequestURI() );
 out.print("\" method=\"post\">");
 out.print("<input type=\"hidden\" name=\"mode\" ");
 out.print("value=\"include\">");
 out.print("<input type=\"submit\" ");
 out.print("value=\" \"> ");
 out.print(" Include another Servlet ..");
 out.print("</form>");

 out.print("</body></html>");
 out.close();
}
```

e-Macao-16-6-124

Task 18: Request Dispatcher

```
public void doPost(HttpServletRequest req,
 HttpServletResponse res)
 throws ServletException, IOException {
    res.setContentType("text/html");
    String mode = req.getParameter("mode");
    PrintWriter out = res.getWriter();
    out.print( "Begin...<br>");
    // Forwarding to Servlet2
    if(mode != null && mode.equals("forward")) {
    req.setAttribute("mode", "Forwarding Response..");
    req.getRequestDispatcher(forwardTo).forward(req,
    res);
 }
```

e-Macao-16-6-125

Task 19: Request Dispatcher

```
    // Including response from Servlet2

    if(mode != null && mode.equals("include")) {
    req.setAttribute("mode", "Including Response..");
    req.getRequestDispatcher(includeIn).include(req,
       res);
    }
  }
}
```

b) Map the servlet at "/DispatcherServlet1" in the `web.xml` file

e-Macao-16-6-126

Task 20: Request Dispatcher

2) Create a servlet as the target servlet built at task 14.
   a) Make sure the servlet is mapped correctly in the `web.xml` file. For instance:

```
<servlet>
  <servlet-name>DispatcherServlet2</servlet-name>
  <servlet-class>
      com.web.TestDispatcherServlet2
  </servlet-class>
 </servlet>
<servlet-mapping>
  <servlet-name>DispatcherServlet2</servlet-name>
  <url-pattern>/DispatcherServlet2</url-pattern>
</servlet-mapping>
```

e-Macao-16-6-127

Task 21: Request Dispatcher

   b) Obtain the attribute "`mode`" of the `Request` object sent from the `TestDispatcherServlet1` and display it

3) What is the difference between `include` and `forward` methods?

e-Macao-16-6-128

Session Tacking

`HTTP` is a stateless protocol and associating requests with a particular client is difficult.

Session tracking mechanism is used to maintain state about a series of requests from the same user.

`javax.servlet.http.HttpSession` defined in Servlet specification allows a servlet containers to use different approaches to track a user's session easily.

e-Macao-16-6-129

HttpSession

`HttpSession` is defined in the Servlet specification for managing the state of a client.

Each `HttpSession` instance is associated with an ID and can store the client's data.

The stored data will be kept privately until the client's session is destroyed.

e-Macao-16-6-130

Obtaining a Session

Servlets do not create sessions by default.

`getSession` method of the `HttpServletRequest` object has to be called explicitly to obtain a user's session.

For example:

```
public class CatalogServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
          HttpServletResponse response)
          throws ServletException, IOException {
      // Get the user's session
    HttpSession session = request.getSession();
     ...
      }
}
```

e-Macao-16-6-131

HttpSession Attributes

Objects, or data, can be bound to a session as attributes.

The following methods can be used to manipulate the attributes:

1) `void setAttribute(String name, Object value)`
   - binds an object to this session, using the name specified

2) `Object getAttribute(String name)`
   - returns the object bound with the specified name in this session, or `null` if no object is bound under the name

3) `Enumeration getAttributeNames()`
   - returns an `Enumeration` of `String` objects containing the names of all objects bound to this session

4) `void removeAttribute(String name)`
   - removes the object with the specified name from this session

e-Macao-16-6-132

Invalidating the Session

A user's session can be invalidated manually or automatically when the session timeouts.

To manually invalidate a session, the session's `invalidate` () method can be used:

```
. . .
HttpSession session = request.getSession();
. . .
// After the process, invalidate the session and clear
// the data
session.invalidate();
. . .
```

e-Macao-16-6-133

Cookies

Cookies are used to provide session ID and can be used to store information shared by the client and server.

When a session is created, an `HTTP` header, `Set-Cookie`, will be sent to the client. This cookie stores the session ID in the client until time-out.

The ID may looks like:

```
Set-Cookie:
JSESSIONID=50BAB1DB58D45F833D78D9EC1C5A10C5
```

This ID will be stored in a client and passed back to the server for each subsequent request.

```
Cookie: JSESSIONID=50BAB1DB58D45F833D78D9EC1C5A10C5
```

e-Macao-16-6-134

Cookie Object

Other than providing session ID, cookie can be used to store information shared by both the client and server.

`javax.servlet.http.Cookie` class provides methods for manipulating the information such as:

```
setValue(String value)
getValue()
setComment(String comment)
getComment()
setMaxAge(int second)
getMaxAge()
. . .
```

e-Macao-16-6-135

Using Cookies

A procedure for using cookies to store information in a client usually includes:

1) instantiating a cookie object
2) setting any attributes
3) sending the cookie

e-Macao-16-6-136

Instantiating a Cookie Object

```
public void doGet (HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {

     String bookId = request.getParameter("Buy");

     if (bookId != null) {
        Cookie Book = new

    Cookie("book_purchased",bookId);
          ...
     }
}
```

e-Macao-16-6-137

Setting Attributes

```
  . . .
    Cookie Book = new

     Cookie("book_purchased",bookId);

     Book.setComment ("Book sold" );
  . . .
}
```

e-Macao-16-6-138

Sending a Cookie

```
  . . .
  Cookie Book = new

     Cookie("book_purchased",bookId);

          Book.setComment ("Book sold" );

     response.addCookie(Book);
  . . .
}
```

e-Macao-16-6-139

Retrieving Information

A procedure to retrieve information from a cookie:

1) retrieve all cookies from the user's request
2) find the cookie or cookies with specific names
3) get the values of the cookies found

e-Macao-16-6-140

Retrieving a Cookie 1

```
public void doGet (HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException { ...

  String bookId = request.getParameter("Remove");
   ...
   if (bookId != null) {
     // Find the cookie that pertains to that book
        Cookie[] cookies = request.getCookies();
```

e-Macao-16-6-141

Retrieving a Cookie 2

```
      for(i=0; i < cookies.length; i++) {
          Cookie thisCookie = cookie[i];
          if (thisCookie.getName().equals

                        ("book_purchased") &&


thisCookie.getValue().equals(bookId)) {
          // Delete the cookie by setting its
          //  maximum age to zero

                thisCookie.setMaxAge(0);
                response.addCookie(thisCookie);
      }
```

e-Macao-16-6-142

Task 22: Cookie

1) Create a servlet that stores the last time the client visits this servlet within the session.
   a) `java.util.Date` could be used to obtain the time-stamp.
   b) The time-stamp should be stored as a cookie.
   c) A message similar to the following should be shown:
   ```
   Your last visit time is Fri Apr 01 14:37:48 CST 2005
   ```

e-Macao-16-6-143

URL Rewriting

If a client does not support cookies, URL rewriting could be used as a mechanism for session tracking.

While using this method, session ID is added to the URL of each page generated.

For example, after a session ID 123456 is generated, the rewritten URL might look like:

```
http://localhost/ServletTest/index.html;jsessionid=123456
```

e-Macao-16-6-144

Methods for URL Rewriting

The `HttpServletResponse` object provides methods for appending a session ID to a URL address string:

```
String encodeURL(java.lang.String url)
```

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

```
String encodeRedirectURL(String url)
```

Encodes the specified URL for use in the `sendRedirect` method or, if encoding is not needed, returns the URL unchanged.

e-Macao-16-6-145

Task 23: URL Rewriting

1) Investigate the usage of URL rewriting.
   a) Create a servlet, named "`URLRewrite`", which shows the following information on a web page:
      – request `URL` (`request.getURL()`)
      – request `URI` (`request.getURI()`)
      – servlet path (`request.getServletPath()`)
      – path info (`request.getPathInfo()`)
      – session id (`request.getSession().getId()`)
      – a hyperlink pointing to another servlet named "`DisplayURL`"
   b) Create a servlet `DisplayURL` which shows the session id.

e-Macao-16-6-146

Task 24: URL Rewriting

c) Make sure that the browser accepts cookies.

d) Browse the `URLRewrite` servlet and click on the link to the `DisplayURL` servlet. What is the session id displayed on both page?

e) Configure the browser to disable the cookies.

f) Repeat step d and observe the result.

g) Modify the `URLRewrite` servlet and call the `response.encodeURL` method to modify the hyperlink pointing to the`DisplayURL` servlet.

h) What is the result now and what is the conclusion?

e-Macao-16-6-147

Servlet Event Listener

Information about container-managed life cycle events, such as initialization of a web application or loading of a database could be useful.

Servlet event listener provides a mechanism to obtain these information.

e-Macao-16-6-148

Event Listener Interfaces

Interfaces of different event listeners:

```
javax.servlet.ServletRequestListener
javax.servlet.ServletRequestAttributeListener
javax.servlet.ServletContextListener
javax.servlet.ServletContextAttributeListener
javax.servlet.http.HttpSessionListener
javax.servlet.http.HttpSessionAttributeListener
```

e-Macao-16-6-149

Example of a Listener

A listener can be used in different situations and here is one of the examples:

1) When a web application starts up, the listener class is notified by the container and prepares the connection to the database.
2) When the application is closed and removed from the web server, the listener class is notified and the database connection is closed.

e-Macao-16-6-150

Task 24: Servlet Event Listener

1) Create `HttpSessionListener` which counts the number of users connected to the server concurrently.
   a) Create a class named `UserCounter` which implements the `HttpSessionListener`.
   b) Define a static integer variable "`counter`" for counting the users.
   c) Find out which methods are needed to implement the `HttpSessionListener` interface.
   d) Within which method should you add or deduct the number of users?
   e) Provide a static method `getUserCounted` to return the number of users connecting currently.

e-Macao-16-6-151

Task 25: Servlet Event Listener

2) Deploy the listener developed in Task 23.
   a) Modify the `web.xml` file as follows to deploy the listener:

```
<listener>
   <listener-class>
      FULL_CLASS_NAME_OF_THE_LISTENER
   </listener-class>
</listener>
```

3) Create a servlet `DisplayUser` which displays the number of connected user by calling the `getUserCount` method of the `UserCounter` class.

4) What can be observed when establishing more connections to the `DisplayUser` servlet?

e-Macao-16-6-152

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-153

Servlet Summary 1

The most commonly used servlet extends the `HttpServlet` class.

The life cycle of a servlet include initialization, service and destruction.

The web container dispatches the requests to the corresponding methods according to their types such as `Get` or `Post`.

The `URL` of a servlet could be mapped as part of the `web.xml` file.

e-Macao-16-6-154

Servlet Summary 2

Data transmitted from a browser to a servlet is considered a request parameter.

`ServletRequest` interface provides methods such as `getParameter()` for accessing the request parameters.

Initial parameters for a servlet could be assigned in the `web.xml` file and extracted within the `init()` method of a servlet using the `getInitParameter` method.

`HttpServletRequest` interface also provides methods for accessing different attributes of an `HTTP` request such as header, `URI`, etc.

e-Macao-16-6-155

Servlet Summary 3

`ServletContext` object is the runtime representation of a web application.

The servlet context object provides:

a) read-only access to context initialization parameters
b) read-only access to application-level file resources
c) read-write access to application-level attributes
d) write access to the application-level log file

e-Macao-16-6-156

Servlet Summary 4

Information and resources can be shared between servlets and the container.

Different approaches could be used:

a) servlet context
b) request dispatching
c) session tracking
d) event listening

A.2.2. JavaServer Pages

e-Macao-16-6-157

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-158

What is JavaServer Pages

`JavaServer Pages` is a `J2EE` technology for building web applications.

A `JSP` page is a textual document that describes how to create a dynamic response to a request.

`JSP` technology builds on:

1) template, or static, content

2) dynamic data

3) encapsulation of functionality through JavaBeans and tag libraries

e-Macao-16-6-159

Goals of JSP

While keeping the benefits of Java Servlet, JSP supports separation of presentation and business logic:

a) web designers can design and update pages without learning Java programming language

b) programmers for Java platform can write codes without dealing with web page design

How to achieve this?

JSP allows web designer to write standard HTML pages containing **tags** that run powerful programs based on Java technology.

e-Macao-16-6-160

Benefits of JSP

1) platform independent
2) roles separation
3) reuse of components and tag libraries
4) separation of dynamic and static content
5) encapsulation of functionality
6) integral parts of J2EE

e-Macao-16-6-161

Simple JSP Example 1

A JSP file may look as follows:

```
<%! private static final String GREETING = "HELLO"; %>
<HTML>
      <HEAD>
            <TITLE>Hello JavaServer Pages</TITLE>
      </HEAD>
<%
  String name = request.getParameter("name");
      if ( (name == null) || (name.length() == 0) ) {
        name = "DEFAULT_NAME";
  }
%>
```

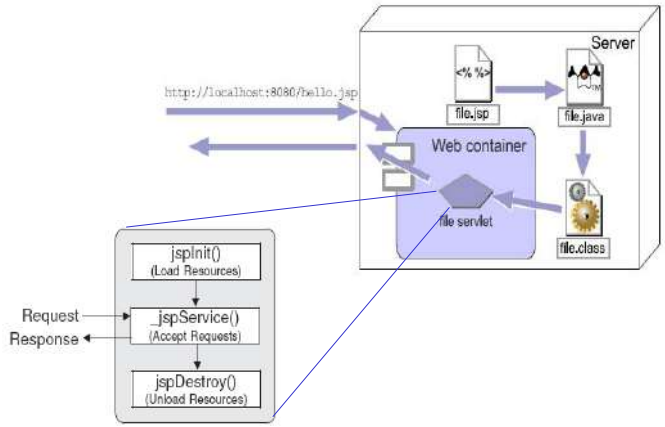
e-Macao-16-6-162

Simple JSP Example 2

```
<%-- THE FOLLOWING IS STANDARD HTML --%>
            <BODY BGCOLOR='white'>
            <B><%= GREETING %>, <%= name %></B>
      </BODY>
</HTML>
```

e-Macao-16-6-163

Life Cycle



e-Macao-16-6-164

Life Cycle Process

1) Web client transmits a request to the web container asking for a `JSP` page.
2) As this `JSP` page is accessed by the first time, it is translated into servlet code.
3) The servlet code is compiled into class file and loaded into the web container.
4) What is followed is similar to the working cycle of a normal servlet:
   a) The web container creates an instance of the servlet class for the `JSP` page and executes the `jspInit` method.
   b) The web container calls the `_jspService` method on the servlet instance for that `JSP` page. the result is sent back to the user.

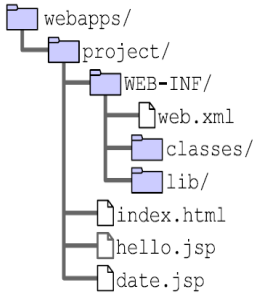
e-Macao-16-6-165

Deployment

`JSP` files can be placed under the deployment directory together with the main `HTML` files.

This allows the `JSP` files to be accessed as the main `HTML` files.

`JSP` files can also be mapped to specific `URLs` in the `web.xml` file.



e-Macao-16-6-166

Deployment Descriptor

The configuration information for `JSP` pages is described in the `web.xml` file rooted on the `<jsp-config>` element.

configuration elements may include:

`<taglib>` - element in mapping of tag libraries

`<jsp-property-group>` - properties of collections of `JSP` files, such as page encoding or automatic includes before and after pages, etc

e-Macao-16-6-167

Example: Deployment Descriptor

Common header and footer for JSP file can de defined in the web.xml file as follows:

```
<jsp-config>
     <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <include-prelude>/header.jsp</include-prelude>
        <include-coda>/footer.jsp</include-coda>
     </jsp-property-group>
</jsp-config>
```

e-Macao-16-6-168

Mapping JSP to a URL

Like a servlet, a JSP page can be mapped to a specific URL by modifying the web.xml file.

For example, mapping a JSP page named "ShowHello.jsp in deployment directory to "/Hello" may as follows:

```
<servlet>
     <servlet-name>ShowHello</servlet-name>
     <jsp-file>/ShowHello.jsp</jsp-file>
</servlet>

<servlet-mapping>
     <servlet-name>ShowHello</servlet-name>
     <url-pattern>/Hello</url-pattern>
</servlet-mapping>
```

full path name such as /WEB-INF/classes/com/ShowHello.jsp

e-Macao-16-6-169

Task 26: Mapping JSP

1) Investigate the mapping mechanism for JSP files.
   a) create a JSP file
   b) put it under the directory: <Your_Web_Context>/WEB-INF/classes/
   c) map this page with a name: myPage
   d) browse it with a web browser

e-Macao-16-6-170

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-171

Scripting Elements

Five kinds of scripting elements are defined in JavaServer Pages:

1) declarations <%! %>
2) scriptlets <% %>
3) expressions <%= %>
4) directives <%@ %>
5) comments <%-- --%>;<% /** **/%>;<!-- -->

e-Macao-16-6-172

Declarations

Declaration tag is used for declaring variables or methods.

Codes generated are outside of the `_jspService()` method.
Syntax: `<%! declaration %>`

Examples:

declaring a variable

```
<%! int i = 0; %>
```

declaring a method

```
<%! public String foo(int i)
        { if (i<3) return("small");
        }
%>
```

e-Macao-16-6-173

Scriptlets

The Java code within the scriptlet tag will be included in the `_jspService` method.

Syntax: <% *scriptlet* %>

Examples :

```
<% int time = 0; %>
<% if (time < 12) { %>
        Good Morning
<% } else { %>
        Good Afternoon
<% } ;%>
```

e-Macao-16-6-174

Expressions

The expression represents a runtime value which is generated for a response.

Syntax: <% *expression* %>

Examples :

```
<B>Thank you</B>, <I> <%= name %> </I>, for registering
```

e-Macao-16-6-175

Directives

Directives are used to define page attributes and do not output to a client.

Syntax: *<%@ directive {attribute="value"} *%>*

Directives can be as follows:

1) page
2) include
3) taglib

e-Macao-16-6-176

Directive: page

Provides page-specific information to a JSP container.

Syntax: `<%@ page %>`

The attributes include:

language	isErrorPage
extends	errorPage
import	contentType
session	pageEncoding
buffer	isScriptingEnabled
autoFlush	isELEnabled
isThreadSafe	

e-Macao-16-6-177

Directive: include

Includes text and/or code at translation time of a JSP.

Syntax: `<%@ include file="relativeURL" %>`

Example:

```
<%@ include file="header.jsp" %>
This is a page with predefined header and footer by
means of the include directive
<%@ include file="footer.jsp" %>
```

e-Macao-16-6-178

Task 27: Directive include

1) Create a JSP file named `header.jsp`.
   a) use declaration directive to declare an integer variable `"count"` for the page
   b) use declaration directive to declare a method `addCount()` which add the variable `"count"` by 1 for every call
   c) use scriptlet to call the `addCount` method
   d) append the following `HTML` code to the end of `header.jsp` which use the expression to display the dynamic content of `"count"`

```
<html>
   <body>
     <center>
        This page has been viewed <%= count %> times
     </center>
   <br>
```

e-Macao-16-6-179

Task 28: Directives include

2) Create a JSP file named footer.jsp.
   a) put HTML code to display the word "Welcome"
   b) append to the HTML code to the file

```
</body>
   </html>
```

3) Create a JSP file named body.jsp.
   1) use directive include to include the header.jsp at the beginning
   2) put a static statement
   3) use directive include to include the footer.jsp at the end

e-Macao-16-6-180

Directive: taglib

used to declare which markup on the page should be considered custom code and what code the markup links to

Syntax: `<%@ taglib uri="uri" prefix="prefixOfTag"%>`

This directive will be discussed in more detail in the session for tag library.

e-Macao-16-6-181

Comments

JSP file can use two different types of comments:

1) JSP document comment
   Examples:
   `<%-- comments ... --%>`, or
   `<% /** comments too ... **/ %>`
2) Comments send back to users as a response
   Examples:
   `<!-- comments ... -->`, or
   `<!-- comments <%=expression %> ... -->`

e-Macao-16-6-182

Guideline for Using Scripting

Overuse of scripting code can make JavaServer Pages confusing and difficult to maintain.

Scripting code will mix the business and presentation logic together.

Scripting code may reduce the reusability of JSP.

Scripting code should only be used when it is necessary.

e-Macao-16-6-183

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-184

JSP Implicit Objects

In servlet, objects such as `HttpServletResquest` or `HttpServletResponse` can be accessed directly.

In `JSP`, some objects are automatically declared by the web container and can be accessed directly by scripting elements. These objects are called implicit objects.

Examples:

`application`	`pageContext`
`config`	`page`
`session`	`out`
`request`	`exception`
`response`	

e-Macao-16-6-185

Implicit Objects: Servlet Equivalent

The following implicit objects have `Servlet` equivalents:

`application` – `javax.servlet.ServletContext`

`config` – `javax.servlet.ServletConfig`

`session` – `javax.servlet.http.HTTPSession`

`request` – `javax.servlet.http.HTTPRequest`

`response` – `javax.servlet.http.HTTPResponse`

e-Macao-16-6-186

Implicit Objects: JSP Specific

`JSP` defines some implicit objects as follows:

`pageContext` – an instance of `javax.servlet.jsp.PageContext` object
e.g. `pageContext.include("header.jsp");`

`page` – synonym for the "`this`" operator.

`out` – an instance of `javax.servlet.jsp.JspWriter` object

`exception` – an instance of `java.lang.Throwable` object

e-Macao-16-6-187

Example: Using Implicit Objects

The following JSP codes use the implicit object "request" to get information of the HTTP header and display it on a web page.

```
<%
 Enumeration enum = request.getHeaderNames();
 while (enum.hasMoreElements()) {
 String headerName = (String) enum.nextElement();
 String headerValue = request.getHeader(headerName);
%>
<b><%= headerName %></b>: <%= headerValue %><br>
<% } %>
```

can be used without declaring

e-Macao-16-6-188

Task 29: Implicit Objects

1) Investigate the usage of implicit objects.
   a) Referring to task 12, initial parameter was defined for a servlet in the `web.xml` file.
   b) Do the same setting for initial parameter in a JSP file named `ShowHello.jsp`.
   c) Call the `getInitParameter()` method of the implicit object "`config`" to get the initial parameter.
   d) Make `ShowHello.jsp` to display the greeting statement on the web page.

e-Macao-16-6-189

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-190

JSP Actions

JSP Actions have functions identical to that of scripting elements but allow abstraction of Java codes for JSP file.

JSP Actions have two types:

1) standard
2) custom

Syntax:

```
<prefix:element {attribute = "value"}* />
```

e-Macao-16-6-191

Standard JSP Actions

`Standard JSP Actions` are completely specified by the `JSP` specification and are available for use with any `JSP` container by default.

Include functionality that is commonly used with `JSP`.

A standard `JSP Action` generally use **`jsp`** as prefix.

Examples:
1) including resources (`<jsp:include/>`)
2) manipulating JavaBeans (`<jsp:useBean/>`)
3) forwarding requests (`<jsp:forward/>`)

e-Macao-16-6-192

Commonly used Standard Actions

Some of the commonly used standard actions will be discussed in this section:

1) `<jsp:include/>`
2) `<jsp:forward/>`
3) `<jsp:param/>`
4) `<jsp:useBean/>`
5) `<jsp:setProperties/>`
6) `<jsp:getProperties/>`

e-Macao-16-6-193

<jsp:include/>

include resources during **runtime**

Syntax:

```
<jsp:include page="urlSpec" flush="true|false"/>
```

Example:

```
<jsp:include page="include_page" flush="true"/>
```

e-Macao-16-6-194

<jsp:include/>: Attribute Flush

Attribute flush indicates whether any existing buffer should be flushed before reading in the included content.

The attribute flush is required in JSP 1.1 and should be set to `true`.

In JSP 1.2 and up, the flush attribute defaults to false and can be left off.

e-Macao-16-6-195

Task 30: <jsp:include/>

1) Investigate the operation of action `include`.
   a) With the `header.jsp` and `footer.jsp` developed in task 27 and task 28, create a `JSP` file named `actionBody.jsp` as follows:

```
<jsp:include page="header.jsp" />
This is a page with predefined header and footer by
means of the include action.
<jsp:include page="footer.jsp" />
```

   b) Browse the `actionBody.jsp` a few times.
   c) Modify the `footer.jsp` to add some text to it.
   d) Refresh the web pages.
   e) What observation did you have?

e-Macao-16-6-196

Task 31: <jsp:include/>

2) Compare with the result from using directive include.
   a) Browse the `body.jsp` developed in task 27 and 28 a few times.
   b) Repeat steps c and d in task 30.
   c) What is the difference comparing to the results of task 30.

e-Macao-16-6-197

<jsp:forward/>

Equivalent to call the `RequestDispather.forwared()` method.

This action forwards a request to a new resource and clears any content that might have previously been sent to the output buffer by the current `JSP`.

Example:

```
<jsp:forward page="relative_URL" />
```

e-Macao-16-6-198

<jsp:forward/>: Parameters

Both the `JSP` forward and include actions can optionally include parameters.

Example:

```
<jsp:forward page="examplePage.jsp">
<jsp:param name="para_1" value="val"/>
<jsp:param name="para_2" value="<%= aVal %>"/>
</jsp:forward>
```

The value can be represented by an expression.

If the parameter specified by the `param` action were exist, the existing is replaced.

e-Macao-16-6-199

JavaBean Actions

The Actions used with the JavaBean:

1) `<jsp:useBean/>`
2) `<jsp:setProperty/>`
3) `<jsp:getProperty/>`

e-Macao-16-6-200

JavaBeans

A JavaBean is a Java class with at least the following features:

1) accessors and mutators (get and set methods) are used to define the properties of the bean
2) has a default constructor
3) no public instance variables
4) not an `Enterprise JavaBeans (EJB)`

e-Macao-16-6-201

<jsp:useBean/>

Declares a JavaBean for use in a `JSP`.

Syntax:

```
<jsp:useBean id="name" class="full_class_name"
  scope="scope" />
```

Examples:

```
<jsp:useBean id="guestBean"
  class="com.web.GuestBean" scope="request"/>
<%
  guestBean.setName(request.getParameter("name"));
guestBean.setEmail(request.getParameter("email"));
%>
  </jsp:useBean>
```

e-Macao-16-6-202

<jsp:useBean/>: Usage

Action

```
<html>
<head>
<title>
      with useBean
</title>
</head>
<body>
<jsp:useBean id="date"
  class="java.util.Date"
/>
The date/time is
  <%= date %>
</body>
</html>
```

Scriptlet

```
<html>
<head>
<title>
      with Scriptlet
</title>
</head>
<body>
<% java.util.Date date =
  new java.util.Date();
%>
The date/time is
  <%= date %>
  </body>
  </html>
```

e-Macao-16-6-203

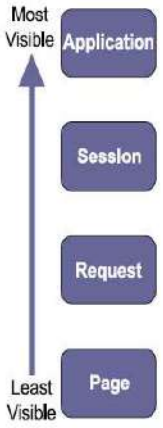
<jsp:useBean/>: Valid Scope 1

application

session

request

page



e-Macao-16-6-204

<jsp:useBean/>: Valid Scope 2

page:

1) The JavaBean will be available by calling the `getAttribute()` method of the `PageContext`.
2) The JavaBean is discarded upon completion of the current request.

request:

1) The JavaBean is available by calling the `getAttribute()` from the current page's `ServletRequest` object.
2) The JavaBean is discarded upon completion of the current request.

e-Macao-16-6-205

<jsp:useBean/>: Valid Scope 3

session:

1) The JavaBean is available by calling the `getAttribute()` from the current page's `HttpSession` object.
2) The JavaBean automatically persists until the session is invalidated.

application:

1) The JavaBean is available by calling the `getAttribute()` of the web application's `ServletContext` object.

e-Macao-16-6-206

<jsp:setProperty/>

The `jsp:setProperty` Action is used to initialize the JavaBean instead of using the scriptlet.

Exmaples:

```
<jsp:useBean id="guestBean"
      class="com.web.GuestBean" scope="request">
<jsp:setProperty name="guestBeam"
      property="name" value="Guest1">
<jsp:setProperty name="guestBean"
      property="email" />
</jsp:useBean>
```

`jsp:setProperty` Action can be used outside of the `jsp:useBean` Action.

e-Macao-16-6-207

<jsp:setProperty>: Attributes

To initialize the bean properties, the following settings can be used:

```
<jsp:useBean id="guestBean"
      class="com.web.GuestBean" scope="request">
<jsp:setProperty name="guestBean" property="*" />
<jsp:setProperty name="guestBean"
property="username" param="user"/>
<jsp:setProperty name="guestBean"
property="username" value="<%=user%>" />
</jsp:useBean>
```

when `property="*"` is used, the request parameters will be iterated to find the matched parameters.

e-Macao-16-6-208

<jsp:getProperty>

The `jsp:getProperty` Action is used to extract the value of an attribute of a JavaBean:

Example:

```
<jsp:getProperty name="guestBean"
      property="username"/>
```

e-Macao-16-6-209

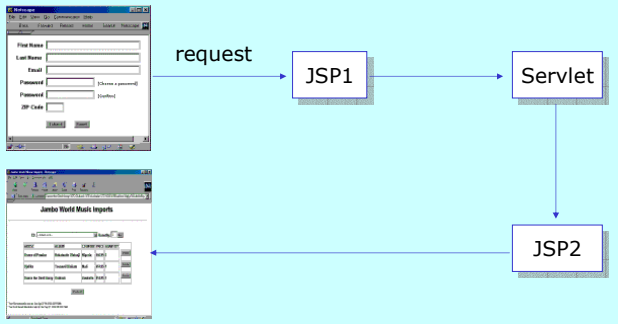
Task 32: <jsp:useBean/>

1) Investigate the usage of `useBean` Action.
   a) Develop a Java class named `User.java`.
   b) The class has two instance variables, `"name"` and `"password"`.
   c) Provide getter and setter for these two variables.
   d) Create a `JSP` file which shows an `HTML` form for user to input username and password.
   e) Information submitted from d) will be stored in an instance of `User` class.
   f) Create another `JSP` file named `displayeInfo.jsp` which displays the information of the `User` bean.
   g) Use `useBean`, `setProperty` and `getProperty` Actions in the `JSP` file.
   h) What is the difference applying different scopes for the useBean Action?

e-Macao-16-6-210

Task 33: <jsp:useBean/>

2) Use the useBean Action to perform request chaining. In this task, the following scenario is performed by modifying task 32.



e-Macao-16-6-211

Task 34: <jsp:useBean/>

a) Modify the `HTML` file in task 32 to show a form for entering user information.

b) Create a Java bean named `FormBean` for storing the information.

c) When the submit button of the form is pressed, the data is transmitted to a `JSP` file, say `Jsp1.jsp`.

d) `Jsp1` will instantiate the `FormBean`, using the `useBean` action with a scope of `request`.

e) Information from the request parameter is stored in the `FormBean` by calling the `jsp:setProperty` Action.

f) Use attribute `property="*"` to populate the data to the `FormBean`.

g) Request is then forwarded to the servlet, Servlet1, through the `jsp:forward` Action.

e-Macao-16-6-212

Task 35: <jsp:useBean/>

h) The controller servlet, Servlet1 extracts the bean passed to it from the attribute of the request as follows:

```
public void doPost (HttpServletRequest request,
HttpServletResponse response) {
  try {
    FormBean f = (FormBean)
                 request.getAttribute ("fBean");
  . . .
```

i) Modify the values of `UserBean` by calling the setter methods of the bean.

e-Macao-16-6-213

Task 36: <jsp:useBean/>

j) Forward the request to the `JSP` page, say `Jsp2.jsp` for rendering the output:

```
getServletConfig().getServletContext().
getRequestDispatcher("/Jsp2.jsp").forward(request,
response);
```

k) Extract the `UserBean` information by calling the `getProperty` action.

l) Display the user info on a web page.

e-Macao-16-6-214

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-215

JSP 2.0 Expression Language

JSP-specific expression language(`JSP EL`), is defined in JSP 2.0 specification.

JSP EL provides a cleaner syntax and is designed specially for JSP.

e-Macao-16-6-216

JSP EL Examples

A variable can be accessed as:

${variable_name}

The property of a bean can be accessed as:

${aBean.name}

Expression can be accessed as:

```
<c:if test="${aBean.age < 20}">
  . . .
</c:if>
```

e-Macao-16-6-217

JSP EL: Syntax

In `JSP EL`, expressions are always enclosed by the ${ } characters.

Any values not begin with **${** is literal.

Literal value contains the **${** has to be escaped with "\" character.

e-Macao-16-6-218

JSP EL: Attributes

Attributes are accessed by name, with an optional scope.

Members, getter methods, and array items are all accessed with a "**.**"

Examples:

a member b in object a → `${a.b}`
a member in an array a[b] → `${a.b} or ${a["b"]}`

e-Macao-16-6-219

JSP EL: Literals

`Boolean: true / false`
`Long:` as `long` values defined by Java

`Float:` as `float` values defined by Java

`String:` identical as in Java

`Null:` identical as in Java

e-Macao-16-6-220

JSP EL: Operators

`[]`

`()`

`-`

`*, /, div, %, mod`

`+, -`

`<, >, <=, >=, lt, gt, le, ge`

`&&, and`

`||, or`

Note: order of preference from top to bottom, left to right

e-Macao-16-6-221

JSP EL: Reserved Words

The following words are reserved and cannot be used in `JSP EL` expression:

`and`	`true`
`or`	`false`
`not`	`instanceof`
`eq`	`empty`
`gt`	`null`
`lt`	`div`
`ge`	`mod`
`ne`	
`le`	

e-Macao-16-6-222

JSP EL: Implicit Objects 1

A set of implicit objects is defined to match the `JSP` equivalents:

1) `pageContext:` the context for the `JSP` page

Through `pageContext`, the following implicit objects can be accessed:

a) `context`
b) `session`
c) `request`

For example, the context path can be accessed as:

```
${pageContext.request.contextPath}
```

e-Macao-16-6-223

JSP EL: Implicit Objects 2

2) `param`

a) maps name of parameter to a single string value

b) same as `ServletRequest.getParameter(String name)`

`e.g. ${param.name}`

3) `paramValues`

a) map name of parameter to an array of string objects

b) same as `ServletRequest.getParameterValues(String name)`

`e.g. ${paramValues.name}`

e-Macao-16-6-224

JSP EL: Implicit Objects 3

4) `header`

a) maps a request header name to a single string header value

b) same as `ServletRequest.getHeader(String name)`

`e.g. ${header.name}`

5) `headerValues`

a) map request header names to an array of string objects

b) same as `ServletRequest.getParameterValues(String name)`

`e.g. ${headerValues.name}`

e-Macao-16-6-225

JSP EL: Implicit Objects 4

Additional implicit objects are available for accessing scope attributes:

1) `pageScope`

2) `requestScope`

3) `sessionScope`

4) `applicationScope`

For example:

`${sessionScope.user.userid}`

e-Macao-16-6-226

JSP EL: Implicit Objects 5

5) `headerValues`

a) maps all the header values

b) same as `ServletRequest.getHeaders()`

6) `cookie`

a) maps the single cookie objects that are available by invoking `HttpServletRequest getCookies()`

b) If there are multiple cookies with the same name, only the first one encountered is placed in the Map

e-Macao-16-6-227

JSP EL: Defining EL Functions 1

Static methods in a Java class can be used as JSP EL functions.

The name and signature of the function can be defined as follows:

<function> element in the Tag Library Descriptor file (TLD) is used for setting up the linkage

```
<taglib>
...
 <function>
    <name>myFunction</name>
    <function-class>
          com.functions.MyFunction
    </function-class>
```

e-Macao-16-6-228

JSP EL: Defining EL Functions 2

```
      <function-signature>
            String bar(String)
      </function-signature>
   </function>
</taglib>
```

The static method, bar(), defined in the class com.functions.MyFunction is now mapped in the JSP EL as a function named myFunction.

e-Macao-16-6-229

JSP EL: Using EL Functions

The previous EL functions can be used as following:

```
${bar('hello')}
```

If the function is defined in a non-default namespace, the prefix must be declared explicitly.

For example:

If bar function is declared in a tag library with a prefix f, the function may be declared as :

```
${f:bar('hello')}
```

e-Macao-16-6-230

JSP EL Compatibility

Using JSP EL may cause compatibility problems with JSP 1.2 and earlier code.

JSP EL is disabled by default if Servlet 2.3 defined web.xml file is used.

Applications uses the Servlet 2.4 defined web.xml file will enable JSP EL automatically.

e-Macao-16-6-231

Enabling / Disabling JSP EL

JSP page can use the isScriptingEnabled page directive to enable or disable JSP EL.

For example:

```
<%@ page isScriptingEnabled="true" %>
```

Element scripting-enabled in the web.xml is used to configure an application-wide usability.

For example:

```
. . .
<jsp-property-group>
    <scripting-enabled>true</scripting-enabled>
 . . .
```

e-Macao-16-6-232

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary

2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary

3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-233

Standard Tag Library

JavaServer Pages Standard Tag Library (JSTL) is an extended set of JSP standard actions includes the following tags:

1) Iteration and conditional
2) Expression Language
3) URL manipulation
4) Internationalization-capable text formatting
5) XML manipulation
6) Database access

e-Macao-16-6-234

Problems with JSP Scriptlet Tags

1) Java code is embedded within scriptlet tags.

2) Non-Java developer cannot understand the embedded Java code.

3) Java code within JSP scriptlets cannot be reused by other JSP pages.

4) Casting to the object's class is required when retrieving objects out of HTTP request and session.

e-Macao-16-6-235

Advantage of JSTL

1) JSTL tags are in XML format and can be cleanly and uniformly blended into a page's HTML markup tags.

2) JSTL tag libraries are organized into four libraries which include most functionality required for a JSP page and are easier for non-programmers to use

3) JSTL tags encapsulate reusable logic and allow to be reused.

4) No casting is requiring while using JSTL tags for referencing objects in the request and session.

5) JSP's EL (Expression Language) allows using dot notation to access the attributes of Java objects.

e-Macao-16-6-236

Disadvantage of JSTL

1) JSTL may add processing overhead to the server:

   like JSP scriplet, tag libraries are also compiled into a resulting servlet and then executed by the servlet container

2) JSTL tags only provide the typical operations but not all:

   scriptlets may be needed if the JSP pages need to do everything

e-Macao-16-6-237

Example: JSTL 1

1) Without JSTL, some scriptlets may look as follows:

```
<%
   List addresses =
   (List)request.getAttribute("addresses");
   Iterator addressIter = addresses.iterator();
    while(addressIter.hasNext()) {
      AddressVO address =
      (AddressVO)addressIter.next();
      if((address != null) {
%>
<%=address.getLastName()%><br/>
```

e-Macao-16-6-238

Example: JSTL 2

```
 <%
       }
       else {
 %>
       N/A<br/>
<%
       }
    }
%>
```

e-Macao-16-6-239

Example: JSTL 3

1) With JSTL, the previous may look as follows:

```
<%@ taglib prefix="c"
  uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach item=${addresses} var="address" >
  <c:choose>
    <c:when test="${address != null}" >
      <c:out value="${address.lastName}"/><br/>
    <c:otherwise>
       N/A<br/>
    </c:otherwise>
  </c:choose>
</c:forEach>
```

e-Macao-16-6-240

Using JSTL

JSTL is standardized, but not a standard part of JSP 1.2 or 2.0.

JSTL must be downloaded and installed separately before being used.

e-Macao-16-6-241

Task 37: Installing the JSTL

1) The JSTL will be installed and setup for used.
   a) Dowonload the library from this URL:
      http://www.apache.org/dist/jakarta/taglibs/standard/
   b) Unpack the file and two jar files are inside the /lib directory:
      a) jstl.jar
      b) standard.jar

e-Macao-16-6-242

Task 38: Installing the JSTL

   c) Copy the jar file from step b to the following directory: <Tomcat_Home>/common/lib.
   d) the jar files can also be copied to the /WEB-INF/lib directory under your application context.
   e) In the JSP page, the following tags can be used to refer to the installed JSTL:

```
<%@ taglib
uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
```

e-Macao-16-6-243

Organization of JSTL

The `JSTL` tags are organized into four libraries:

Library features	Recommended prefix
Core (control flow, URLs, variable access)	`c`
Text formatting	`fmt`
XML manipulation	`x`
Database access	`sql`

e-Macao-16-6-244

JSTL: Core Tags 1

The core tags can be subdivided into a few categories:

1) General-purpose
   a) `out`
   b) `set`
   c) `catch`
   d) `remove`
2) Flow control
   a) `forEach`
   b) `forTokens`

e-Macao-16-6-245

JSTL: Core Tags 2

3) conditional
   a) `if`
   b) `choose`
   c) `when`
   d) `otherwise`

e-Macao-16-6-246

JSTL: Core Tags 3

4) `URL` management
   a) `url`
   b) `import`
   c) `redirect`
   d) `param`

e-Macao-16-6-247

JSTL Tags: <c:out>

This tag evaluates the JSTL expression and send output to the page's current JspWriter object.

Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<html><head><title>&lt;c:out&gt;</title></head>
  <body>
   <c:out value="${'<tag> , &'}"/>
   <br>
   <c:out value='<tag> , &' escapeXml="false"/>
  </body>
</html>
```

e-Macao-16-6-248

Attributes of <c:out> tag

1) **value:** expression to be evaluated and send
2) **escapeXml:** default is true and characters <,>,&,' and " result in <, >, and &, ', and "
3) **default:** defines the default value to be used in case the expression fails or results in null

e-Macao-16-6-249

Task 39: <c:out> tag

1) Investigate the result of using different attributes of <c:out> tag.
   a) Follow the previous example to test the output.
   b) View the source of the web page to see the actual output of the page.
   c) What is the difference with different values of the attribute escapeXml?

e-Macao-16-6-250

JSTL Tags: <c:set>

This tag evaluates an expression and uses the results to set a scoped variable, a JavaBean or a java.util.Map object.

Examples:

```
<c:set value="expression" target="target object"
property="name of property" />

<c:set value="value" var="varName"/>
```

e-Macao-16-6-251

Attributes of <c:set> tag1

1) **value:** expression to be evaluated
2) **var:** the name of the result variable representing the evaluated result from value attribute
3) **scope:** scope of the object named by the var attribute including:
    1) page (default)
    2) request
    3) session
    4) application

e-Macao-16-6-252

Attributes of <c:set> tag 2

4) **target:** a JavaBean of a java.util.Map object whose property will be set
5) **property:** the name of the property of the target object which will be set by the value attribute

e-Macao-16-6-253

Usage of <c:set> tag

1) set a scoped variable for use later

   for example:

```
<c:set value="100" var="totalCost"
scope="session"/>
```

2) set the property of a JavaBean or Map object

```
<c:set value="pass" target="student_A"
property="grade"/>
```

e-Macao-16-6-254

JSTL Tags: <c:catch>

This tag provides a complement to the JSP error page mechanism.

It works as a try-catch statement.

Code surrounded by a catch tag will never cause the error page mechanism to be invoked.

If a var attribute is set, the exception will be stored in the variable identified by the var attribute.

The var attribute always has page scope.

e-Macao-16-6-255

JSTL Tags: <c:remove>

This tag is used to remove a scoped variable

For example:

```
<c:remove var="cart" scope="session"/>
```

e-Macao-16-6-256

JSTL Tags: <c:forEach>

This tag provides iteration over a collection of objects.

supports iteration over an array, `java.util.Collection`, `java.util.Iterator`, `java.util.Enumeration`, or a `java.util.Map`

Example:

```
<c:forEach var="name" varStatus="status"
      begin="expression" end="expression"
      step="expression">

      body content

</c:forEach>
```

e-Macao-16-6-257

Attributes of <c:forEach> tag1

`var:` defines the name of the current object, or primitive, exposed to the body of the tag during iteration

`items:` attribute defines the collection of items to iterate over

`varStatus:` defines the name of the scope variable that provides the status of the iteration

Properties of `varStatus` may be:

```
current
index
count
first
begin
end
step
```

e-Macao-16-6-258

Attributes of <c:forEach> tag 2

`begin:` an int value that sets where the iteration should begin

`end:` The end attribute is an int value that determines inclusively where the iteration is to stop

`step:` The step attribute is an int value that determines the “step” to use when iterating

e-Macao-16-6-259

Example: <c:forEach> tag 1

1) This example displays the `varStatus` value in a loop.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
<H2>froEach varStatus</H2>
<UL>
<c:forEach var="count" begin="0" end="10" step="2"
  varStatus="status">
<LI><c:out value="${count}<br>"
  escapeXml="false"/>
<c:out value="current: ${status.current}<br>"
  escapeXml="false"/>
<c:out value="index: ${status.index}<br>"
  escapeXml="false"/>
```

e-Macao-16-6-260

Example: <c:forEach> tag 2

```
<c:out value="count: ${status.count}<br>"
  escapeXml="false"/>
<c:out value="first: ${status.first}<br>"
  escapeXml="false"/>
<c:out value="begin: ${status.begin}<br>"
  escapeXml="false"/>
<c:out value="end: ${status.end}<br>"
  escapeXml="false"/>
<c:out value="step: ${status.step}<br>"
  escapeXml="false"/>
</c:forEach>
</UL>
```

e-Macao-16-6-261

Example: <c:forEach> tag 3

2) This example uses the `<c:forEach>` tag to loop through an array and display on the web page.

```
<% String[] words = { "foo", "bar", "baz"};
pageContext.setAttribute("words", words); %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
<html>
<head>
<H2>Key Words:</H2>
</head>
<body>
<UL>
```

e-Macao-16-6-262

Example: <c:forEach> tag 4

```
  <c:forEach var="word" items="${words}">
  <LI><c:out value="${word}"/>
  </c:forEach>
</UL>
<H2>Values of the test Parameter:</H2>
 <c:forEach var="val" items="${paramValues.test}">
   <LI><c:out value="${val}"/>
 </c:forEach>
</body>
</html>
```

e-Macao-16-6-263

JSTL Tags: <c:forTokens>

This tag parses a `String` into tokens based on a given delimiter.

It works similar to the `forEach` tag with an extra attribute `delime` specifying a token delimiter.

Example:

```
<c:forTokens var="name" delime=","
items="Bryan,Frank,Gab">
 <c:out value="${name}"/>
</c:forTokens>
```

e-Macao-16-6-264

JSTL Tags: <c:if>

This tag works similar to a Java `if` and `switch`.

Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
. . .
<c:if test="${user == null}">
<form>
  Name: <input name="name">
  Pass: <input name="pass">
</form>
</c:if>
. . .
```

e-Macao-16-6-265

Attributes of <c:if> tag

`test`: the condition for testing

`var`: an optional attribute that defines the name of a scoped variable

`scope`: defines the scope of the var attribute.
(page, request, session or application)

e-Macao-16-6-266

JSTL Tags: <c:choose> 1

for more than one options, use the `<c:choose>`, `<c:when>` and `<otherwise>` tag

Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
. . .
<c:choose>
  <c:when test="${user == null}">
    <form>
     Name: <input name="name">
     Pass: <input name="pass">
    </form>
```

e-Macao-16-6-267

JSTL Tags: <c:choose> 2

```
    </c:when>
    <c:otherwise>
       Welcome ${user.name}
    </c:otherwise>
 </c:choose>
. . .
</body>
</html>
```

e-Macao-16-6-268

Task 40: <c:choose> tag

1) Use tags `<c:choose>`, `<c:when>` and `<c:otherwise>` to develop a `JSP` page which generates the follows:

```
1  (small)
2  (small)
3  (small)
4  (medium)
5  (medium)
6  (medium)
7  (medium)
8  (large)
9  (large)
10 (large)
```

e-Macao-16-6-269

JSTL Tags: <c:url>

This tag provides automatically encoded `URLs`.

Session information and parameters are encoded with a `URL`.

This tag is used when client does not support cookies.

e-Macao-16-6-270

Attributes of <c:url> tag

`value:` provides the `URL` to be processed

`context:` defines the name of the context

`var:` exports the encoded `URL` to a scoped variable

`scope:` defines the scope of the `var` object

For example:

```
<c:url var="thisURL" value="newPage.jsp">
<c:param name="aVariable" value="${v.id}"/>
<c:param name="aString" value="Simple String"/>
</c:url>
<a href="<c:out value="${thisURL}"/>">Next</a>
```

The above generates a `URL` as follows:

```
newPage.jsp?aVariable=24&aString=Simple+String
```

e-Macao-16-6-271

JSTL Tags: <c:redirect>

This tag provides the functionality to call the `HttpServletResponse.sendRedirect` method.

It can have attributes as follows:

`url`: the `URL` the client should be redirected to

`context`: the context of the `URL` specified by the `url` attribute

e-Macao-16-6-272

JSTL Tags: <c:import>

This tag provides all of the functionality of the `include` Action.

It allows for inclusion of absolute `URLs`, e.g. the content from a different web site.

Example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<c:import url="http://www.yahoo.com" />
```

e-Macao-16-6-273

JSTL Tags: <c:param>

This tag is used within the body of `<c:import>` tag to set `URL` parameters.

Examples:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<c:import url="http://search.yahoo.com/search"
var="yahoo">
<c:param name="p" value="java" />
</c:import>
<c:out value="${yahoo}" escapeXml="false" />
```

e-Macao-16-6-274

Other Tags

Other than the core tags, there are tags for different purposes such as :

database tags:
`<sql:setDataSource>,<sql:query>,<sql:update>. . .`

formatting tags:
`<fmt:formatNumber>,<fmt:parseNumber> . . .`

internationalization tags:
`<fmt:setLocale>,<fmt:setBundle>. . .`

`XML` manipulation tags:
`<x:parse>,<x:if>,<x:choose>,<x:transform>. . .`

e-Macao-16-6-275

Custom Tags

Like HTML, custom tags abstract code behind markup and provide a clean separation between logic and content.

Custom tags are designed to be used easily for a non-programmer.

Unlike scriptlet, custom tags can be packaged into a JAR file and deployed across web applications.

e-Macao-16-6-276

When to Use Custom Tags

Custom tags can be used to embedded dynamic functionality in a JSP.

Examples:

1) support the View partition in a MVC (Model-View-Controller) design pattern
2) support multi-lingual site
3) produce different formats of output to different clients such as web browser, PDA or web application
4) complement to the JSTL to provide full support for conditionals and iterations

e-Macao-16-6-277

Simple JSP 2.0 Custom Tags

introduced in JSP 2.0 with a simple life cycle

easier to write and use than the classic custom tag handlers

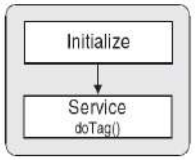
based on the javax.servlet.jsp.SimpleTag interface

e-Macao-16-6-278

Life Cycle

Has only two parts:

1) Initialization
   a) set the parent and body
   b) set by the JSP container
2) Service – doTag()
   a) implemented by the custom tag developer



e-Macao-16-6-279

SimpleTag Interface 1

All `SimpleTag` classes should implement the `javax.servlet.jsp.tagext.SimpleTag` interface

The interface defines the following methods:

`doTag()` – implemented by the tag developer and invoked by a `JSP` container during execution

`getParent()` – returns the custom tag surrounding this tag

e-Macao-16-6-280

SimpleTag Interface 2

`setJspBody(javax.servlets.jsp.JspFragment)` – invoked by a JSP container during runtime before the `doTag()` method

`setJspContext(javax.servlets.jsp.JspContext)` – invoked by a JSP container during runtime before the `doTag()` method

`setParent(javax.servlets.jsp.JspTag` – invoked by a `JSP` container during runtime to set the current parent tag

e-Macao-16-6-281

How to Develop Simple Tags

The `javax.servlet.jsp.tagext.SimpleTagSupport` class is the base implementation of the `SimpleTag` interface.

A custom tag can extend `SimpleTagSupport` and override the `doTag()` method.

e-Macao-16-6-282

Task 41: Simple Custom Tag

1) Develop a simple tag.
   a) Create a class named `HelloSimpleTag`.
   b) This class should be a subclass of `SimpleTagSupport` class.
   c) Allow the tag output a string in the `doTag()` method.
   d) The class may look like the follows:

```
package web.jsp;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import javax.servlet.jsp.*;
import java.io.IOException;
public class HelloSimpleTag extends
SimpleTagSupport{
```

e-Macao-16-6-283

Task 42: Simple Custom Tag

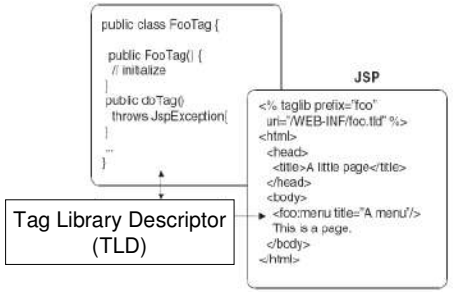
```
  public void doTag() throws JspException,
  IOException {
    JspWriter out = getJspContext().getOut();
    out.println("Hello World!");
  }
}
```

e-Macao-16-6-284

How to Use Custom Tags

A collection of custom tags designed for a common goal can be packaged into a library.

The custom tags within the library can be used by a JSP as described by a `Tag Library Descriptor` (`TLD`) file.



e-Macao-16-6-285

Tag Library Descriptor 1

Tag Library Descriptor is an `XML` file with "`tld`" extension or a `JAR` file used to bind the custom tags to the markup appears in a `JSP` file.

For example, following `TLD` file will bind the `CountTag` to a `JSP` with a name "`count`":

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/
 web jsptaglibrary_2_0.xsd" version="2.0">
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD</short-name>
```

e-Macao-16-6-286

Tag Library Descriptor 2

```
    <tag>
      <name>count</name>
      <tag-class>com.web.CountTag</tag-class>
      <body-content>empty</body-content>
  </tag>
</taglib>
```

e-Macao-16-6-287

TLD: Tag Elements 1

All tag definitions must be nested inside the <taglib> element.

The following tags are mandatory and should appear only once:

```
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>Example TLD</short-name>
```

e-Macao-16-6-288

TLD: Tag Elements 2

Each tag is defined by a <tag> element.

Within the <tag> element, the following attribute tags could be defined:

<name>: unique element name of the custom tag

<tag-class>: full class name for the tag class

<body-content>: types of code allowed to be inserted into the body of the custom tag when used by a JSP:

1) empty – tag body should be empty
2) JSP – tag body may be empty or containing scripting elements
3) scriptless – no scripting elements allowed
4) tagdependent – the body may contain non-JSP content like SQL statements

e-Macao-16-6-289

Task 43: Custom Tag Library

1) Follow the example to create a custom tag library which defines the HelloSimpleTag with a name "hello".
   a) Modify the name element.
   b) Save the file as example.tld in the WEB-INF directory.

e-Macao-16-6-290

Using Tag Library

A tag library can be referenced and used in a JSP by different methods.

Two of them are:

1) define a relative URI in JSP file
2) define a web application-wide URI

e-Macao-16-6-291

TLD: Relative URI

A relative URI can be defined in JSP file without a protocol and host.

For example:

```
<%@ taglib uri="/WEB-INF/example.tld" prefix="ex" %>
<html>
  . . .
  <ex:hello/>
  . . .
```

Note:

A root-relative URI should start with a "/", while a non-root-relative URI has no leading "/"

e-Macao-16-6-292

TLD: Application-Wide URI 1

An abstract URI can be defined by an entry in the web.xml file.

Example:

in web.xml file:

```
<taglib>
  <taglib-uri>
      http://www.example.com/example
  </taglib-uri>
  <taglib-location>
      /WEB-INF/example.tld
  </taglib-location>
</taglib>
```

e-Macao-16-6-293

TLD: Application-Wide URI 2

in JSP file:

```
<%@ tablib uri="http://www.exmple.com/example"
prefix="ex" %>
. . .
<ex:hello/>
```

e-Macao-16-6-294

Vertical Concepts Outline

e-Macao-16-6-295

Summary 1

JSP can produce dynamic content using scriptlet or tags.

While keeping the benefit of Servlet, JSP also provides separation between business and presentation logic for a web application.

Using tags in JSP allows the separation to be achieved easily.

e-Macao-16-6-296

Summary 2

The life cycle of a JSP is similar to that of a Servlet except a JSP file has to be compiled into a Servlet class before being used.

Five kinds of scripting elements can be used in JavaServer Pages:

1) declarations <%! %>
2) scriptlets <% %>
3) expressions <%= %>
4) directives <%@ %>
5) comments <%-- --%>;<% /** **/%>;<!-- -->

e-Macao-16-6-297

Summary 3

The following implicit objects are defined in JSP and can be used without declaration:

application	pageContext
config	page
session	out
request	exception
response	

e-Macao-16-6-298

Summary 4

JSP 2.0 specification defines Expression Language which provides a cleaner syntax then scriptlet.

JSP Actions can cooperate with JSP EL to provide a clean abstraction of Java codes making the JSP file easier to be maintained.

e-Macao-16-6-299

Summary 5

JavaServer Pages Standard Tag Library (JSTL) is an extended set of JSP standard Actions. Tags are available for the follows:

1) Iteration and conditional
2) Expression Language
3) URL manipulation
4) i18n-capable text formatting
5) XML manipulation
6) Database access

JSP 2.0 define a Simple Custom Tags which can be developed easily.

Tag Library Descriptor file is used to bind custom tag library to a JSP file.

A.2.3. Filters

e-Macao-16-6-300

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-301

Filter: Basic Concepts

`Filter` is a new feature in the servlet 2.3 specification.

`Filter` usually acts as a components between a request and a resource in a web application.

`Filters` can:

1) read request data
2) wrap request data
3) redirect a request
4) manipulate response data
5) generate its own response
6) wrap a response
7) return errors to the client

e-Macao-16-6-302

Filter: Advantages

Layers of `Filters` can be added for pre-processing and post-processing to a request and response.

`Filters` can perform similar functionality as `Servlets` and request dispatcher.

Unlike `Servlet` which had to be programmed differently for chaining, applying `Filters` to existing web application resources is easier.

e-Macao-16-6-303

Filter: Sample Applications

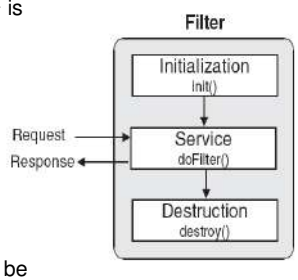
1) Access Control
   a) authentication, logging, auditing
   b) role-based security
   c) MIME-type redirection
2) Content Manipulation
   a) modify headers (request and response)
   b) data transformation
      - encryption, compression
      - `XSLT`, conversion

e-Macao-16-6-304

Filter: Life Cycle

`Filter's` life cycle mimics that of a `Servlet`:

1) initialization
   a) occurs only once when the `Filter` is first loaded
2) service
   a) occurs every time the `Filter` is accessed
3) destruction
   a) invoked after web application has finished using the `Filter`
   b) all resources of the `Filter` should be terminated



e-Macao-16-6-305

Filter: Interface

`javax.servlet.Filter`

a) For initialization:
```
public void init(FilterConfig config)
  throws ServletException
```
b) For service:
```
public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
  throws java.io.IOException, ServletException
```
c) For destruction:
```
public void destroy()
```

e-Macao-16-6-306

Filter: FilterConfig Object

`FilterConfig` object is used for `Filter` configuration.

`<init-param>` elements is used in the `web.xml` file, as for a servlet, to define custom initialization parameters

methods available:

```
String getFilterName()
String getInitParameter(String parameterName)
Enumeration getInitParameterNames()
ServletContext getServletContext()
```

e-Macao-16-6-307

Filter: FilterChain Object

`FilterChain` object is for invoking next `Filter` in chain (if any) or requested resource.

A method `doFilter` is defined for this purpose.

Methods available:

```
public void doFilter(ServletRequest request,
ServletResponse response)
throws java.io.IOException, ServletException
```

e-Macao-16-6-308

Filter: Deployment

`Filter` is deployed in a servlet container like a `servlet`.

Web application deployment descriptor file (`web.xml`) is also used for configuring a `Filter`.

`Filter` is defined via `<filter>` element in the `web.xml` file:

```
<filter>
        <filter-name>name</filter-name>
        <filter-class>class</filter-class>
        <init-param>
              <param-name>name</param-name>
              <param-value>value</param-value>
        </init-param>
        . . .
  </filter>
```

e-Macao-16-6-309

Filter: Mapping

Mapping of `Filter` is defined via `<filter-mapping>` element and has two forms:

a) Map to a specific `servlet` as follows:

```
<filter-mapping>
      <filter-name>name</filter-name>
      <servlet-name>name</servlet-name>
</filter-mapping>
```

b) Map to a `URL` pattern as follows:

```
<filter-mapping>
      <filter-name>name</filter-name>
      <url-pattern>pattern</url-pattern>
</filter-mapping>
```

e-Macao-16-6-310

Task 44: Simple Filter

1) A `Filter` can work as a normal Servlet. Try to build a simple `HelloWorld` Filter, deploy and test it.
   a) Create a `Filter` named "`FilterHelloWorld.java`". Note that `Filter` has to implement the `javax.servlet.Filter` interface.
   b) implement the init, `doFilter` and destroy methods. Don't do anything for init and destroy methods at this stage. Just try to implement `doFilter` to generate an `HTML` page showing a `String` "`HelloWorld`".
   c) Deploy your Filter at `Tomcat` and add the followings to the `web.xml` file:

```
<filter>
  <filter-name>FilterHelloWorld</filter-name>
  <filter-class>
        com.web.FilterHelloWorld
  </filter-class>
</filter>
```

e-Macao-16-6-311

Task 45: Simple Filter

```
<filter-mapping>
    <filter-name>FilterHelloWorld</filter-name>
    <url-pattern>/FilterHelloWorld</url-pattern>
</filter-mapping>
```

2) A `Filter` can do what a `Servlet` can. What is the difference between the `doFilter` method of a `Filter` and the service method of a `javax.servlet.Servlet` interface?

e-Macao-16-6-312

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

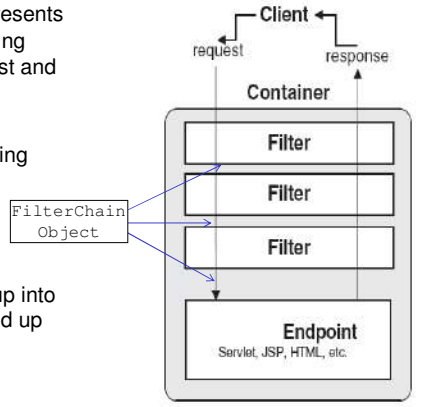
e-Macao-16-6-313

FilterChain Object

The `FilterChain` object represents the possible stack of Filters being executed on a particular request and response.

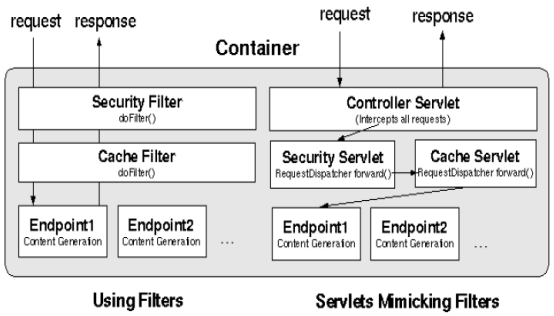
A mechanism for cleanly applying layers of functionality to a `ServletRequest` and `ServletResponse`.

Functionality is easily divided up into many logical layers and stacked up as desired.



e-Macao-16-6-314

Filter Versus Servlet



e-Macao-16-6-315

Defining a Filter Chain 1

To define a `filter` chain, put two or more filter declarations into the configuration file and supply appropriate values for the `<url-pattern>` elements .

For example, in the `web.xml` file, the following entries is set:

```
...
<filter>
      <filter-name>FilterAllRequests</filter-name>
      <filter-class>mypackage1.FilterOne</filter-class>
</filter>
<filter-mapping>
      <filter-name>FilterAllRequests</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

e-Macao-16-6-316

Defining a Filter Chain 2

```
<filter>
      <filter-name>FilterMyDocs</filter-name>
      <filter-class>mypackage1.FilterTwo</filter-class>
</filter>
<filter-mapping>
      <filter-name>FilterMyDocs</filter-name>
      <url-pattern>/mydocs/*</url-pattern>
</filter-mapping>
...
```

If a request for a resource like `http://127.0.0.1:8080/mydocs/foo.html` is received, the container will apply `FilterAllRequests` and `FilterMyDocs` according to their order appearing in the `web.xml` file.

e-Macao-16-6-317

Invoking Filter Chain

A `Filter` can invoke another Filter by calling the `doFilter` method of the `FilterChain` object.
For example: `chain.doFilter();`

The ordering of `Filter` execution matches the ascending order of filter-mapping elements defined in `web.xml` file.

```
public void doFilter(ServletRequest req, ServletResponse
res, FilterChain chain) throws IOException,
ServletException
{
      . . .
      chain.doFilter();
}
```

e-Macao-16-6-318

Task 46: Filter Chain

1) Add a hit counter `Filter` to a simple `hello.html` file.
   a) Create an `HTML` file as follows:

```
<html>
   <head>
      <title>HTML Page</title>
   </head>
   <body bgcolor="#FFFFFF">
            Hello World!
   </body>
</html>
```

e-Macao-16-6-319

Task 47: Filter Chain

b) Create a `Filter` for counting the hit rate for the `hello.html` page.
   1. This `Filter` has to implements `javax.servlet.Filter` interface.
   2. Declare a static integer variable count for counting the hit.
   3. Declare a `FilterConfig` object for the reference received from the init methods.
   4. There are three methods needed to be implemented: `init`, `doFilter` and `destroy`.
   5. Implement the `init` method. What type of argument should be received?
   6. Define an initial parameter named count in the `web.xml` file. Its initial value should be 0. Use corresponding method to get this initial parameter in the Filter.

e-Macao-16-6-320

Task 48: Filter Chain

7. Implements the `doFilter` method. The major task for the `doFilter` is add one to the counter and add a message to the response. The message may look like this: "The page has been viewed 3 times". You have your response object from the argument of the method and try to get a `PrintWriter` from there and write the message out to the response.
8. After modify the response, call the `doFilter` method of the `FilterChain` object received from the method's argument. this will pass the control to next filter or the end resource if no more filter exists.
9. Implement the destroy method to clear the `FilterConfig` object.

e-Macao-16-6-321

Task 49: Filter Chain

c) Deploy the web application correctly. In the `web.xml` file, define the `Filter` as follows:

```
. . .
 <filter>
    <filter-name>CounterFilter</filter-name>
    <filter-lass>
      your_filter_full_class_name
   </filter-class>
    <init-param>
       <param-name>Counter</param-name>
        <param-value>0</param-value>
     </init-param>
```

e-Macao-16-6-322

Task 50: Filter Chain

```
    </filter>

    <filter-mapping>
       <filter-name>CounterFilter</filter-name>
       <url-pattern>maping_pattern</url-pattern>
    </filter-mapping>
```

d) Change the value of `<url-pattern>` tag to allow `CounterFilter` to work for all `HTML` files.

e) Try access the `hello.html` and check out the result. Make sure to turn off the cache option of the browser.

e-Macao-16-6-323

Task 51: Filter Chain

2) Stack another `Filter` on top of the hit counter and named it `AuthenticationFilter`. This `Filter` is simplified for this exercise.

The functionality of the `Filter` is as follows:

When the client want to access the `hello.html` page, the `AuthenticationFilter` will check if the user has been login or not. A login page may show up if the user has not been login. Once the user pass the login process, a permission will be granted and the request will pass to the hit counter filter, followed by the `hello.html` page.

Please be noted that by using `Filter`, no change is made to the target resource, `hello.html`, at all.

e-Macao-16-6-324

Task 52: Filter Chain

a) The `AuthenticationFilter` may look as follows:

```
package com.web;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.*;
 import javax.servlet.http.*;
 public class AuthenticationFilter implements Filter {
         private FilterConfig config;

         public AuthenticationFilter() {
         super();
 }
```

e-Macao-16-6-325

Task 53: Filter Chain

```
// initialize the FilterConfig object.
// we are not using this object in this filter
public void init(FilterConfig config) throws
   ServletException {
         // TODO Auto-generated method stub
         this.config = config;
}
public void doFilter(
         ServletRequest req,
         ServletResponse res,
         FilterChain chain) throws IOException,
         ServletException {
```

e-Macao-16-6-326

Task 54: Filter Chain

```
String nextPage;
  HttpServletRequest request=(HttpServletRequest)req;
  HttpServletResponse response
   =(HttpServletResponse)res;
HttpSession = request.getSession();
String userName = request.getParameter("username");
String passWord = request.getParameter("password");
String login = (String)session.getAttribute("login");
```

e-Macao-16-6-327

Task 55: Filter Chain

```
// if the user has login already, pass to next filter
// make sure that you check if it is null
if (login!= null && (login.equals("true")) )
      chain.doFilter( req,res);
}
// print out the login in form, you may dispatch to
// other login page
else{
```

e-Macao-16-6-328

Task 56: Filter Chain

```
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println ("<form action="+uri+" method='POST'>") ;
out.println ( "<table>");
out.println ( " <tr><td>User:</td><td><input
  type='text' name='username'></td></tr>");
out.println ( "<tr><td>Password:</td><td><input
  type='password' name='password'></td></tr>");
```

e-Macao-16-6-329

Task 57: Filter Chain

```
    out.println ( " <tr><td colspan='2'><input
      type=submit></td></tr>");
    out.println ( " </table>");
    out.println ("</form>");
    }
   }
  }
 public void destroy() {
 }
}
```

e-Macao-16-6-330

Task 58: Filter Chain

b) Modify the `web.xml` file to stack the `AuthenticationFilter` on top of the `CounterFilter` as follows:

```
. . .

<filter>
   <filter-name>AuthFilter</filter-name>
   <filter-
   class>com.web.AuthenticationFilter</filter-class>
</filter>
```

e-Macao-16-6-331

Task 59: Filter Chain

```
<!-- the AuthenticationFilter is applied to all html
   files
-->
<filter-mapping>
   <filter-name>AuthFilter</filter-name>
   <url-pattern>*.html</url-pattern>
</filter-mapping>
. . .
```

e-Macao-16-6-332

Task 60: Filter Chain

b) Try to access the `hello.html` again.

c) Try to enter a wrong user name or password.

d) Try to use “user” as user name and “pass” as password to login in. What is the difference between this and step c?

e) Try to access the `hello.html` couple times and check the output. Make sure to disable the cache option of the browser.

e-Macao-16-6-333

Task 61: Filter Chain

f) Set up the `Tomcat` server to make session expire after 1 minute. Put the following statement to the `web.xml` file:

```
<session-config>
      <session-timeout>
          1 <!--minute-->
      </session-timeout>
</session-config>
```

g) Wait for a minute and try to access the `hello.html` again.

e-Macao-16-6-334

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-335

Filter Dispatcher

By default, Filters will only handle a request made by a client.

Request dispatched using either the `forward()` or `include()` methods of the `RequestDispatcher` object will not be handled.

This can be re-configured via `web.xml` file by using the `<dispatcher>` element as follows:

```
<filter-mapping>
 <filter-name>AuthenticationFilter</filter-name>
 <url-pattern>/*</url-pattern>
 <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

e-Macao-16-6-336

Filter Dispatcher Elements

There are four types of dispatcher elements:

1) `REQUEST`
2) `INCLUDE`
3) `FORWARD`
4) `ERROR`

More than one dispatcher elements can be used as the same time such as:

```
. . .
      <dispatcher>INCLUDE</dispatcher>
      <dispatcher>REQUEST</dispatcher>
. . .
```

Noted that if the dispatcher elements used, only the declared dispatcher call will be handled.

e-Macao-16-6-337

Vertical Concepts Outline

1) Servlet
   a) basic concepts
   b) http servlet
   c) servlet context
   d) communication between servlets
   e) summary
2) JavaServer Pages
   a) basic concepts
   b) scripting elements
   c) implicit objects
   d) actions
   e) expression language
   f) tag library
   g) summary
3) Filter
   a) basic concepts
   b) filter chain
   c) filter dispatcher
   d) wrapper
   e) summary

e-Macao-16-6-338

Filter: Wrapper

`Wrapper` is a new feature of `Filters` introduce in `Servlet` 2.3 specification.

A request or response can be wrapped inside a customized one.

Custom coding can then be used to manipulate the wrapped request and response.

Request and response are wrapped differently.

e-Macao-16-6-339

Invocating Wrappers in Filter

`Wrappers` are invocated within the `doFilter` method of a `Filter`:

```
public void doFilter(ServletRequest request,
   ServletResponse response FilterChain chain)
   throws java.io.IOException, ServletException {

     // Process request
     // Wrap request and/or response
     chain.doFilter(request or wrappedRequest,
              response or wrappedResponse);
     // Process (wrapped) response
   }
```

e-Macao-16-6-340

ServletRequest Wrapper

For `ServletRequest`, a corresponding `ServletRequestWrapper` is available for sublcassing as a wrapper:

```
javax.servlet.ServletRequestWrapper implements
ServletRequest
```

Constructor:

```
public ServletRequestWrapper(ServletRequest req)
```

e-Macao-16-6-341

HttpServletRequest Wrapper

For `HttpServletRequest`, a corresponding `HttpServletRequestWrapper` is provided for sublcassing as wrapper:

```
javax.servlet.HttpServletRequestWrapper extends
          ServletRequestWrapper implements
HttpServletRequest
```

Constructor:

```
public HttpServletRequestWrapper(HttpServletRequest
   req)
```

e-Macao-16-6-342

Task 62: Request Wrappers

1) Use a `Filter` to change the request headers before a `servlet` or `JSP` receives the request. A request wrapper is used to wrapped the request and pass it to the `FilterChain.doFilter( )` method, instead of the original request destination.

a) Create a class that extends `HttpServletRequestWrapper`.

```
import javax.servlet.http.HttpServletRequestWrapper;
import javax.servlet.http.HttpServletRequest;
import java.util.*;
public class RequestWrapper extends
 HttpServletRequestWrapper{
   public RequestWrapper(HttpServletRequest request){
      super(request);
   }
```

e-Macao-16-6-343

Task 63: Request Wrappers

```
    public Locale getLocale(){
        return new Locale("English", "Canada");
    }
}
```

b) Create a `Filter` name `RequestFilter` which uses the `RequestWrapper` to wrapped the `ServletRequest` and passes it to the target.

```
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestFilter implements Filter {
```

e-Macao-16-6-344

Task 64: Request Wrappers

```
    private FilterConfig config;
   public RequestFilter( ) {}
    public void  init(FilterConfig filterConfig)
    throws ServletException{
             this.config = filterConfig;
     }
     public void  doFilter(ServletRequest request,
     ServletResponse response, FilterChain chain)
      throws java.io.IOException,ServletException {
```

e-Macao-16-6-345

Task 65: Request Wrappers

```
    RequestWrapper wrapper = null;
     ServletContext context = null;
      if (request instanceof HttpServletRequest)
         wrapper = new
          RequestWrapper((HttpServletRequest)request);
          if (wrapper != null)
             chain.doFilter(wrapper,response);
      else
             chain.doFilter(request,response);
}
   public void destroy( ){}
}
```

e-Macao-16-6-346

Task 66: Request Wrappers

c) Modify the `web.xml` file as follows:

```
<servlet>
      <servlet-name>requestjsp</servlet-name>
      <jsp-file>/request.jsp</jsp-file>
</servlet>
<servlet-mapping>
      <servlet-name>requestjsp</servlet-name>
      <url-pattern>/requestjsp</url-pattern>
</servlet-mapping>
```

e-Macao-16-6-347

Task 67: Request Wrappers

```
<filter>
   <filter-name>RequestFilter</filter-name>
   <filter-class>com.web.RequestFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>RequestFilter</filter-name>
   <url-pattern>/requestjsp</url-pattern>
</filter-mapping>
```

d) Deploy the files and try to browse the file `/requestjsp` under the application context. Try to browse the file through `/request.jsp` under the application context. What is the difference?

e-Macao-16-6-348

Servlet Response Wrapper

Similar to request wrapper, there are `ServletResponseWrapper` and `httpServletResponseWrapper` availabe for subclassing to create the corresponding wrappers.

```
javax.servlet.ServletResponseWrapper implements
      ServletResponse
```

Constructor :

```
public ServletResponseWrapper(ServletResponse res)
```

e-Macao-16-6-349

HttpServlet Response Wrapper

For `HttpServletResponseWrapper`:

```
javax.servlet.http.HttpServletResponseWrapper
   extends ServletResponseWrapper
   implements HttpServletResponse
```

Constructor:

```
public HttpServletResponseWrapper
(HttpServletResponse response)
```

e-Macao-16-6-350

Task 68: Response Wrapper

1) Use `Filter` and `Wrapper` to compress the content requested by a client. A `Filter` is used to intercept the request for a web page and a response wrapper is used to capture the response and pass it through a `GZIPOutputStream` to compress the data before sending it to the client.
   a) Write a class named `GZIPResponseStream` extending the standard `ServletOutputStream`, which is used to send output to the client. Methods in the `ServletOutputStream` are overridden to write compressed response data out to the client. The header of the response should also be modified adding an entry "`Content-Encoding`". The skeleton code may look as follows:

e-Macao-16-6-351

Task 69: Response Wrapper

```
import java.io.*;
import java.util.zip.GZIPOutputStream;
import javax.servlet.*;
import javax.servlet.http.*;
public class GZIPResponseStream extends
  ServletOutputStream {
//declare variables
protected ByteArrayOutputStream baos = null;
protected GZIPOutputStream gzipstream = null;
protected boolean closed = false;
protected HttpServletResponse response = null;
protected ServletOutputStream output = null;
```

e-Macao-16-6-352

Task 70: Response Wrapper

```
// A constructor that receive the original response and
// replace the output stream with a GZIPOutputStream

public GZIPResponseStream(HttpServletResponse response)
throws IOException {
    super();
    closed = false;
    this.response = response;
    this.output = response.getOutputStream();
    baos = new ByteArrayOutputStream();
    gzipstream = new GZIPOutputStream(baos);
}
```

e-Macao-16-6-353

Task 71: Response Wrapper

```
// Override the close method that will modify the header
// entries such as "Content-Length" and
// "Content-Encoding" before closing the stream.
public void close() throws IOException {
    if (!closed) {
        throw new IOException("Stream closed"); }
    gzipstream.finish();
    byte[] bytes = baos.toByteArray();
    response.addHeader("Content-Length",
    Integer.toString(bytes.length));
    response.addHeader("Content-Encoding", "gzip");
    output.write(bytes);
```

e-Macao-16-6-354

Task 72: Response Wrapper

```
    output.flush();
    output.close();
    closed = true;
}

// Override the flush() and various write methods to
// use the gzipstream instead of the original stream

public void flush() throws IOException {
if (closed) {
      throw new IOException("Fail to flush"); }
   gzipstream.flush();
}
```

e-Macao-16-6-355

Task 73: Response Wrapper

```
public void write(int b) throws IOException {
  if (closed) {
      throw new IOException("Cannot write to a closed
       output stream");
  }
      gzipstream.write((byte)b);
      }
  flush();
  close();
}
```

e-Macao-16-6-356

Task 74: Response Wrapper

```
public void write(byte b[]) throws IOException {
  if (closed) {
  throw new IOException("Cannot write to a closed output
  stream"); }
  gzipstream.write(b, 0, b.length);
  flush();
  close();
}
```

e-Macao-16-6-357

Task 75: Response Wrapper

```
public void write(byte b[], int off, int len) throws
      IOException {
  System.out.println("writing...");
  if (closed) {
  throw new IOException("Cannot write to a closed output
  stream"); }
  gzipstream.write(b, off, len);
  flush();
  close();
}
}
```

e-Macao-16-6-358

Task 76: Response Wrapper

b) Write a class named `GZIPResponseWrapper` extends the `HttpServletResponseWrapper`. The main function of this wrapper is to replace the original `OutputStream` with a `GZIPResponseStream` that we defined in previous steps. The `getWriter()` is also overridden to obtain a writer from the `GZIPResponseStream`.
The skeleton code may look as follows:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

e-Macao-16-6-359

Task 77: Response Wrapper

```
public class GZIPResponseWrapper extends
   HttpServletResponseWrapper {
   protected HttpServletResponse origResponse = null;
   protected ServletOutputStream stream = null;
   protected PrintWriter writer = null;
// Constructor
public GZIPResponseWrapper
   (HttpServletResponse response) {
         super(response);
         origResponse = response;
}
```

e-Macao-16-6-360

Task 78: Response Wrapper

```
// Create a GZIPResponseStream from the original
   Response
public ServletOutputStream createOutputStream()
   throws IOException {
      return (new GZIPResponseStream(origResponse));
}
```

e-Macao-16-6-361

Task 79: Response Wrapper

```
// Overridden the getOutputStream and replace the
// ServletOutputStream with GZIPResponseStream
public ServletOutputStream getOutputStream() throws
IOException {
   if (writer != null) {
         throw new IllegalStateException(
         "getWriter() has already been called!");
   }
   if (stream == null)
         stream = createOutputStream();
   return (stream);
}
```

e-Macao-16-6-362

Task 80: Response Wrapper

```
// Overridden the getWriter and piped
// writer from the GZIPResponseStream
public PrintWriter getWriter() throws IOException {
   if (writer != null) {
         return (writer);
   }
   if (stream != null) {
    throw new IllegalStateException(
    "getOutputStream() has alreadybeen called!");
   }
```

e-Macao-16-6-363

Task 81: Response Wrapper

```
      stream = createOutputStream();
      writer = new PrintWriter
      (new OutputStreamWriter(stream, "UTF-8"));
      return (writer);
   }
}
```

e-Macao-16-6-364

Task 82: Response Wrapper

c) Write a class named `GZIPFilter` implements the `javax.servlet.Filter` interface. This `Filter` will check whether the client will accept gzip format. If so, the `Filter` will wrap the response with the `GZIPResponseWrapper` and let it compress the data. Otherwise, the ordinary response will be returned.
The skeleton code may look as follows:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GZIPFilter implements Filter {
   public void init(FilterConfig filterConfig) {
     //no implementation needed
}
```

e-Macao-16-6-365

Task 83: Response Wrapper

```
public void doFilter(ServletRequest req,
   ServletResponse res,FilterChain chain) throws
   IOException, ServletException {
   if (req instanceof HttpServletRequest) {
    HttpServletRequest request =
         (HttpServletRequest) req;
    HttpServletResponse response =
         (HttpServletResponse) res;
    String ae = request.getHeader("accept-encoding");
    if (ae != null && ae.indexOf("gzip") != -1) {
         System.out.println
         ("GZIP supported, compressing.");
```

e-Macao-16-6-366

Task 84: Response Wrapper

```
            GZIPResponseWrapper wrappedResponse = new
            GZIPResponseWrapper(response);
            chain.doFilter(req, wrappedResponse);
            return;
          }
          chain.doFilter(req, res);
    }
}
```

e-Macao-16-6-367

Task 85: Response Wrapper

d) In order to test the `Filter`, download a web page such as `www.yahoo.com/index.html` and save it under your web application context, e.g. `<TOMCAT_HOME>/webapps/FilterTest/`

e) Modify the `web.xml` file to apply the `GZIPFilter` to the downloaded page.

f) Browse the downloaded page and check the output from the console of `Tomcat`.

e-Macao-16-6-368

Vertical Concepts Outline

e-Macao-16-6-369

Filter: Summary 1

`Filter` is a new feature in the `servlet` 2.3 specification.

Filters are usually used for:
1) read request data
2) wrap request data
3) redirect a request
4) manipulate response data
5) generate its own response
6) wrap a response
7) return errors to the client

e-Macao-16-6-370

Filter: Summary 2

`Filter` can stack up as a `Filter chain` to provide layers of functionality to requested and response.

Functionality is easily divided up into many logical layers.

e-Macao-16-6-371

Filter: Summary 3

`Filters` do not handle dispatched request by default.

`<dispatcher>` element is used in the `web.xml` file to configure a `Filter` to handle dispatched requests.

e-Macao-16-6-372

Filter: Summary 4

`Wrapper` is a new feature of `Filters` introduce in `Servlet` 2.3 specification.

`Wrappers` are used to wrap and modify requests or responses.

Requests and responses are wrapped with different objects.

A.3. Horizontal Concepts

Horizontal Concepts

e-Macao-16-6-374

Course Outline

1) introduction

2) vertical concepts
   a) servlet
   b) java server page
   c) filters

3) horizontal concepts
   a) exception
   b) database connectivity
   c) security
   d) internationalization

4) case study

A.3.1. Exceptions

e-Macao-16-6-375

Horizontal Concepts Outline

1) Exceptions
   a) introduction
   b) error handling
   c) error objects
   d) logging

2) DataBase Connectivity
   a) jdbc review
   b) datasource
   c) connection pooling

1) Security
   a) introduction
   b) declarative security
   c) programmatic security
   d) secure communication

2) Internationalization
   a) introduction
   b) encoding
   c) resource bundles

3) Summary

e-Macao-16-6-376

Exception

If an exception is thrown from a `Servlets` or a `JSP`, it is passes to the container and the reactions will be depending on the container.

Create a `JSP` as follows and check out what is the response of the `Tomcat` server.

```
<%
if (true)
throw new Exception("An Exception thrown by JSP!");
%>
```

e-Macao-16-6-377

Handling Exception

Exceptions can be handled in following ways:

1) use `try-catch-finally` statements
2) forward the `HTTP` request to a `JSP` error page
3) forward the `HTTP` request to a `Servlet` to handle the error
4) declare error pages for specific error codes and allow the container to forward to these pages

e-Macao-16-6-378

Declaring Error Page

In `JSP`, the following directive forwards the request to "`myErrorPage.jsp`" when exception is thrown:

```
<%@page errorPage="myErrorPage.jsp" %>
```

The error can also be forward to a `Servlet` for handling the exception.

e-Macao-16-6-379

JSP Error Page

In `JSP`, the directive `<%@ page isErrorPage="true" %>` is used to declare that the file for exception handling.

Implicit object "`exception`" can be used in the error page to provide exception messages.

```
<%@ page isErrorPage="true" %>
<html>
. . .
<body>
This is the error message :<br>
"<%=exception.getMessage()%>"
</body>
</html>
```

Used to declarr that this is the error handling page

e-Macao-16-6-380

Task 86: JSP Error Handling

1) Test the error handling using a `JSP` error page.
   a) Define an application-wide parameter "`admin email`" in the `web.xml` file with a value "`admin@servlet.com`".
   b) Create a `JSP` page which will throw an error message as in the previous example. Remember to use the "`page`" directive with attribute "`errorPage`" correctly.
   c) Create an error handling page to catch the exception thrown by the `JSP` page at (b). The page needed to show the error message from the implicit object "exception" and the admin email address.
   d) What will happen when the "`isErroPage`" attribute is missing?

e-Macao-16-6-381

Task 87: JSP Error Handling

2) Test the error handling using a Servlet error page.
   a) Create a Servlet named "ErrorServlet".
   b) Within this Servlet, get the initial parameter "admin email" defined in the web.xml file.
   c) Within this Servlet, you can retrieve the Exception through the request object as following :

```
Exception e =
    (Exception)request.getAttribute
      ("javax.servlet.jsp.jspException");
```

   d) Modify the previous ThrowError.jsp as following to test the output:

```
<%@ page errorPage="ErrorServlet" %>
  <% if (true)  throw new Exception
     ("An Exception!");
  %>
```

e-Macao-16-6-382

Handling Specific Error

Error handling pages for specific exception can be declared in the web.xml file with the tag <error-page>.

Container will redirect the request to the specific page according to the exception occurred.

For example, in the web.xml file, error page can be defined as follows:

```
<error-page>
   <exception-type>java.lang.Exception</exception-type>
   <location>/Errorjsg.jsp</location>
</error-page>
<error-page>
   <error-code>404</error-code>
   <location>/Errorjsg.jsp</location>
</error-page>
```

Throwable

HTTP response code

e-Macao-16-6-383

Error Objects

The Servlet specification defines some attributes which can be retrieved from the request object for debugging:

javax.servlet.error.status_code
javax.servlet.error.exception_type
javax.servlet.error.message
javax.servlet.error.exception
javax.servlet.error.request_uri
javax.servlet.error.servlet_name

e-Macao-16-6-384

Task 88: Error Object

1) Create a JSP page which will send an email after receiving an error. The email will contain messages extracted from the error. The container will be configured to handle the forwarding operation.
   a) Download two packages from SUN and put inside the folder "<Tomcat_home>/common/lib":

      JavaMail: http://java.sun.com/products/javamail/

      JavaBeans Activation Framework(JAF) : http://java.sun.com/products/javabeans/glasgow/jaf.html

   b) Create a JSP named EmailErrorPage.jsp as follows:

```
<%@page isErrorPage="true" import="java.util.*,
 javax.mail.*, javax.mail.internet.*" %>
<%
Properties props = new Properties();
```

e-Macao-16-6-385

Task 89: Error Object

```
props.put("mail.smtp.host","smtp.macau.ctm.net")
;
Session msession =
 Session.getInstance(props,null);
String email =
 application.getInitParameter("lecturer email");
MimeMessage message= new MimeMessage(msession);
message.setSubject("[Application Error]");
message.setFrom(new InternetAddress(email));
message.addRecipient(Message.RecipientType.TO,
 new InternetAddress(email));
String debug = "";
```

e-Macao-16-6-386

Task 90: Error Object

```
Integer status_code
=(Integer)request.getAttribute
   ("javax.servlet.error.status_code");
if (status_code != null) {
 debug += "status_code: "+status_code.toString()
      + "\n";
}
 Class exception_type=
 (Class)request.getAttribute
   ("javax.servlet.error.exception_type");
if (exception_type != null) {
 debug += "exception_type:
   "+exception_type.getName() + "\n";
}
```

e-Macao-16-6-387

Task 91: Error Object

```
 String m=
(String)request.getAttribute
    ("javax.servlet.error.message");
if (m != null) {
    debug += "message: "+m + "\n";
 }

 Throwable e =(Throwable)
   request.getAttribute
    ("javax.servlet.error.exception");
if (e != null) {
  debug += "exception: "+ e.toString() + "\n";
 }
```

e-Macao-16-6-388

Task 92: Error Object

```
String request_uri =
 (String)request.getAttribute
    ("javax.servlet.error.request_uri");
if (request_uri != null) {
  debug += "request_uri: "+request_uri + "\n";
 }
String servlet_name=
 (String)request.getAttribute
    ("javax.servlet.error.servlet_name");
if (servlet_name != null) {
    debug += "servlet_name: "+servlet_name;
 }
```

e-Macao-16-6-389

Task 93: Error Object

```
message.setText(debug);
 Transport.send(message);
 %>
<html><head><title>EmailErrorPage</title></head>
<body>
   <h3>An Error Has Occurred</h3>
 This site is unavailable! requested.
   <br>Please send a description of the
   problem to:
 <a href="mailto:<%=email%>"><%=email%></a>.
</body>
</html>
```

e-Macao-16-6-390

Task 94: Error Object

c) Add a tag `<error-page>` to the `web.xml` file as follows:

```
<error-page>
   <error-code>404</error-code>
   <location>/EmailErrorPage.jsp</location>
 </error-page>
```

d) In the `EmailErrorPage.jsg` file, "`lecturer email`" is used as the email address for the sender and receiver for the email. Try to modify this to use different email addresses. However, the real email address is defined in the `web.xml` file as an initial parameter as follows:

```
<context-param>
    <param-name>lecturer email</param-name>
    <param-value>miltongm@gmail.com</param-value>
 </context-param>
```

Modify this to your own email address

e-Macao-16-6-391

Logging

Logging is used to keep a record of important information in some serialized form such as text file or information printed to `System.err` or `System.out`.

For constantly log information, a more robust logging API will be prefer than `System.out.println()` method.

Some Logging API:

1) `java.util.logging` package
2) `Log4J` (`jakarta.apache.org/log4j`)

e-Macao-16-6-392

Example: Logger 1

The following example shows the basic logging functionality of the `javax.util.logging` package.

```
<%@ page import="java.util.logging.*"%>
<% Logger logger = Logger.getLogger("example");
<% logger.setLevel(Level.ALL);
logger.addHandler(new FileHandler("/log.txt"));
 String info = request.getParameter("info");
 if (info != null && !info.equals("")) {
 logger.info(info);
 }
%>
```

e-Macao-16-6-393

Example: Logger 2

```
<html>
<head>
<title>A Simple Logger</title>
</head>
<body>
Logging examples
<form>
Information to log:<input name="info"><br>
<input type="submit">
</form>
</body>
</html>
```

e-Macao-16-6-394

Loggers and Levels

A Logger object is used to log messages for a specific system of application components.

`java.util.logging.Level` object is used to manage different types of logged information.

Types can be:

1) SEVERE
2) WARNING
3) INFO
4) CONFIG
5) FINE,FINNER AND FINNEST
6) OFF
7) ALL

e-Macao-16-6-395

Handlers

`java.util.logging` packages defines some `Handlers` for handling information.

1) `StreamHandler`: logged information is exported to a `java.io.OutputStream`.
2) `MemoryHandler`: `LogRecord` objects are kept in memory.
3) `SocketHandler`: information is logged using a network socket.
4) `FileHandler`: information is logged to a local file.

A.3.2. Database Connectivity

e-Macao-16-6-396

Horizontal Concepts Outline

1) Exceptions
   a) introduction
   b) error handling
   c) error objects
   d) logging

2) DataBase Connectivity
   a) jdbc review
   b) datasource
   c) connection pooling

1) Security
   a) introduction
   b) declarative security
   c) programmatic security
   d) secure communication

2) Internationalization
   a) introduction
   b) encoding
   c) resource bundles

3) Summary

e-Macao-16-6-397

JDBC Review 1

`JDBC` allows data stored in different databases to be accessed using a common Java API.

In general, Java applications that use a database almost always use `JDBC` to communicate with it.

e-Macao-16-6-398

JDBC Review 2

Important interfaces and classes:

`javax.sql.DataSource`–interface for obtaining connections to a database

`java.sql.Statement`–interface for executing `SQL` statements on a database

`java.sql.Connection`–object represents a physical connection with a database and is governed by underlying `JDBC` driver

`java.sql.ResultSet`–object returned as the results of an `SQL` statement

e-Macao-16-6-399

JDBC Review: DriverManger 1

Early version of `JDBC` may use an object called `DriverManger` to obtain the connection of a database as following:

```
String url = "jdbc:hsqldb:" + dbDir +"/my_database";
String user = "sa"; // hsqldb default
String password = ""; // hsqldb default
Class.forName("org.hsqldb.jdbcDriver");
Connection conn =
DriverManager.getConnection(url, user, password);
```

e-Macao-16-6-400

JDBC Review: DriverManger 1

The previous example has two problems:

1) The code is vendor specific.
2) The `DriverManager` is not an interface but a class and cannot be optimized by a Vendor easily.

e-Macao-16-6-401

DataSource

`DataSource` can solve the previous mentioned problems easily because `DataSource` is an interface which allows vendors' optimizations.

`DataSource` objects can be managed by container for higher efficiency.

Disadvantage:

`DataSource` needed to be configured in a container-dependent method.

e-Macao-16-6-402

Configuring DataSource 1

The following example shows the procedure for creating a datasource connecting a MySQL database to Tomcat server.

1) A database "dbTest" is assumed to have been created in MySQL already.
2) Downloaded and installed the required library as follows:
   a) Download the MySQL connector/J from www.mysql.com.

      file: mysql-connector-java-3.1.7.zip

      URL for download:
      http://dev.mysql.com/downloads/connector/j/3.1.html
   b) Extract the zip file and copy the file, mysql-connector-java-3.1.7-bin.jar, to <TOMCAT_HOME>/common/lib.

e-Macao-16-6-403

Configuring DataSource 2

The following steps will configure Tomcat with the DataSource connected to MySQL:

a) modify the <TOMCAT_HOME>/conf/server.xml by adding the following code segment within the tag <GlobalNamingResources>:

```
<Resource name="jdbc/Testdb"
            auth="Container"
            type="javax.sql.DataSource"
            driverClassName="com.mysql.jdbc.Driver"
 url="jdbc:mysql://localhost:3306/dbTest"
                  username="root"
                  password="1234"/>
```

e-Macao-16-6-404

Configuring DataSource 3

b) modify the <TOMCAT_HOME>/conf/context.xml by adding the following code segment within the tag <Context>:

```
<ResourceLink
      global="jdbc/Testdb"
      name="jdbc/Testdb"
      type="javax.sql.DataSource"/>
```

e-Macao-16-6-405

Configuring DataSource 4

c) The setting in step 2 can also be done as follows:

   Create a file META-INF/context.xml under the context of the web application. If the context of the web application is "dbTest", the context.xml may look as follows:

```
<Context docBace="dbTest" path="/dbTest"
  reloadable="true">
<ResourceLink global="jdbc/Testdb" name="jdbc/Testdb"
  type="javax.sql.DataSource"/>
</Context>
```

d) Restart Tomcat server and a DataSource is ready for connection.

e-Macao-16-6-406

Task 95: Connecting DataBase

1) Examine different ways, with and without `DataSource`, for connecting a database. A database named "`dbTest`" with a table "`testdata`" is assumed to have been created in a running `MySQL` server.
   a) Deploy the following Servlet, which extracts data from the database connected through `DriverManger`:

```
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DatabaseServlet extends HttpServlet
{
```

e-Macao-16-6-407

Task 96: Connecting DataBase

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException, java.io.IOException {
String sql = "select * from testdata";
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
ResultSetMetaData rsm = null;
response.setContentType("text/html");
PrintWriter out = response.getWriter( );
out.println("<html><head><title>Servlet
Database  Access</title></head><body>");
```

e-Macao-16-6-408

Task 97: Connecting DataBase

```
try{
//load the database driver
  Class.forName ("com.mysql.jdbc.Driver");
//The JDBC URL for database
  String url =
    "jdbc:mysql://127.0.0.1:3306/dbTest";
// Create the java.sql.Connection to the
// database  using DriverManager
  conn =
    DriverManager.getConnection(url,"root",
    "1234");
//Create a statement for executing some SQL
      stmt = conn.createStatement( );
```

e-Macao-16-6-409

Task 98: Connecting DataBase

```
//Execute the SQL statement
     rs = stmt.executeQuery(sql);

//Get info from the ResultSetMetaData object
  rsm = rs.getMetaData( );
// Display the data
 int colCount =  rsm.getColumnCount( );
  for (int i = 1; i <=colCount; ++i){
  out.println("<th>" + rsm.getColumnName(i) +
  "</th>");}          }
  out.println("</tr>");
  while( rs.next( )){
    out.println("<tr>");
```

e-Macao-16-6-410

Task 99: Connecting DataBase

```
        for (int i = 1;  i <=colCount; ++i)
                out.println("<td>" + rs.getString(i)
                + "</td>");
               out.println("</tr>");}
      } catch (Exception e){
         throw new ServletException(e.getMessage( ));
      } finally {
          try{
                if(stmt != null)
                  stmt.close( );
               if (conn != null)
                   conn.close( );
           } catch (SQLException sqle){ }
      }
```

e-Macao-16-6-411

Task 100: Connecting DataBase

```
            out.println("</table><br><br></body></html>");

        } //doGet
}
```

b) Modify the previous `Servlet` and make it use a `DataSource` to create a database connection. The set up for connection may look as follows:

```
Context ctx = new InitialContext();
DataSource ds=
(DataSource)ctx.lookup("java:/comp/env/jdbc/Testdb");
 conn = ds.getConnection();
```

e-Macao-16-6-412

Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

Connection pooling allows a thread to get connection from a pool and return it to the pool when the work is done.

The connection may either be a new, or already-existing connection.

e-Macao-16-6-413

Advantages

Connection pooling can greatly increase the performance of Java application, while reducing overall resource usage.

The main advantages are:

a) Reduced connection creation time - the overhead for creating connection will be avoided if connections are "recycled."

b) Simplified programming model – Only simple `JDBC` programming techniques is required.

c) Controlled resource usage – The resource is controlled by the container effectively.

e-Macao-16-6-414

Tomcat Implementation

Sun has standardized the concept of connection pooling in JDBC through the JDBC-2.0 Optional Package API.

As in previous example, Tomcat has implemented the APIs with MySQL Connector/J.

For Tomcat 5.0, install the following libraries in <Tomcat_HOME>/common/lib:

a) Jakarta-Commons DBCP 1.0

b) Jakarta-Commons Collections 2.0

c) Jakarta-Commons Pool 1.0

For Tomcat 5.5, the required libraries are located in a single JAR at <TOMCAT_HOME>/common/lib/naming-factory-dbcp.jar

e-Macao-16-6-415

Tomcat Configuration

For Tomcat, the following attributes can be added to the Resource element in the server.xml file between the </GlobalNamingResources> tag:

maxActive: Maximum number of connections in connection pool. Make sure the mysqld max_connections is large enough to handle all of the connections. A value of 0 represents "no limit".

maxIdle: Maximum number of idle connections to retain in pool. Set to -1 for no limit.

maxWait: Maximum time to wait for a connection to become available in millisecond. Set to -1 to wait indefinitely.

e-Macao-16-6-416

Connection pool leaks

While using connection pooling, a web application has to explicitly close ResultSet, Statement, and Connection or they will never being available for reuse causing a connection pool leak.

The Jakarta-Commons DBCP can be configured to prevent this problem while adding the attributes to the Resource configuration for your DBCP DataSource as follows:

```
removeAbandoned="true"
removeAbandonedTimeout="60"
```

e-Macao-16-6-417

Example: Connection Pool 1

After setting up the connection pool configuration, the server.xml file of the Tomcat server may look as follows:

```
. . .
<GlobalNamingResources>
 . . .
   <Resource  name="jdbc/Testdb"
              auth="Container"
              type="javax.sql.DataSource"
              driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/dbTest"
              username="root"
              password="1234"
```

e-Macao-16-6-418

Example: Connection Pool 2

```
            maxActive="20"
            maxIdle="10"
            maxWait="-1"
            removeAbandoned="true"
            removeAbandonedTimeout="60"
/>
```

A.3.3. Security

e-Macao-16-6-419

Horizontal Concepts Outline

1) Exceptions
   a) introduction
   b) error handling
   c) error objects
   d) logging

2) DataBase Connectivity
   a) jdbc review
   b) datasource
   c) connection pooling

1) Security
   a) introduction
   b) declarative security
   c) programmatic security
   d) secure communication

2) Internationalization
   a) introduction
   b) encoding
   c) resource bundles

3) Summary

e-Macao-16-6-420

Servlet / JSPs Security

Problem to address:

1) Authentication, Authorization and Access Control (AAA)
2) Secure Encrypted Communication

e-Macao-16-6-421

Security Features

Authentication, Authorization and Access Control

1) Declarative Security:
   a) access control configuration is separated from the `Servlet` and `JSP` code
   b) no security-related code is written
   c) static security that runtime condition can not be checked

2) Programmatic Security:
   a) flexible but need more work
   b) run-time condition such as client's credit limit can be considered

e-Macao-16-6-422

Role-Based Security 1

Role-Based Security

The `servlet` specification only specifies that roles should exist and a container must recognize them. How to assign a user to a role is not specified.

In `Tomcat`, the `<TOMCAT_HOME>/conf/tomcat-users.xml` file is used to define the mapping for a user. Its default content may look as follows:

```
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
```

e-Macao-16-6-423

Role-Based Security 2

```
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat"
          roles="tomcat"/>
  <user username="role1" password="tomcat"
          roles="role1"/>
  <user username="both" password="tomcat"
          roles="tomcat,role1"/>
  <user username="admin" password=""
          roles="admin,manager"/>
</tomcat-users>
```

e-Macao-16-6-424

Applying Role-Based Security 1

The `web.xml` file is used to applied the role-based security to certain web applications. The tag `<security-constraint>` is used as follows:

```
<web-app>
...
 <security-constraint>
      <web-resource-collection>
            <web-resource-name>
                  SecuredWebPage
            </web-resource-name>
            <url-pattern>/secured/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
```

e-Macao-16-6-425

Applying Role-Based Security 2

```
            </web-resource-collection>
            <auth-constraint>
                  <role-name>role1</role-name>
            </auth-constraint>
      </security-constraint>
      <login-config>
            <auth-method>BASIC</auth-method>
      </login-config>
...
```

e-Macao-16-6-426

Task 101: Role-Based Security

1) Follow the previous example to test the role-based security feature of Tomcat.
   a) Create a secured directory under your Web application context.
   b) Use the default users setting in Tomcat's tomcat-users.xml file.
   c) Put two web pages under the secured directory.
   d) Try to access one of the secured web pages.
   e) What will happen when a wrong user name or password is received?
   f) Try to access the secured web page with correct user name and password (user name: role1, password: tomcat).
   g) Can all web pages under the secured directory be accessed?

e-Macao-16-6-427

Authentication 1

HTTP supports two built-in authentication schemes:

1) basic
   a) user name and password are essentially sent as plain text
   b) password could be spoofed by a malicious server
   c) once authentication is issued, the client will have authentication for a given subset of server resources
   d) only used over an encrypted and with strong server authentication link

e-Macao-16-6-428

Authentication 2

2) digest
   a) Introduced in HTTP 1.1 to improve the basic authentication.
   b) Not the password but an encrypted digest of the password is sent and it cannot be determined by sniffing the network.
   c) Most but not all browser support.
   d) Access may be gained by just working with the digest of the password.
   e) Note: using SSL is still a better choice for securing important content.

e-Macao-16-6-429

Form-Based Authentication 1

Custom design authentication form can be used for authentication.
Modification of the `web.xml` file is needed as follows:

```
<web-app>
. . .
  <security-constraint>
. . .
  </security-constraint>
 <login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
```

e-Macao-16-6-430

Form-Based Authentication 2

```
        <form-login-page>/login.html</form-login-page>
       <form-error-page>/loginError.jsp</form-error-page>
    </form-login-config>
 </login-config>
. . .
</web-app>
```

e-Macao-16-6-431

Form-Based Authentication 3

The login form may look as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
 Transitional//EN">
<html>
<head>
    <title>Login Form</title>
</head>
<body bgcolor="#ffffff">
<h2>Please Login to the Application</h2>
<!-- The value of action is mandatory -->
<form method="POST"  action="j_security_check">
```

e-Macao-16-6-432

Form-Based Authentication 4

```
<table border="0"><tr>
<td>Enter the username: </td><td>
<!-- The value of the text name is mandatory -->
<input type="text" name="j_username" size="15">
</td>
</tr>
<tr>
<td>Enter the password: </td><td>
<!-- The value of the password name is mandatory -->
<input type="password" name="j_password" size="15">
```

e-Macao-16-6-433

Form-Based Authentication 5

```
</td>
</tr>
<tr>
<td> <input type="submit" value="Submit"> </td>
</tr>
</table>
</form>
</body>
</html>
```

e-Macao-16-6-434

Form-Based Authentication 6

The `loginError.jsp` file may look as follows:

```
<html>
<head>
    <title>Login Error</title>
</head>
 <body bgcolor="#ffffff">
 <h2>Authentication Fail</h2><br>
 . . .
 </body>
</html>
```

e-Macao-16-6-435

Form-Based Authentication 7

Once the user is authorized, the container will maintain the login session with a cookie containing the session-id and send it back to the user for subsequent requests.

If the role of the user is not allowed for certain resources, a "`403 Access Denied`" response will be received by the user.

Note:

a) still not a strong authentication
b) session tracking and `URL` redirecting is difficult.
c) cookie must be enabled

e-Macao-16-6-436

Programmatic Security

Problems with role-based security:

Role-based security cannot deal with runtime based checking such as the user's credit limit.

It cannot filter resources by the role of the user.

`HttpServletRequest` object provides methods to perform different logics based on the runtime information about the user.

e-Macao-16-6-437

HttpServletRquest 1

The following methods are available from the `HttpServletRequest` object for security checking purpose:

`String getAuthType()`:
returns the name of the authentication scheme for determining how form information was submitted

`boolean isUserInRole(java.lang.String role)`:

To check if a user is in the given role.

`String getProtocol()`:

returns the protocol that was used to send the request for checking if a secure protocol was used

e-Macao-16-6-438

HttpServletRquest 2

`boolean isSecure()`:

a boolean value representing if a `HTTPS` request was made.

`Principle getUserPrinciple()`:

returns a `java.security.Principle` object that contains the name of the current authenticated user.

`String getRemoteUser()`:

If the user is not authenticated, `null` will be return.

e-Macao-16-6-439

Example: Programmatic Security 1

1) The following `Servlet` checks the user's role and generates different content according to the role.

```
if (request.isUserInRole("manager")) {
        out.println("<B>Hello Manager");
            out.println (request.getRemoteUser());
            out.println ("</B></br>");
 }
   else if (request.isUserInRole("role1")) {
        out.println("<B>Hello User");
        out.println (request.getRemoteUser());
        out.println ("</B></br>");
    }
```

e-Macao-16-6-440

Example: Programmatic Security 2

```
else {
       throw new IOException("User does not have
       access!");
 }
```

e-Macao-16-6-441

Task 102: Programmatic Security

1) After logged in through a form-based authentication, access right will last within the same session. Try to access another secured page under the same context.
2) Try to delete the cookie stored in the browser from a sender "`localhost`". Browse to the same secured page again. What happen?
3) Create a page to allow the user to logout. The page should perform the follows:
   a) Check if the user was authenticated.
   b) Find out if the user were under a specific role.
   c) Logout the user by invalidate the session.
4) The skeleton code may look as follows:

e-Macao-16-6-442

Task 103: Programmatic Security

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class LogoutServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
   HttpServletResponse response)
    throws ServletException, java.io.IOException {
   HttpSession session = request.getSession( );
   response.setContentType("text/html");
   PrintWriter out = response.getWriter( );
   out.println("<html><head><title>Logout Authenticated
      User</title></head><body>");
```

e-Macao-16-6-443

Task 104: Programmatic Security

```
out.println("request.getRemoteUser( ) returns: ");
 //get the logged-in user's name
  String userName = request.getRemoteUser( );
 //If request.getRemoteUser( ) return null then the
  //user is not authenticated
  out.println(userName == null ? "Not authenticated." :
 userName + "<br>" );
  out.println("request.isUserInRole(\"admin\")  returns:
 ");
 //Find out whether the user is in the admin role
  out.println(isInRole + + "<br>" );
```

e-Macao-16-6-444

Task 105: Programmatic Security

```
   //log out the user by invalidating the HttpSession
     session.invalidate( );
     out.println("</body></html>");
  }
  . . .
}
```

e-Macao-16-6-445

Secured Communication

Other than controlling the access to certain resources, encrypting the transmitted data to provide secured communication is equally important.

The level of security can be configured with the `web.xml` file by the `<transport-guarantee>` element within the tag `<user-data-constraint>`.

The `<transport-guarantee>` element has three levels of security:

`NONE` – default and requires no security

`INTEGRAL` – container must ensure the integrity of information

`CONFIDENTIAL` – information sent must be both private and unchanged

e-Macao-16-6-446

Security Configuration: Tomcat 1

`Tomcat` needs specific configuration to provide secured communication.

a) In `<TOMCAT_HOME>/conf/server.xml`, find the following entry and modify the redirect attribute to "443", the default port `HTTPS` :

```
<Connector
  port="80" maxThreads="150"
   minSpareThreads="25"
   maxSpareThreads="75"
   enableLookups="false"
   redirectPort="443"
   acceptCount="100"
   connectionTimeout="20000"
   disableUploadTimeout="true"
/>
```

e-Macao-16-6-447

Security Configuration: Tomcat 2

b) In `<TOMCAT_HOME>/conf/server.xml`, uncomment the following entry and add a keypass attribute representing the password used for the keystore:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector port="443"
     maxThreads="150" minSpareThreads="25"
     maxSpareThreads="75" enableLookups="true"
     disableUploadTimeout="true"
     acceptCount="100" scheme="https" secure="true"
     clientAuth="false" sslProtocol="TLS"
     keypass="123456"
/>
```

e-Macao-16-6-448

Tomcat Configuration 3

c) Generate a self certified keystore:

```
%JAVA_HOME%/bin/keytool  -genkey -keystore
  mystore.keystore -alias tomcat -keyalg RSA
```

d) Put the keystore file generated to the home directory of `Tomcat`.

e-Macao-16-6-449

Task 106: Using HTTPS

1) Test the secure communication function provided by `Tomcat` server.
   a) Follow previous slides to configure your `Tomcat` and restart it.
   b) Modify the `web.xml` file to add the `<transport-guarantee>` element for a protected resource. Put a value "`CONFIDENTIAL`" for this element.
   c) Access the protected resource. Is there any changes to the protocol used?

A.3.4. Internationalization

e-Macao-16-6-450

Horizontal Concepts Outline

1) Exceptions
   a) introduction
   b) error handling
   c) error objects
   d) logging

2) DataBase Connectivity
   a) jdbc review
   b) datasource
   c) connection pooling

1) Security
   a) introduction
   b) declarative security
   c) programmatic security
   d) secure communication

2) Internationalization
   a) introduction
   b) encoding
   c) resource bundles

3) Summary

e-Macao-16-6-451

Introduction: Internationalization 1

Internationalization is also known as i18n representing the process of designing an application supporting multi-lingual without engineering changes.

e-Macao-16-6-452

Introduction: Internationalization 2

An internationalized program has the following characteristics:

1) With the addition of localization data, the same executable can run worldwide.
2) Textual elements, such as status messages and the `GUI` component labels, are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically.
3) Support for new languages does not require recompilation.
4) Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
5) It can be localized quickly.

e-Macao-16-6-453

Problems with Encoding

When designing a web application, character encoding is a major problem for internationalization:

1) The default character encoding of `HTTP` is `ISO-8859-1(Latin-1).`
2) `ISO-8859-1` uses only 8 bits and cannot be extended easily.

Java uses Unicode as default character encoding.

`UTF-8` is a common way to use Unicode which encoding Unicode characters using a varying number of bytes depending on the character set.

e-Macao-16-6-454

Clients' Encoding

When invoking a method such as `getParameter()` to obtain data from client, sometimes the returned `String` may not be encoded properly. The following code snippet can avoid this situation:

```
String value = request.getParameter("param");
value = new String(value.getBytes(),
   request.getCharacterEncoding());
```

e-Macao-16-6-455

Specifying Encoding

While sending information to client, the encoding can be specified by manipulating the `content-type` header :

```
response.setContentType("text/html; charset=UTF-8");
ServletOutputStream sos = response.getOutputStream();
PrintWriter out =
new PrintWriter(new OutputStreamWriter(sos,"UTF-8"),
true);
response.setLocale("","");
out.println("<html>");
```

By substituting the `UTF-8` with specific encoding, different encoding can be specified.

e-Macao-16-6-456

i18n Implementation 1

Different ways can be done to provide multi-lingual support for a web site.

The following example illustrates one of the ways which uses a mechanism called resource bundle to provide i18n support.

Assume a simple web page, `welcome.html`, as follows:

```
<html>
<head>
<title>Hello!</title>
</head>
```

e-Macao-16-6-457

i18n Implementation 2

```
<body>
   <cr>Welcome to the multi-language page<br/>
   <i>multi-language page</i></cr>
</body>
</html>
```

e-Macao-16-6-458

Resource Bundle Files

In order to provide multi-lingual support, resource bundles can be used to store the content information in different languages.

The resource bundle files for various languages may look like as follows:

For English:

```
title=Welcome!
welcome=<b>Welcome</b>
```

For Chinese:

```
title=    !
welcome=<b>    </b>
```

e-Macao-16-6-459

Naming of Resource Bundles

Each resource bundle file is just a simple property text file containing key/value pairs information.

Each resource bundle file has a name starting with the base name, appending with "_" and a two-digit language code. An extension ".properties" should be used for this file.

For example, if the base name for resource bundle is "resource", the locale-specific property file will then be "`resource_en.properties`" for English client. The name can also be extend with country code like `_zh_TW` and `_zh_CN` representing Taiwan and China respectively.

The resource bundle files should be placed under the WEB-INF/classes directory or any sub-directory of it.

e-Macao-16-6-460

ResourceBundle Object

`java.util.ResourceBundle` provides static methods which takes base name and Locale object to return a resource bundle with proper values.

For example:

```
Locale locale = request.getLocale();
ResourceBundle rb = ResourceBundle.getBundle("resource",
 locale);
```

This code will check the `Locale` of the client and if, for example, it were `zh_TW`, the values of the `resource_zh_TW.properties` file would be loaded.

e-Macao-16-6-461

List of Country Code

The following link lists the country code used for the Resource Bundle file:

`http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html`

e-Macao-16-6-462

Loading Resource Bundles 1

A JavaBean may be used as a façade for loading the content of the resource bundle object. `Getter` and `Setter` methods should be defined in this JavaBean.

For example:

```
package com.web;
public class Welcome {
protected String title = null;
protected String welcome = null;
public String getTitle() {
        return title;
}
```

e-Macao-16-6-463

Loading Resource Bundles 2

```
   public void setTitle(String title) {
      this.title = title;
   }
   public String getWelcome() {
      return welcome;
   }
   public void setWelcome(String welcome) {
      this.welcome = welcome;
   }
}
```

e-Macao-16-6-464

Using Resource Bundles 1

In JSP file, scriptlets may be used to extract the values from the resource bundle as follows:

```
<%@ page import="java.util, com.web.Resource"%>
<%
Locale locale = request.getLocale();
ResourceBundle rb = ResourceBundle.getBundle("resource",
locale);
Welcome content = new Welcome();
content.setTitle(rb.getString("title"));
content.setWelcome(rb.getString("welcome"));
request.setAttribute("content", content);
%>
```

e-Macao-16-6-465

Using Resource Bundles 2

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt"
prefix="c" %>
<html>
<head>
<title>${content.title}</title>
</head>
<body>
${content.welcome}
</body>
</html>
```

e-Macao-16-6-466

JSTL i18n Tags 1

`JSTL` provides a set of `i18n` tags for supporting internationalization.

The following example illustrates how to use the tag message:

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<fmt:bundle basename="resource">
```

e-Macao-16-6-467

JSTL i18n Tags 2

```
<html>
<head>
<title><fmt:message key="title"/></title>
</head>
<body>
<fmt:message key="welcome"/>
</body>
</html>
</fmt:bundle>
```

e-Macao-16-6-468

Task 107: i18n

1) Create three `ResourceBundle` files for a web page: one for English, one for Traditional Chinese and one for Simplified Chinese. A `JSP` page with a form for selecting different languages is used to display the content with different languages. The output may look as follows:

e-Macao-16-6-469

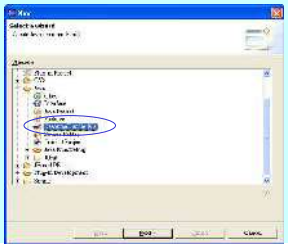
Task 108: i18n

e-Macao-16-6-470

Task 109: i18n

a) Tools for creating the `ResourceBundle` files can be used. There is a free plugin, named `Jinto`, for `Eclipse` which converts input into unicode and generates `ResourceBundle` files. This plugin can be downloaded at :
`http://www.guh-software.de/jinto_en.html`

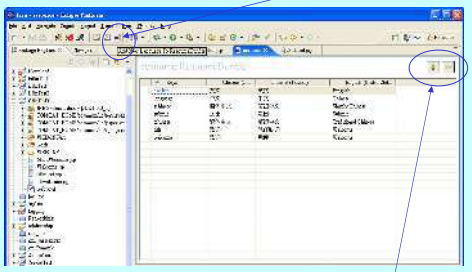
b) After installed this plugin, create a new resource bundle file at `Eclipse: File` → `New` → `Others` → `Java` → `ResourceBundle File`



e-Macao-16-6-471

Task 110: i18n

c) For adding a new language, just press this **button**



d) For adding or deleting an entry, press these **buttons**.

e-Macao-16-6-472

Task 111: i18n

d) While using JSTL tags for the JSP file, the JSP file may look like the following in order to produce the required effect:

```
<%@ page contentType="text/html" pageEncoding="UTF-
   8"%>
<%@ taglib  uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
<%@ taglib  uri="http://java.sun.com/jsp/jstl/fmt"
   prefix="fmt" %>

<c:if test="${param.language == 'en'}">
  <fmt:setLocale value="en" />
</c:if>
```

e-Macao-16-6-473

Task 112: i18n

```
<c:if test="${param.language == 'zh_TW'}">
  <fmt:setLocale value="zh_TW" />
</c:if>
<c:if test="${param.language == 'zh_CN'}">
  <fmt:setLocale value="zh_CN" />
</c:if>
<fmt:bundle basename="resource" >
<fmt:setBundle basename="resource" var="currLang"/>
<html>
  <head>
    <title>
      <fmt:message key="title" />
    </title>
```

e-Macao-16-6-474

Task 113: i18n

```
</head>
  <body bgcolor="white">
    <h1>
      <fmt:message key="welcome" />
    </h1>

    <form action="jstlselect.jsp">
      <p>
      <input type="radio" name="language" value="en"
        ${currLang == 'en' ? 'checked' : ''}>
      <fmt:message key="english" /><br>
      <input type="radio" name="language"
   value="zh_TW"
        ${currLang == 'zh_TW' ? 'checked' : ''}>
```

e-Macao-16-6-475

Task 114: i18n

```
<fmt:message key="tchinese" /><br>
      <input type="radio" name="language"
   value="zh_CN"
        ${currLang == 'zh_CN' ? 'checked' : ''}>
      <fmt:message key="schinese" /><br>
      <p>
      <input type="submit" value="<fmt:message
   key="submit" />" >
      </form>
  </body>
</html>
</fmt:bundle>
```

A.3.5. Summary

e-Macao-16-6-476

Horizontal Concepts Outline

1) Exceptions
   a) introduction
   b) error handling
   c) error objects
   d) logging
2) DataBase Connectivity
   a) jdbc review
   b) datasource
   c) connection pooling
1) Security
   a) introduction
   b) declarative security
   c) programmatic security
   d) secure communication
2) Internationalization
   a) introduction
   b) encoding
   c) resource bundles
3) Summary

e-Macao-16-6-477

Summary

1) In this section, the following supporting technologies are presented:
   a) Exception handling
   b) Database Connectivity
   c) Internationalization
   d) Security

e-Macao-16-6-478

Summary: Exception Handling 1

1) Directive `<%@page errorPage="myErrorPage.jsp"%>` indicates a page for handling the exception.

2) Directive `<%@ page isErrorPage="true"%>` indicateds that the current page can handle exception.

3) `Servlet` can also be used for handling exception.

4) A specific page can be declared within the `web.xml` file to handle specific error

e-Macao-16-6-479

Summary: Exception Handling 2

1) Error Objects are defined by the Servlet specification to provide useful information for debugging such as:
   a) status code
   b) exception type
   c) exception
   d) request uri

e-Macao-16-6-480

Summary: Exception Handling 2

Logging is used to keep record of important information.

The `java.util.logging` package provides a standard API for logging.

The following logging handlers are defined for handling different logged information:

a) `SteramHandler`
b) `MemoryHandler`
c) `SocketHandler`
d) `FileHandler`

e-Macao-16-6-481

Summary: Database Connectivity

`DataSource` is used for connecting database with high efficency.

`DataSource` is managed by container but needed to be configured for different server.

Connection Pooling create and manage a pool of connections and can be accessed and managed easily through `DataSource`.

e-Macao-16-6-482

Summary: Security

The security of a web site can be enforced through role-based or programmatic authentication.

Role-Based security can be set up easily in Tomcat server but is lack of flexibility.

Programmatic security can provide runtime checking and provide a more flexible access control.

Programmatic involves more work.

e-Macao-16-6-483

Summary: Internationalization

Internationalization is also known as i18n and aim to provide multi-lingual support for a web site.

One of the ways to provide multi-lingual web site is through the usage of `ResouceBundle` file.

A.4. Case Study

Case Study

e-Macao-16-6-485

Course Outline

1) J2EE introduction
2) vertical concepts
   a) Servlets
   b) JavaServer Pages
   c) Filters
3) horizontal concepts
   a) exceptions
   b) security
   c) internationalization
   d) database connectivity
4) case study

e-Macao-16-6-486

Case Study Outline

1) hands-on practice

e-Macao-16-6-487

Task 115: Hands-On Practice

1) Develop a portion of a web site to review the technology and techniques discussed in this course.
   a) Create a multi-lingual supported web site which can detect the locale of the client's browser to provide corresponding contents.
   b) Use `DataSource` to connect to a database and make use of the default connection pooling.
   c) `Filter` is used to provide a hit counter of the web site.

e-Macao-16-6-488

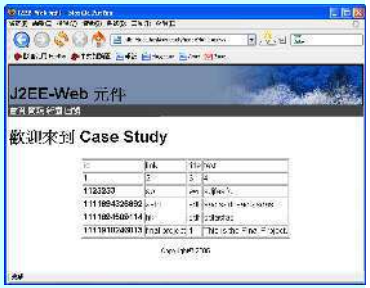
Task 116: Hands-On Practice

d) "Page not found" (404) exception should be handled by the container to redirect the client to the default web page of the site.

e) Basic security should be set up to protected the content of a specific directory.

e-Macao-16-6-489

Task 117: Hands-On Practice



e-Macao-16-6-490

Task 118: Hands-On Practice

B. Assessment

B.1. Set 1

1. (8%)	In a JSP page you are required to insert a JSP fragment called insert.jsp, where this JSP fragment requires an additional request parameter called title. What is necessary to perform this insertion?	
	A.	<%@ include file='insert.jsp' title='Web Wonk'%>
	B.	<jsp:include page="insert,jsp' title="Web Wonk"/>
	C.	<%@ include file='insert.jsp' %>Web Wonk<%@include%>
	D.	<jsp:include page='insert.jsp'> <jsp:param name='title' value='Web Wonk'/> </jsp:include>
Answer	D	

2. (10%)	Which deployment description snippet would you use to declare the use of a tag library?	
	A.	<taglib> <uri>http://jsp_prj.com/taglib.tld</uri> <location>/WEB-INF/taglib.tld</location> </taglib>
	B.	<taglib> <taglib-uri>http:// jsp_prj.com/tablib.tld</taglib-uri> <taglib-location>/WEB-INF/tablib.tld</taglib-location> </taglib>
	C.	<tag-lib> <uri>http:// jsp_prj.com/taglib.tld</uri> <location>/WEB-INF/taglib.tld</location> </tag-lib>
	D.	<tag-lib> <taglib-uri>http://jsp_prj.com /taglib.tld </taglib-uri> <taglib-location>/WEB-INF/taglib.tld</taglib-location> </tag-lib>
Answer	B	

3. (8%)	Where is the following declared JavaBean accessible? <jsp:useBean id=”ABean” class=”com.examples.ABean”/>	
	A.	Throughout the remainder of the JSP page.
	B.	Within other servlets or JSP pages in the same Web application.
	C.	Within other servlets or JSP pages in the same servlet context.
	D.	Throughout all future invocations of the JSP page, until the session expires.
Answer	A	

4. (10%)	Exhibit:
	1. public class ABean {
	2. private int count;
	3. public void setCount(int count) {
	4. this.count = count;
	5. }
	6. public int getCount() {
	7. return count;
	8. }
	9. }
	Given:
	1. <html>
	2. <body>
	3. <jsp:useBean id=”myBean” class=”ABean”>
	4.
	5. </jsp:useBean>
	6. </body>
	7. </html>
	Which of the following answer inserted individually at line 4, will initialize the count property of the newly created ABean myBean?	

	A.	<% myBean.count =1; %>
	B.	<% ABean.count=1; %>
	C.	<jsp:setProperty name=”myBean” property=”count” value=”1” />
	D.	<jsp:init property=”count” value=”1” />
Answer	C	

5. (8%)	Given servlet A:
	1. public class A extends HttpServlet {
	2. public void doPost(HttpServletRequest req,
	HttpServletResponse resp)
	throws ServletException {
	3. String id = "aString";
	4.
	5. }
	6. }
	Servlet A and servlet B share the same active session.
	Which, inserted at line 4, will allow servlet B to access the value "aString" in subsequent POST requests to servlet B?	

	A.	req.getSession().put("ID",id);
	B.	req.getSession().setValue("ID"),id)
	C.	req.getSession().putAttribute("ID",id);
	D.	req.getSession().setAttribute("ID",id);
Answer	D	

6. (8%)	Which method in the HttpServlet class services the HTTP POST request?	
	A.	doPost(ServletRequest, ServletResponse)
	B.	doPOST(ServletRequest, ServletResponse)
	C.	servicePost(HttpServletRequest, HttpServletResponse)
	D.	doPost(HttpServletRequest, HttpServletResponse)
Answer	D	

7. (8%)	Which method is required for using the URL rewriting mechanism of implementing session support?	
	A.	HttpServletRequest.encodeURL()
	B.	HttpServletRequest.rewriteURL()
	C.	HttpServletResponse.encodeURL()
	D.	HttpServletResponse.rewriteURL()
Answer	C	

8. (8%)	Which of the following implicit objects can be used to store attributes that need to be accessed from all the sessions of a web application?	
	A.	application
	B.	session
	C.	request
	D.	page
Answer	A	

9. (8%)	Select the correct statement about the following code.
	<%@ page language="java" %>
	<html><body>
	out.print("Hello ");
	out.print("World ");
	</body></html>	

	A.	It will print Hello World in a single line
	B.	It will generate compile-time errors.
	C.	It will only print Hello in one line and world in another line.
	D.	None of above.
Answer	D	

10. (8%)	Which of the following lines would you use to include the output of DataServlet into any other servlet?	
	A.	RequestDispatcher rd = request.getRequestDispatcher("/servlet/DataServlet"); rd.include(response);
	B.	RequestDispatcher rd = request.getRequestDispatcher(); rd.include("/servlet/DataServlet", request, response);
	C.	RequestDispatcher rd = request.getRequestDispatcher(); rd.include("/servlet/DataServlet", response);
	D.	RequestDispatcher rd = request.getRequestDispatcher("/servlet/DataServlet"); rd.include(request, response);
Answer	D	

11. (8%)	Which of the following code types cannot be used within a scriptlet tag?	
	A.	if block
	B.	while block
	C.	Code block
	D.	Static block
Answer	D	

12. (8%)	For a FilterChain object, chain, which of the following method can be called to invoke another Filter?	
	A.	chain.next();
	B.	chain.doNext();
	C.	chain.doFilter();
	D.	None of the above.
Answer	C	

B.2. Set 2

1. (8%)	Which method is required for using the URL rewriting mechanism of implementing session support?	
	A.	HttpServletRequest.encodeURL()
	B.	HttpServletRequest.rewriteURL()
	C.	HttpServletResponse.encodeURL()
	D.	HttpServletResponse.rewriteURL()
Answer	C	

2. (8%)	Where is the following declared JavaBean accessible? <jsp:useBean id=”ABean” class=”com.examples.ABean”/>	
	A.	Throughout the remainder of the JSP page.
	B.	Within other servlets or JSP pages in the same Web application.
	C.	Within other servlets or JSP pages in the same servlet context.
	D.	Throughout all future invocations of the JSP page, until the session expires.
Answer	A	

3. (8%)	Given servlet A:
	1. public class A extends HttpServlet {
	2. public void doPost(HttpServletRequest req,
	HttpServletResponse resp)
	throws ServletException {
	3. String id = “aString”;
	4.
	5. }
	6. }
	Servlet A and servlet B share the same active session.
	Which, inserted at line 4, will allow servlet B to access the value “aString” in subsequent POST requests to servlet B?	

	A.	req.getSession().put(“ID”,id);
	B.	req.getSession().setValue(“ID”),id)
	C.	req.getSession().putAttribute(“ID”,id);
	D.	req.getSession().setAttribute(“ID”,id);
Answer	D	

4. (8%)	For a FilterChain object, chain, which of the following method can be called to invoke another Filter?	
	A.	chain.next();
	B.	chain.doNext();
	C.	chain.doFilter();
	D.	None of the above.
Answer	C	

5. (8%)	Which of the following implicit objects can be used to store attributes that need to be accessed from all the sessions of a web application?	
	A.	application
	B.	session
	C.	request
	D.	page
Answer	A	

6. (8%)	In a JSP page you are required to insert a JSP fragment called insert.jsp, where this JSP fragment requires an additional request parameter called title. What is necessary to perform this insertion?	
	A.	<%@ include file='insert.jsp' title='Web Wonk'%>
	B.	<jsp:include page="insert,jsp' title="Web Wonk"/>
	C.	<%@ include file='insert.jsp' %>Web Wonk<%@include%>
	D.	<jsp:include page='insert.jsp'> <jsp:param name='title' value='Web Wonk'/> </jsp:include>
Answer	D	

7. (8%)	Which two represent valid JSP expressions?	
	A.	<%= Match.random() %>
	B.	<% x %>
	C.	<% int x = “4” + “2”; %>
	D.	<% String x = “4” + “2” %>
Answer	A	

8. (8%)	Select the correct statement about the following code.
	<%@ page language="java" %>
	<html><body>
	out.print("Hello ");
	out.print("World ");
	</body></html>	

	A.	It will print Hello World in a single line
	B.	It will generate compile-time errors.
	C.	It will only print Hello in one line and world in another line.
	D.	None of above.
Answer	D	

9. (10%)	Which deployment description snippet would you use to declare the use of a tag library?	
	A.	<taglib> <uri>http://jsp_prj.com/taglib.tld</uri> <location>/WEB-INF/taglib.tld</location> </taglib>
	B.	<taglib> <taglib-uri>http:// jsp_prj.com/tablib.tld</taglib-uri> <taglib-location>/WEB-INF/tablib.tld</taglib-location> </taglib>
	C.	<tag-lib> <uri>http:// jsp_prj.com/taglib.tld</uri> <location>/WEB-INF/taglib.tld</location> </tag-lib>
	D.	<tag-lib> <taglib-uri>http://jsp_prj.com /taglib.tld </taglib-uri> <taglib-location>/WEB-INF/taglib.tld</taglib-location> </tag-lib>
Answer	B	

10. (10%)	Exhibit:
	1. public class ABean {
	2. private int count;
	3. public void setCount(int count) {
	4. this.count = count;
	5. }
	6. public int getCount() {
	7. return count;
	8. }
	9. }
	Given:
	1. <html>
	2. <body>
	3. <jsp:useBean id=”myBean” class=”ABean”>
	4.
	5. </jsp:useBean>
	6. </body>
	7. </html>
	Which of the following answer inserted individually at line 4, will initialize the count property of the newly created ABean myBean?	

	A.	<% myBean.count =1; %>
	B.	<% ABean.count=1; %>
	C.	<jsp:setProperty name=”myBean” property=”count” value=”1” />
	D.	<jsp:init property=”count” value=”1” />
Answer	C	

11. (8%)	Which of the following lines would you use to include the output of DataServlet into any other servlet?	
	A.	RequestDispatcher rd = request.getRequestDispatcher("/servlet/DataServlet"); rd.include(response);
	B.	RequestDispatcher rd = request.getRequestDispatcher(); rd.include("/servlet/DataServlet", request, response);
	C.	RequestDispatcher rd = request.getRequestDispatcher(); rd.include("/servlet/DataServlet", response);
	D.	RequestDispatcher rd = request.getRequestDispatcher("/servlet/DataServlet"); rd.include(request, response);
Answer	D	

12. (8%)	Given this fragment from a Web application deployment descriptor? <context-param> <param-name>user</param-name> <param-value>tessking</param-value> </context-param> From within a servlet's doPost method, which retrieves the value of the user parameter?	
	A.	getServletConfig().getAttribute("user");
	B.	getServletContext().getAttribute("user");
	C.	getServletConfig().getInitParameter("user");
	D.	getServletContext().getInitParameter("user");
Answer	D	