

Ministerio de Educación Superior, Ciencia y Tecnología (Mescyt)

Centro de Tecnología y Educación Permanente TEP

Pontificia Universidad Católica Madre y Maestra (PUCMM)

Diplomado en Programación en Lenguaje JAVA

Laboratorio 04

Módulo III

Introducción a Java Enterprise Edition 7

Parte II

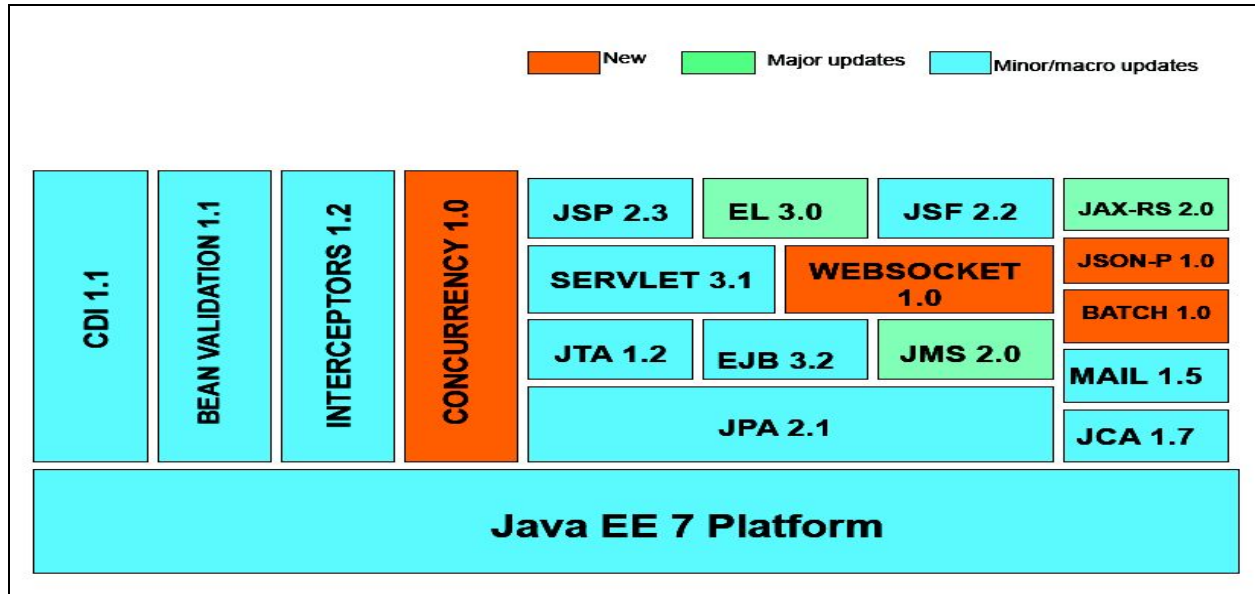
Facilitador: Ing. Eudris Cabrera

Fecha : 13 Agosto del 2016

Santiago de los Caballeros, República Dominicana

Introducción

Plataforma Java EE 7



Cómo empezar con Java EE 7 ?

Descargar e instalar:

JDK 8(ó superior):

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Instalar el IDE de su preferencia:

- NetBeans 8.0.2 superior, versión completa o “Java EE”
- <http://netbeans.org/downloads/>
- Eclipse Luna (4.4.0) o superior
- <http://www.eclipse.org/neon/>
- Otros

Guía del laboratorio

Este laboratorio está enfocado en introducir los conceptos fundamentales de **Java Enterprise Edition**. Destacando los aspectos más importantes de Java EE 7.

Java EE 7 cubre las siguientes especificaciones:

- Java Persistence API 2.1 (JSR 338)
- Java API for RESTful Web Services 2.0 (JSR 339)
- Java Message Service 2.0 (JSR 343)
- JavaServer Faces 2.2 (JSR 344)
- Contexts and Dependency Injection 1.1 (JSR 346)
- Bean Validation 1.1 (JSR 349)
- Batch Applications for the Java Platform 1.0 (JSR 352) *
- Java API for JSON Processing 1.0 (JSR 353) *
- Java API for WebSocket 1.0 (JSR 356) *
- Java Transaction API 1.2 (JSR 907)

En este laboratorio nos enfocaremos en las especificaciones **Java API for RESTful Web Services 2.0 (JSR 339)**, **JavaServer Faces 2.2 (JSR 344)** y **Java API for JSON Processing 1.0 (JSR 353)**.

Requerimientos

Java 1.7+ (Recomendado Java 8)

Netbeans 8.0.1+

Apache Maven 3.2.1+

Glassfish 4.1.1

Mysql Server 5.0+

Introducción a los servicios web

Wikipedia:

Un servicio web (en inglés, Web service) es una pieza de software que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet.

La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web.

Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Enlaces:

https://es.wikipedia.org/wiki/OASIS_%28organizaci%C3%B3n%29

https://es.wikipedia.org/wiki/World_Wide_Web_Consortium

Estándares empleados

Web Services Protocol Stack: Así se denomina al conjunto de servicios y protocolos de los servicios Web.

XML (Extensible Markup Language): Es el formato estándar para los datos que se vayan a intercambiar.

SOAP (Simple Object Access Protocol) o XML-RPC (XML Remote Procedure Call):

Protocolos sobre los que se establece el intercambio.

Otros protocolos: los datos en XML también pueden enviarse de una aplicación a otra mediante protocolos normales como HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), o SMTP (Simple Mail Transfer Protocol).

WSDL (Web Services Description Language): Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.

UDDI (Universal Description, Discovery and Integration): Protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.

WS-Security (Web Service Security): Protocolo de seguridad aceptado como estándar por OASIS (Organization for the Advancement of Structured Information Standards). Garantiza la autenticación de los actores y la confidencialidad de los mensajes enviados.

REST (Representational State Transfer): arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, POST, PUT, DELETE, etc) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente.

Inconvenientes de los servicios Web

Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como **CORBA (Common Object Request Broker Architecture)**.

Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como **RMI (Remote Method Invocation)**, **CORBA** o **DCOM** (Distributed Component Object Model).

Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

Ventajas de los servicios web

Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.

Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Razones para crear servicios Web

La principal razón para usar servicios Web es que se pueden utilizar con HTTP sobre TCP (Transmission Control Protocol) en el puerto 80.

Servicios Web REST

Roy Thomas Fielding en su tesis doctoral, Estilos Arquitecturales y el Diseño de Arquitecturas de Software basadas en Red, describe Representational State Transfer (REST) como un enfoque para desarrollar servicios web y una alternativa a otras especificaciones de computación distribuida tales como CORBA o DCOM.

Enlaces:

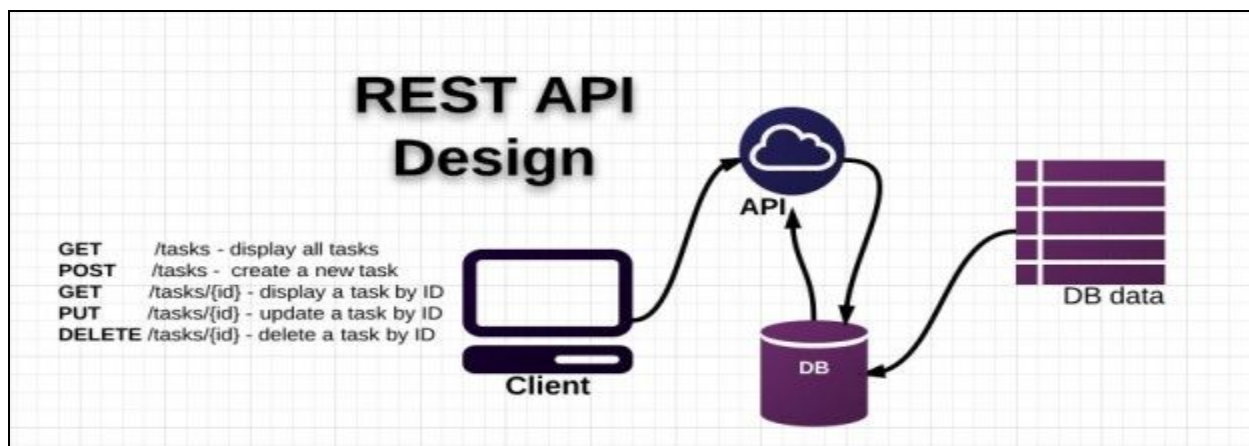
<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

https://en.wikipedia.org/wiki/Roy_Fielding

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

En resumen se trata de:

- Arquitectura de sistema distribuido sobre el protocolo HTTP.
- Protocolo cliente / servidor sin estado.
- Operaciones definidas, GET, POST, PUT y DELETE soportando un CRUD.
- Identificador único para los recursos (URI).
- Estándar Flexible
- Servicio web sobre HTTP representado con JSON.



JSON (JavaScript Object Notation):

Es un formato ligero para el intercambio de datos.

Su simplicidad ha dado lugar a la generalización de su uso, especialmente como alternativa a XML como representación de objetos y es el estándar de facto para servicios REST.

```
2
3 Ejemplo de una representación JSON
4
5 { "campo": "valor",
6   "hijos": [
7     {"id": "1", "valor": "1"},
8     {"id": "2", "valor": "3"},
9
10    ]
11 }
```

JAX-RS

- API de Java para crear servicios web REST
- JAX-RS 1.0 fue liberada en el 2008 para la versión Java EE 6 bajo la especificación JSR-311
- Simplifica el proceso de creación de servicios Web mediante Plain Old Java Objects (POJOs) y anotaciones.
- Mapea las peticiones HTTP con invocaciones de método Java
- Incluye anotaciones para implementar servicios Web: **@Path**, **@Get**, **@Put**, **@Post**, **@Delete**, **@PathParam**, **@QueryParam**, **@Produces**, **@Consumes**, entre otros.

Ejemplo Clase JAX-RS

```

* @author ecabrerar
*/
@Path("/flights")
public class FlightRestService {

    @Inject
    FlightService flightService;

    @POST
    @Consumes({MediaType.APPLICATION_JSON})
    public void create(Flight entity) {
        flightService.create(entity);
    }

    @PUT
    @Path("/{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Integer id, Flight entity) {...3 lines }

    @DELETE
    @Path("/{id}")
    public void remove(@PathParam("id") Integer id) {...3 lines }

    @GET
    @Path("/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Flight find(@PathParam("id") Integer id) {...3 lines }

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public List<Flight> findAll() {...3 lines }
}

```

¿Qué hay de nuevo en Java EE 7?

Java API for RESTful Web Services 2.0 (JSR 339) / JAX-RS 2.0:

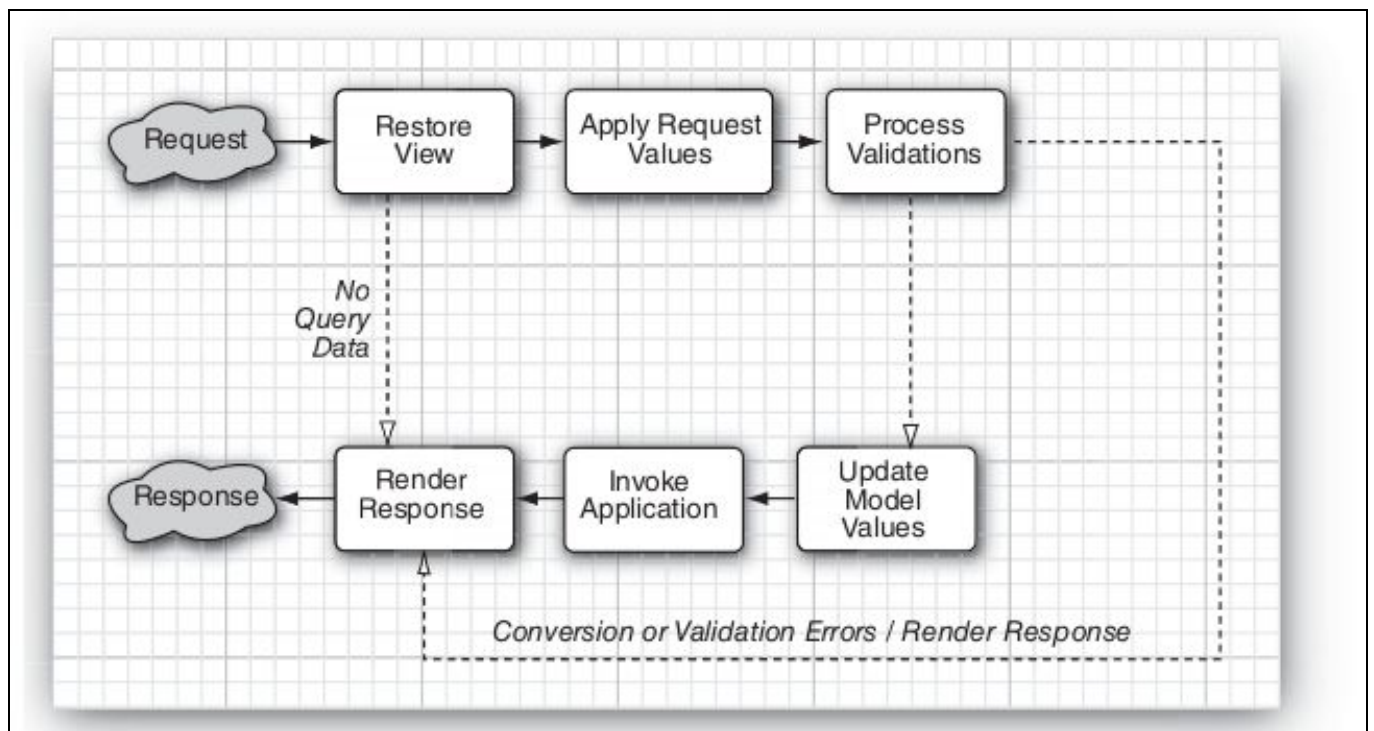
- Introduce elementos que ayudan a la productividad.
- Simplifica el API
- Incluye los siguientes aspectos:
 - API para el Cliente
 - Llamadas Asíncronas vía Http
 - Filtros e interceptores

Java Server Faces(JSF)

- Tecnología y Framework para el desarrollo de aplicaciones web en Java.
- Incluye:
 - APIs para el manejo de eventos, validar entradas, esquema de control de navegación
 - Administración de estados.
 - Basado en componentes
 - Eventos gestionados desde el servidor
- Disponible desde la versión 1.0, 2004 en Java EE

- Existen librerías tipo extensiones para el manejo de componentes visuales:
 - RichFaces
 - ICEFaces
 - PrimeFaces
- ¿Cómo trabaja JSF?
 - Utiliza taglib los cuales están asociados a clases manejadoras.
 - Todas las etiquetas son procesadas y presentada mediante HTML, mapeando cada etiqueta con su representación en el server. **Codificación**
 - Cada petición es manejada vía POST y **decodificado** los valores para ser procesadas.

- **Ciclo de vida en JSF**



- **Managed Beans**



- Representan la separación de la vista con la regla de negocio.




- Son componentes reusables.
- Facilitan el procesamiento de la información desde el formulario al servidor y viceversa.

<pre> L */ @ManagedBean(name = "miFormulario") @SessionScoped public class MiFormularioBean implements Serializable{ </pre>	Definición
<pre> @Size(max = 5) private String nombre; private String apellido; int contador; </pre>	Propiedades
<pre> public String procesarDatosForm(){ /** * Controlar cualquier cosa que sea necesario en su procesamiento... */ nombre=nombre.toUpperCase(); // return "resultado?faces-redirect=true"; } </pre>	Set & Get
<pre> public int getContador() { return contador; } </pre>	
<pre> public void setContador(int contador) { this.contador = contador; } </pre>	

- **I18n - Internacionalización**

- Permite construir aplicaciones multi-idomas de una manera limpia y práctica.
- Se apoya de archivos de propiedades para estos fines.



i18n

- ▶  **mensajes_es.properties**
- ▶  **mensajes_fr.properties**
- ▶  **mensajes_us.properties**

- **I18n - Internacionalización - Cambiado el Idioma**

```

public void cambiarIdiomaFrances(ActionEvent evento){
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new Locale("fr"));
}

public void cambiarIdiomaEspanol(ActionEvent evento){
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new
Locale("es"));
}

public void cambiarIdiomaIngles(ActionEvent evento){
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new
Locale("us"));
}

```

- Control de Navegación
 - Permite establecer la forma de como cambiar las vistas en función a los peticiones del usuario.
 - Proceso simplificado desde la versión 2.X sin necesidad de configurar en el faces-config.xml
 - Permite realizar redirect (GET, ?faces-redirect=true) y mensajes Flash.
- Control de Navegación - Ejemplo

```

//Método de navegación
public String navegacion(){
    if(valor.equalsIgnoreCase("...")){
        FacesContext.getCurrentInstance().getExternalContext().getFlash().put("mensaje", "Tomando dle flash");
        return "pagina1?faces-redirect=true";
    }else{
        return null; //En la misma pagina.
    }
}

//Recuperando llamada GET
<f:metadata>
    <f:viewParam name="matricula" value="#{ejemplos.matricula}" />
</f:metadata>

<h:link value="LLamada via get" outcome="pagina1"/>

<h:link value="LLamada via get" outcome="pagina1" includeViewParams="true">
    <f:param name="matricula" value="20011136"/>
</h:link>

```

- **Plantillas**

- Permiten reutilizar bloques de códigos en la vista.
- Simplifica el mantenimiento del sistema.
- Mecanismo similar a Apache Tiles

- **Plantillas**

- Permiten reutilizar bloques de códigos en la vista.
- Simplifica el mantenimiento del sistema.
- Mecanismo similar a Apache Tiles.
- Etiquetas utilizadas en JSF:
 - ui:insert
 - ui:define
 - ui:composition
 - ui:include
 - ui:component

- **Plantillas - Ejemplos - Template**

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <title>
      <ui:insert name="tituloHeader">Titulo por defecto...</ui:insert>
    </title>
  </h:head>
  <h:body>
    <ui:insert name="cabecera">

    </ui:insert>
    <ui:insert name="cuerpo">

    </ui:insert>
    <ui:insert name="piePagina">

    </ui:insert>
  </h:body>
</html>
```

- **Plantillas - Ejemplos - Cliente**

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">

    <ui:composition template="/WEB-INF/template/template.xhtml">

        <ui:define name="tituloHeader">
            Hola mundo desde el template...
        </ui:define>

        <ui:define name="piePagina">
            <p>
                Represento el pie de pagina.
                <ui:include src="paraIncluir.xhtml"/>
            </p>
        </ui:define>

        <ui:define name="cuerpo">
            <p>Represento el cuerpo de la pagina.</p>
        </ui:define>

        <ui:define name="cabecera">
            <p>Represento el header de pagina.</p>
        </ui:define>

    </ui:composition>

</html>

```

- **Recursos**

- Trabajar con imágenes, CSS, JS entre otros recursos, puede representar un problema a la hora de especificar las rutas relativas.
- JSF contiene un mecanismo para simplificar dicho problema al programador, la carpeta **resources**.
- Algunas etiquetas:
 - h:outputStylesheet
 - h:outputScript
 - h:graphicImage

¿Qué hay de nuevo en Java EE 7?

JavaServer Faces 2.2 (JSR 344) / JSF 2.2 :

- Pertenece a Java EE 7.
- Cambio de espaciado de nombre.
- Incluye cambios en los siguiente aspecto:

- Soporte HTML5.
- Componente **File Upload**.
- Faces Flow.
- Protección sobre Cross Site Request Forgery
- Multi-Templating.

- **JSF 2.2 - File Upload**

- Incluye la etiqueta **h:inputFile**. Debe estar dentro de un form con el **enctype** “**multipart/form-data**”
- La propiedad del Bean es del tipo `javax.servlet.http.Part`.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>JSF 2.2</title>
  </h:head>
  <h:body>
    Subir Archivos
    <h:form enctype="multipart/form-data">
      <h:inputFile value="#{Form.archivo}"/>
      <h:commandButton action="#{Form.upload()}" value="Subir Archivo"/>
    </h:form>
  </h:body>
</html>
```

```
@Named("Form")
@SessionScoped
public class MiForm implements Serializable{
    private Part archivo;

    public String upload() throws IOException {
        //Obteniendo el archivo.
        FileOutputStream fOut=new FileOutputStream("/tmp/mi_archivo");
        int read=0;
        byte[] data=new byte[1024];
        InputStream in=archivo.getInputStream();

        while((read = in.read(data)) != -1){
            fOut.write(data, 0, read);
        }

        return null;
    }

    public void setArchivo(Part archivo) {
        this.archivo = archivo;
    }

    public Part getArchivo() {
        return archivo;
    }
}
```


- **JSF 2.2 - Soporte HTML5**

- Se incluye el taglib: xmlns:pt="<http://xmlns.jcp.org/jsf/passthrough>" para traspasar propiedades.
- Se incluye la etiqueta **f:passThroughAttribute** y **f:passThroughAttributes** para combinar con las etiquetas **JSF**.
- Incluye el traspaso de elementos.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
```

```
<p>
  <label>Email:</label>
  <br/>
  <input type="email" jsf:id="email" name="email" value="#{formularioBean.email}" required="required"/>
</p>
<p>
  <label>Fecha de Llegada:</label>
  <br/>
  <input type="date" jsf:id="fecha" name="fecha" value="#{formularioBean.fecha}" required="required"/>
</p>
<p>
  <progress jsf:id="progressBar" max="100" value="0"/>
</p>
```

- **JSF 2.2 - HTML 5 - Traspaso de Propiedades**

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:pt="http://xmlns.jcp.org/jsf/passthrough"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>JSF 2.2</title>
  </h:head>
  <h:body>
    Formulario
    <h:form enctype="multipart/form-data">
      Email:
      <h:inputText value="#{Form.email}" pt:placeholder="Digite el correo" pt:type="email" />
      Matricula:
      <h:inputText value="#{Form.matricula}" >
        <f:passThroughAttribute name="placeholder" value="Digite la matricula"/>
        <f:passThroughAttribute name="type" value="number"/>
      </h:inputText> <br/>
      Nombre:
      <h:inputText value="#{Form.nombre}">
        <f:passThroughAttributes value="#{Form.atributosHtml5}"/>
      </h:inputText>
      <h:commandButton action="#{Form.upload()}" value="Procesar"/>
      <h:messages/>
    </h:form>
  </h:body>
</html>

```

- JSF 2.2 - HTML 5 - Traspaso elementos JSF

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsf="http://xmlns.jcp.org/jsf">
  <head jsf:id="head">
    <title>Ejemplo 2.2</title>
  </head>
  <body jsf:id="body">
    <form jsf:id="form">
      <input type="text" jsf:id="nombre"
            placeholder="Digite el nombre"
            jsf:value="#{Form.nombre}"/>
      <button jsf:action="#{Form.upload()}">Procesar</button>
    </form>
  </body>
</html>

```

Java API for JSON Processing 1.0 (JSR 353)

JSR 353 es el API de Java para procesamiento JSON (JSON-P) y define un API para el proceso (por ejemplo, análisis, generar, transformar y consulta) JSON.

Este JSR forma parte de Java EE 7.

El API permite producir y consumir JSON de manera secuencial (equivalente a StAX en el mundo XML) y construir un modelo de objetos de Java para JSON (equivalente a DOM en el mundo XML)

Puntos importantes del API.

Basados en DOM.

JsonBuilder - Construye un objeto JSON o un arreglo JSON

JsonReader - Lee un objeto JSON o un arreglo

JsonWriter - Escribe un objeto JSON o un arreglo

Streaming APIs

JsonGenerator

JsonParser

Sintaxis JSON

```
{
  "firstName": "Duke",
  "lastName": "Java",
  "age": 18,
  "streetAddress": "100 Internet Dr",
  "city": "JavaTown",
  "state": "JA",
  "postalCode": "12345",
  "phoneNumbers": [
    { "Mobile": "111-111-1111" },
    { "Home": "222-222-2222" }
  ]
}
```

Construir objeto JSON con JsonBuilder

```
JsonObject value = new JsonBuilder()
    .beginObject()
    .add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25)
    .beginObject("address")
    .add("streetAddress", "21 2nd Street")
    .add("city", "New York")
    .add("state", "NY")
    .add("postalCode", "10021")
    .endObject()
    .beginArray("phoneNumber")
    .beginObject()
    .add("type", "home")
    .add("number", "212 555-1234")
    .endObject()
    .beginObject()
    .add("type", "home")
    .add("number", "646 555-4567")
    .endObject()
    .endArray()
    .endObject()
    .build();
```

Ejemplo de uso de JsonReader

```
String json = "...";
JsonReader reader = new JsonReader(new StringReader(json));
JsonValue value = reader.readObject();
reader.close();
```

Ejemplo de uso de JsonWriter

```
JsonWriter jsonWriter = new JsonWriter(new FileWriter(...));
JsonObject jsonObject = new JsonBuilder()
    .beginObject()
    . . .
    .endObject()
    .build();
jsonWriter.writeObject(jsonObject);
jsonWriter.close();
```

Escribiendo en un archivo usando JsonGenerator

```
FileWriter writer = new FileWriter("test.txt");
JsonGenerator gen = Json.createGenerator(writer);
gen.writeStartObject()
    .write("firstName", "Duke")
    .write("lastName", "Java")
    .write("age", 18)
    .write("streetAddress", "100 Internet Dr")
    .write("city", "JavaTown")
    .write("state", "JA")
    .write("postalCode", "12345")
    .writeStartArray("phoneNumbers")
        .writeStartObject()
            .write("type", "mobile")
            .write("number", "111-111-1111")
        .writeEnd()
        .writeStartObject()
            .write("type", "home")
            .write("number", "222-222-2222")
        .writeEnd()
    .writeEnd()
.writeEnd();
gen.close();
```

Leyendo un archivo con formato JSON utilizando JsonParser

```
JsonParser parser = Json.createParser(new StringReader(jsonData));
while (parser.hasNext()) {
    JsonParser.Event event = parser.next();
    switch(event) {
        case START_ARRAY:
        case END_ARRAY:
        case START_OBJECT:
        case END_OBJECT:
        case VALUE_FALSE:
        case VALUE_NULL:
        case VALUE_TRUE:
            System.out.println(event.toString());
            break;
        case KEY_NAME:
            System.out.print(event.toString() + " " +
                             parser.getString() + " - ");
            break;
        case VALUE_STRING:
        case VALUE_NUMBER:
            System.out.println(event.toString() + " " +
                               parser.getString());
            break;
    }
}
```