Introducción
Java EE 7

Carlos Camacho @ccamachog
Eudris Cabrera @eudriscabrera

MAKE THE FUTURE JAVA

22 de Febrero 2014, PUCMM, Santiago de los Caballeros, R. D.

Asuntos Legales

Los conceptos y juicios de valor emitidos en esta presentación es responsabilidad personal y no se puede entenderse como una posición oficial de alguna empresa con la que hemos tenido relación laboral.

Todas las marcas registradas, así como todos los logotipos, imágenes, fotografías, audio y vídeos mostrados en esta presentación son propiedad de sus respectivos propietarios.

Su utilización es solamente para fines ilustrativos y no pretendemos dar a entender cualquier afiliación con esas empresas.

Sobre Carlos Camacho

MAKE THE FUTURE JAVA

- Ing. Telemático |Magíster Tecnología Educativa | Profesor Departamento Sistema, PUCMM.
- Más de 6 años de experiencias en desarrollo de sistemas bajo Java SE y Java EE.
- Fundación Código Libre, Santiago - Miembro Fundador | Coordinador de Sistemas, Reefer Services, S.A.S | Presidente, AvatharTech E.I.R.L.

Comunidades

- twitter: @ccamachog
- LinkedIn: @ccamachog

carlosalfredocamacho@gmail.com

Sobre Eudris Cabrera

MAKE THE FUTURE JAVA

- Desarrollador Informático / Consultor en PAFI
  (Programa de Administración Financiera Integrada / Ministerio de Hacienda).
- Desarrollador Java EE / SE, consultor y a veces entrenador en Java desde hace más de 6 años.
- Entusiasta de la tecnología y software libre.

eudris@gmail.com

Comunidades

- Github: @ecabrerar
- Google Groups: @letsrockjava
- LinkedIn: @eudriscabrera

Agenda

- Ecosistema Java
- Conceptos generales sobre JEE
- Servidores de Aplicaciones y/o contenedores JEE
- Aspectos importantes de Java EE 7
- Java EE 7 APIs
- Presentación Java Dominicano

Objetivos

- Introducir los conceptos fundamentales de Java Enterprise Edition.
- Destacar los aspectos más importantes de Java EE 7

Ecosistema Java

- Plataforma Java:
  - Multi-plataforma.
  - Utiliza una máquina virtual para su ejecución (JVM)
  - Creado por James Gosling
  - Esta dividida en:
    - Java SE
    - Java EE
    - Java ME
    - Javafx
  - El estandar es manejado por Java Community Process (JCP)

Ecosistema Java

MAKE THE FUTURE JAVA

- Popularidad del lenguaje Java por TIOBE

Feb 2014	Feb 2013	Change	Programming Language	Ratings	Change
1	2	▲	C	18.334%	+1.25%
2	1	▼	Java	17.316%	-1.07%
3	3		Objective-C	11.341%	+1.54%
4	4		C++	6.892%	-1.87%
5	5		C#	6.450%	-0.23%
6	6		PHP	4.219%	-0.85%
7	8	▲	(Visual) Basic	2.759%	-1.89%
8	7	▼	Python	2.157%	-2.79%
9	11	▲	JavaScript	1.929%	+0.51%
10	12	▲	Visual Basic .NET	1.798%	+0.79%

Febrero 2014, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

!Java donde quiera¡

MAKE THE FUTURE JAVA

- 5 trillones SIMS y Smart Cards
- 3 trillones dispositivos móviles.
- 80 millones TV, incluyendo Blu-ray, printers, maquinas bancarias, eBooks Reader y Carros

Trillones, representa el término inglés de 10^12, http://www.oracle.com/us/technologies/java/embedded/overview/index.html

Java Platform, Enterprise Edition (EE)

- Es parte de la plataforma Java.
- Útil para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java con arquitectura de N capas distribuidas.
- Se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

Diferencia entre JSE y JEE

JSE es la versión estándar de java. Es la api base del lenguaje mientras que **JEE** podríamos decir que es una versión extendida de **JSE** ya que añade a la versión estándar nuevas clases e interfaces para realizar aplicaciones web y cliente/servidor.

Lo que permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez integrable con tecnologías anteriores.

Java EE

Las razones que empujan a la creación de la plataforma JEE:

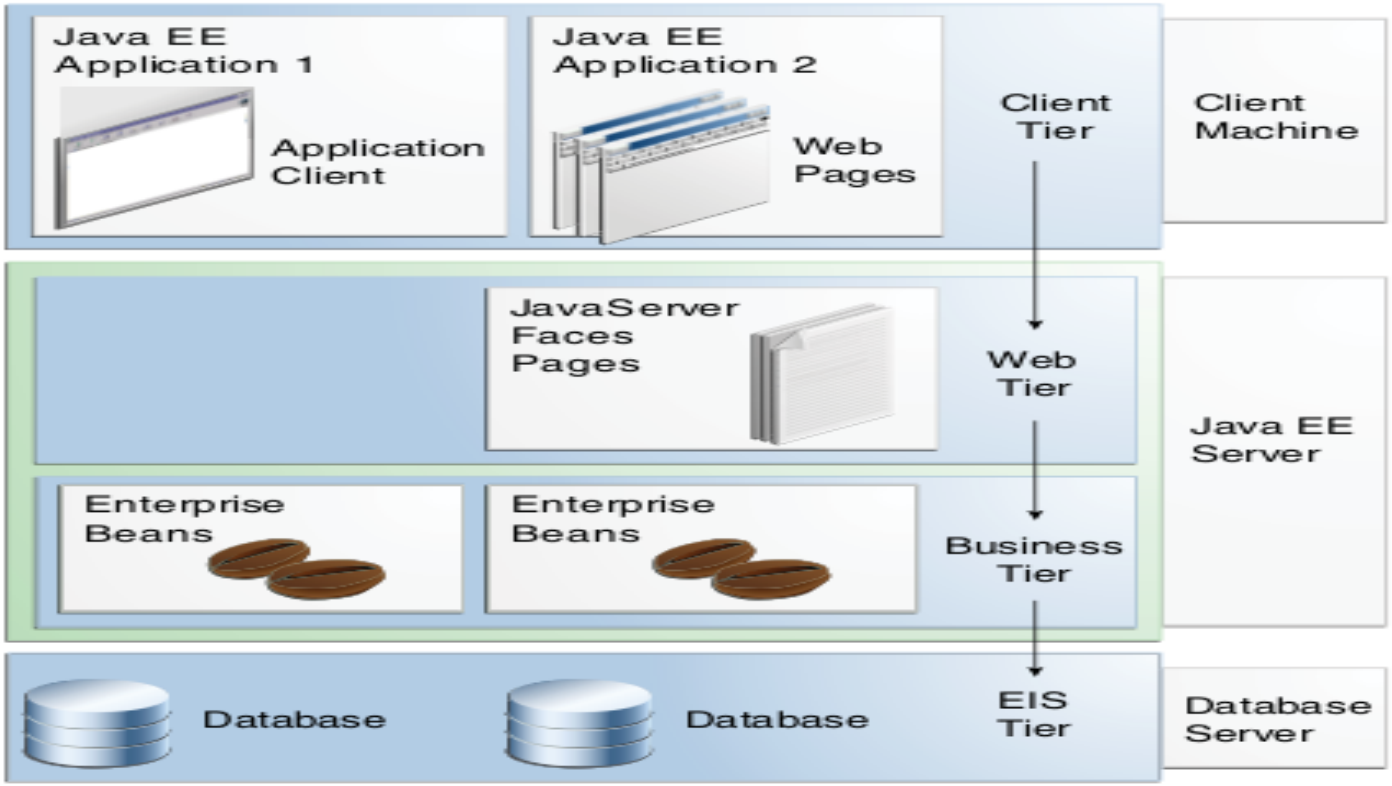
- Programación eficiente.
- Extensibilidad frente a la demanda del negocio.
- Integración.

Arquitectura JEE

La arquitectura JEE implica un modelo de aplicaciones distribuidas en diversas capas o niveles (tier).

Niveles o Capas JEE

Cliente	HTML, Applet, aplicaciones Java, etc.
Web	JSF, JSP, Servlet
Negocio	EJB, JPA, JAX-WS y JAX-RS Web services
Sistema de Información Empresariales	JDBC, JTA, Java EE Connector, JPA



Aplicación Empresarial multicapas

Servidores Java EE

Es un servidor de aplicaciones que implementa los APIs de la plataforma Java EE y provee los servicios del estándar Java EE.

Los servidores Java EE muchas veces son llamados servidores de aplicaciones, porque permiten servir datos a los clientes, de la misma forma que un servidor web permite servir páginas web a un browser.

Un servidor Java EE puede alojar varios tipos de componentes correspondientes a una aplicación multi-capas. En este sentido, ofrece un entorno de ejecución estandarizado para estos componentes.

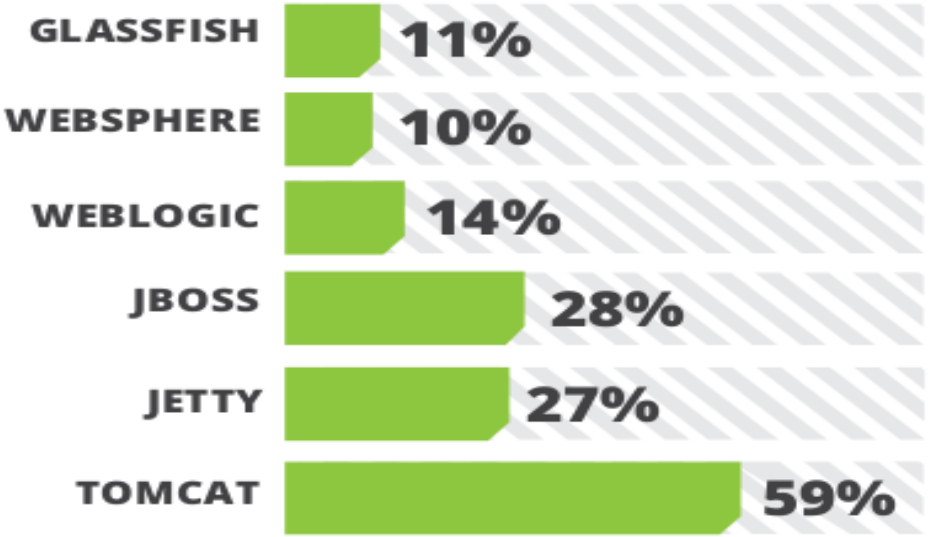
Servidores Java EE

Servidores de Aplicaciones certificados para Java EE 6

Servidores Java EE

MAKE THE FUTURE JAVA

Popularidad



Junio 2012

Contenedores Java EE

Son la interfaz entre el componente y la funcionalidad de bajo nivel proporcionada por la plataforma para soportar ese componente.

La funcionalidad del contenedor está definido por la plataforma, y es diferente para cada tipo de componente.

No obstante, el servidor permite que los diferentes tipos de componentes puedan trabajar juntos para proporcionar funcionalidad en una aplicación empresarial.

Tipos de contenedores

Contenedor Web:

Maneja la ejecución de los Servlets y páginas JSP y JSF.

Contenedor Cliente:

Provee una interfaz de conexión entre el servidor Java EE y las aplicaciones clientes, tales como aplicaciones Java SE, entre otras.

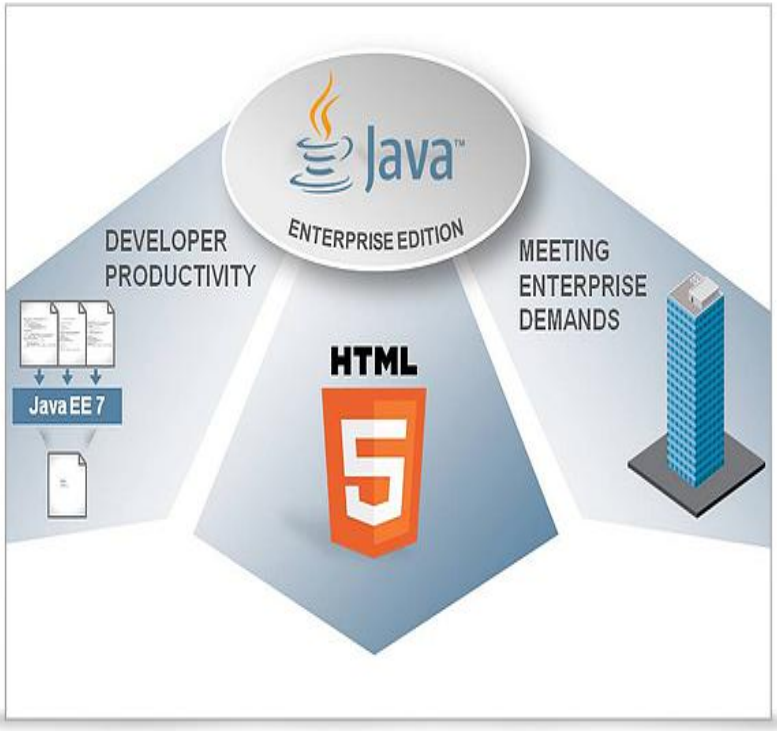
Contenedor EJB:

Gestiona la ejecución de los Enterprise JavaBeans.

Aspectos importantes de Java EE 7

El objetivo más **importante** de la plataforma Java EE 7 es **simplificar** el desarrollo proporcionando una **base común** para los diversos tipos de componentes en la plataforma Java EE.

Los desarrolladores se benefician de las mejoras de **productividad** con más **anotaciones** y **menos configuración XML**, más **Plain Old Java Objects (POJOs)** y paquetes simplificados.



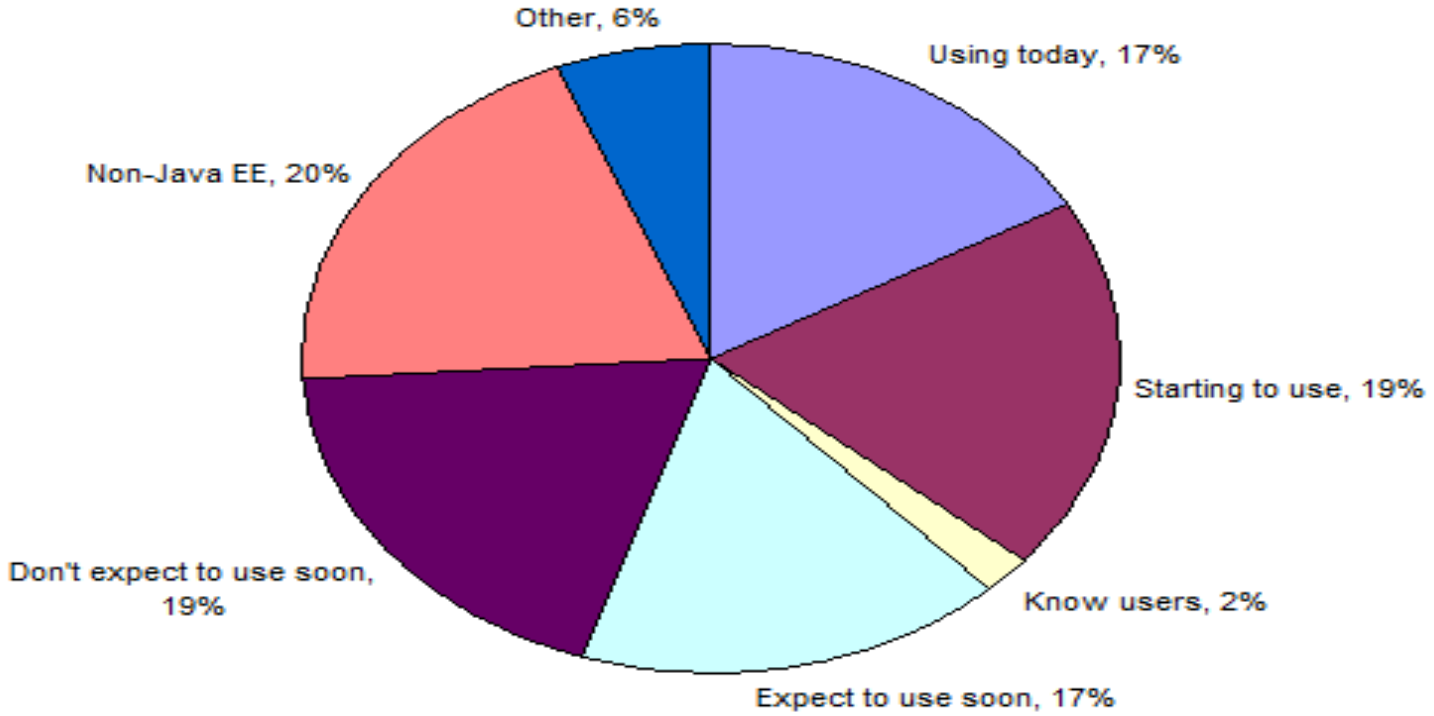
Reacciones de la comunidad

"**Only a few years ago, Java EE was used mostly by larger companies, now it becomes interesting even for one-person shows**." Adam Bien, Java Champion.

"**Java** is already the **most pervasive server side development** and **deployment platform** for the **cloud**. With **Java EE 7**, **enterprise developers** and **deployers** will see additional cloud-specific **innovation** built into the standard and available across many **vendor implementations**." Oracle Inc.

"Java is already widely used in cloud environments, so it's important that **Java EE 7** standards evolve into a **modern technology foundation** that satisfies the growing demand for a viable s**tandards-based programming** model choice for the **cloud**." SAP

Adopción de Java EE 7



https://blogs.oracle.com/theaquarium/entry/survey_sez_java_ee_71

Java EE 7 APIs

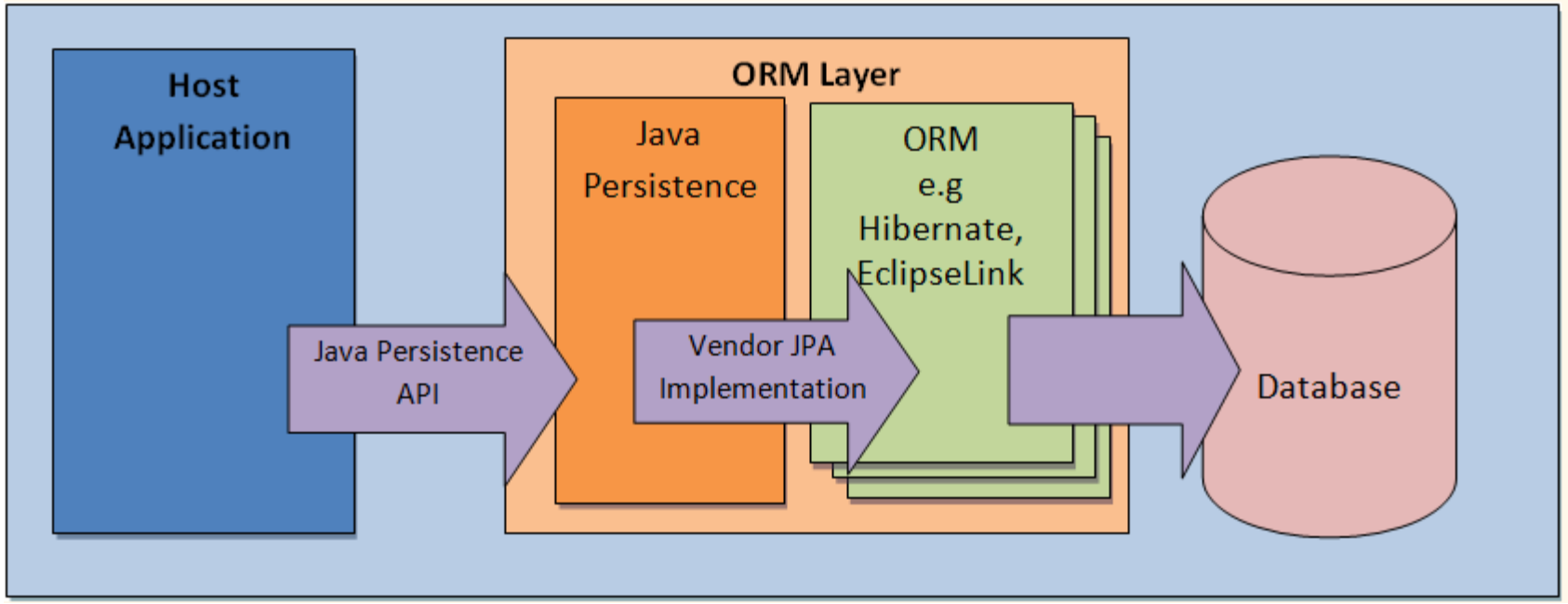
Java EE 7 APIs

- 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)

Java Persistence API (JPA) 2.1

- Framework para manejar los datos de una base de datos relacional a un mapeo de objeto. ORM.
- Permite que nuestra aplicación no dependa de un vendedor de base de datos.
- Representan una abstracción para manejar las diferentes implementaciones existentes:
  - Hibernate.
  - EclipseLink.
  - OpenJPA.

Java Persistence API (JPA) 2.1



Java Persistence API (JPA) 2.1

MAKE THE FUTURE JAVA

Clase Entidad - JPA

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 *
 * @author vacax
 */
@Entity
@Table(name = "estudiantes")
public class Estudiante implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int matricula;
    private String nombre;

    /**...3 lines */
    public int getMatricula() {...3 lines }

    /**...3 lines */
    public void setMatricula(int matricula) {...3 lines }

    /**...3 lines */
    public String getNombre() {...3 lines }

    /**...3 lines */
    public void setNombre(String nombre) {...3 lines }

}
```

Java Persistence API (JPA) 2.1

MAKE THE FUTURE JAVA

Archivo Persistencia - JPA

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="TallerJavaEE7PU" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>jdbc/sample</jta-data-source>
        <properties>
            <property name="javax.persistence.schema-generation.database.action" value="create"/>
        </properties>
    </persistence-unit>
</persistence>
```

Java Persistence API (JPA) 2.1

Insertando un Objeto

```java
/**
 *
 * @author vacax
 */
@Stateless
public class MiEJB {
    @PersistenceContext(unitName = "TallerJavaEE7PU")
    private EntityManager em;

    public Estudiante persistirEstudiante(int matricula, String nombre) {
        Estudiante estudiante=new Estudiante(matricula, nombre);
        em.persist(estudiante);
        return estudiante;
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")


}
```

Java Persistence API (JPA) 2.1

- Los elementos nuevos JPA 2.1:
  - Soporte procedimientos almacenados
  - Tipo de conversión
  - Criteria Update / Delete
  - Creación runtime named Queries.
  - Entre otros.

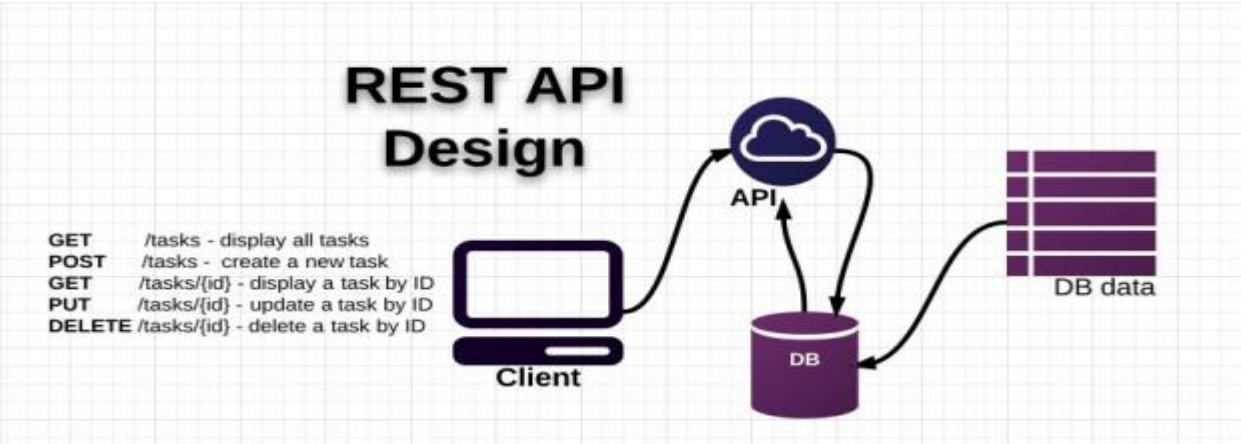
Java EE 7 APIs

- 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)

RESTful WebServices 2.1

MAKE THE FUTURE JAVA

- Representational State Transfer (REST):
  - 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).



RESTful WebServices 2.1

- JAX-RS 1.0:
  - Liberada en el 2008 para la versión Java EE 6. JSR-311
  - Simplifica el proceso de creación de servicios Web mediante Plain Old Java Objects (POJOS) y anotaciones.
  - No requiere configuración adicional para utilizarlos en entornos Java EE 6.
  - Incluye anotaciones para implementar servicios Web:
    - @Path, @Get, @Put, @Post, @Delete, @Produces, @Consumes, entre otros.

RESTful WebServices 2.1

Ejemplo Clase JAX-RS

```java
//Omitiendo los import javax.ws.rs.*;
/**
 * REST Web Service
 *
 * @author vacax
 */
@Path("restful")
public class MiServicioRestful {

    @Context
    private UriInfo context;
    @EJB
    private MiEJB miEJB;


    public MiServicioRestful() {
    }

    /**
     * Es ejecutado mediante la URL $contexto/webresources/restful y retorna
     * una lista de estudiantes en formato JSON.
     */
    @GET
    @Produces("application/json")
    public List<Estudiante> getListadoEstudiantes() {
        System.out.println("Recuperando lista de estudiantes");
        return miEJB.getListaEstudiantes();
    }

}
```

RESTful WebServices 2.1

MAKE THE FUTURE JAVA

Resultado llamada URL, método GET

RESTful WebServices 2.1

- JAX-RS 2.0:
  - Introduce elementos que ayudan a la productividad.
  - Simplifica el API
  - Incluye los siguientes aspectos:
    - API para el Cliente
    - Llamadas Asincrónicas vía Http
    - Filtros e interceptores

RESTful WebServices 2.1

- API para el Cliente

```
Client client = ClientFactory.newClient();

WebTarget target = client.target("http://example.com/shop");
Form form = new Form().param("customer", "Bill")
                      .param("product", "IPhone 5")
                      .param("CC", "4444 4444 4444 4444");

Response response = target.request().post(Entity.form(form));
assert response.getStatus() == 200;

Order order = response.readEntity(Order.class);
```

RESTful WebServices 2.1

- API Cliente Asincrónico

```
InvocationCallback<Response>
callback = new InvocationCallback {
   public void completed(Response res) {
      System.out.println("Request success!");
   }

   public void failed(ClientException e) {
      System.out.println("Request failed!");n
   }
};
client.target("http://example.com/customers")
      .queryParam("name", "Bill Burke")
      .request()
      .async()
      .get(callback);
```

RESTful WebServices 2.1

- Server-side Asincrónico Http

```
@Path("/listener")
public class ChatListener{

   List<AsyncResponse> listeners = ...some global list...;


   @GET
   public void listen(@Suspended AsyncResponse res) {
      list.add(res);
   }

}
```

RESTful WebServices 2.1

- Server-side Filtros - Antes de la Ejecución

```
@Provider
public class CacheControlFilter implements ContainerResponseFilter {
   public void filter(ContainerRequestContext req, ContainerResponseContext res) {
      if (req.getMethod().equals("GET")) {
         req.getHeaders().add("Cache-Control", cacheControl);
      }
   }
}
```

RESTful WebServices 2.1

- Server-side Filtros - Después de la Ejecución

```
@Provider
public class HttpOverride implements ContainerRequestFilter {

   public void filter(ContainerRequestContext ctx) {
      String method = ctx.getHeaderString("X-Http-Method-Override");
      if (method != null) ctx.setMethod(method);
   }
}
```

RESTful WebServices 2.1

- Interceptores

```
@Provider
//Codifica una salida con GZIP.
public class GZIPEndoer implements WriterInterceptor {
   public void aroundWriteTo(WriterInterceptorContext ctx) throws IOException,
WebApplicationException {
      GZIPOutputStream os = new GZIPOutputStream(ctx.getOutputStream());
      try {
         ctx.setOutputStream(os);
         return ctx.proceed();
      } finally {
         os.finish();
      }
   }
}
```

Java EE 7 APIs

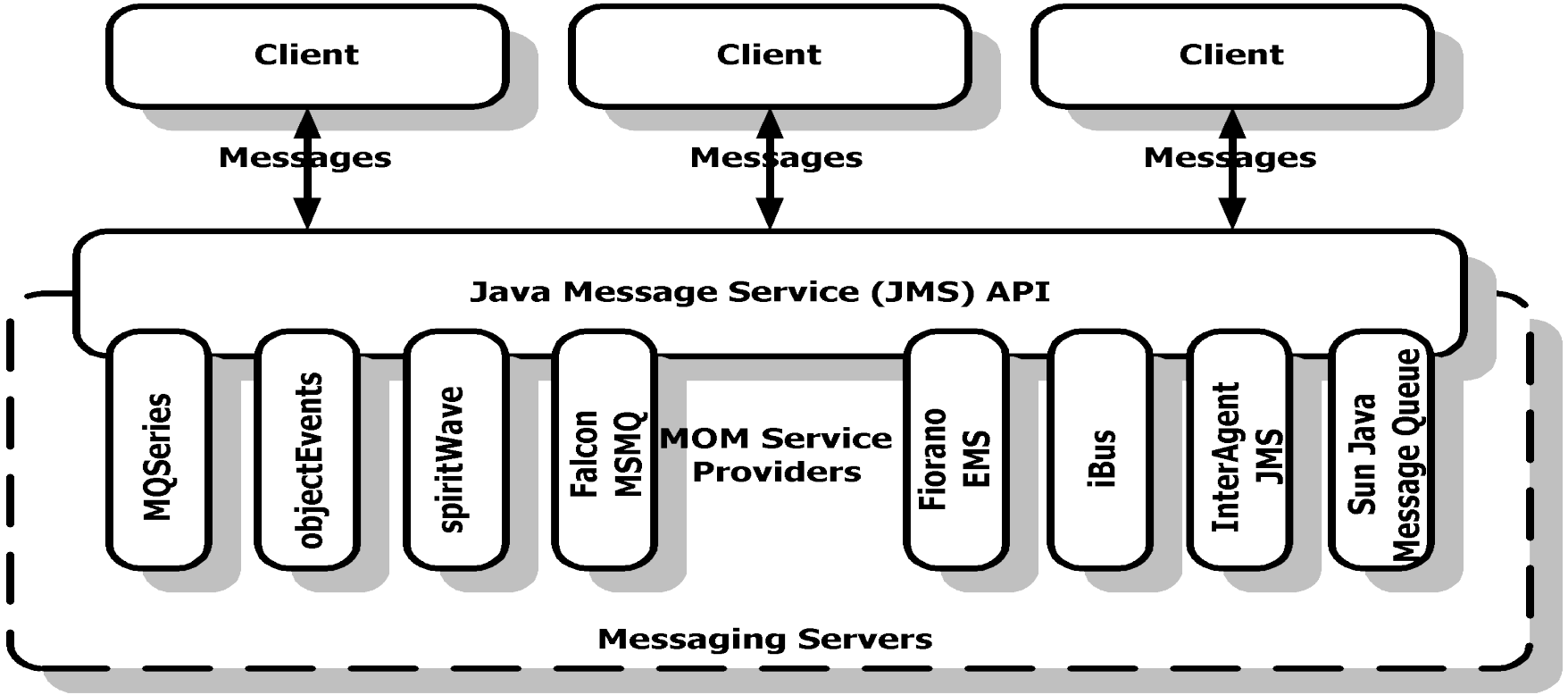
- Java EE 7 cubre las siguientes especificaciones:
  - Java Persistence API 2.1 (JSR 338)
  - Java API for RESTful Web Services 2.0 (JSR 339)
  - **Java Message Service 2.0 (JSR 343)**
  - JavaServer Faces 2.2 (JSR 344)
  - Contexts and Dependency Injection 1.1 (JSR 346)
  - Bean Validation 1.1 (JSR 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)

Java Message Service 2.0 (JSR 343)

Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java crear, enviar, recibir y leer mensajes.

- Permite comunicación confiable de manera síncrona y asíncrona.
- JMS 1 fue desarrollada con la versión JDK 1.4
- JSR 343: JMS 2.0, la nueva versión de la especificación JMS evoluciona y permite realizar funcionalidad que, antes, se realizaba en 20 líneas, en 6; por poner un simple ejemplo.

Java Message Service 2.0 (JSR 343)



Java Message Service 2.0 (JSR 343)

El cambio más grande en JMS 2.0 es la introducción de una nueva API para enviar y recibir mensajes, que reduce la cantidad de código de un desarrollador debe escribir.

Para aplicaciones JEE la nueva API también soporta la inyección de recursos. Esto permite que el servidor de aplicaciones gestiones los objetos JMS, lo que simplifica aún más la aplicación.

JMS 2.0 es parte de la plataforma Java EE 7 y se puede utilizar en la Web o EJB aplicaciones Java EE, o se puede utilizar independiente en un entorno Java SE.

Java Message Service 2.0 (JSR 343)

JMS 1.1

```
public void sendMessage(String payload) {
    Connection connection = null;
    try {
        connection = connectionFactory.createConnection();
        connection.start();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(demoQueue);
        TextMessage textMessage = session.createTextMessage(payload);
        messageProducer.send(textMessage);
    } catch (JMSException ex) {
        ex.printStackTrace();
    } finally {
        if (connection != null) {
            try {
                connection.close();
            } catch (JMSException ex) {
                ex.printStackTrace();
            }
        }
    }
```

Java Message Service 2.0 (JSR 343)

JMS 2.0

```
public void sendMessage(String message) {
    context.createProducer().send(myQueue, message);
}
```

Java EE 7 APIs

- 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)

JavaServer Faces 2.2 (JSR 344)

- 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

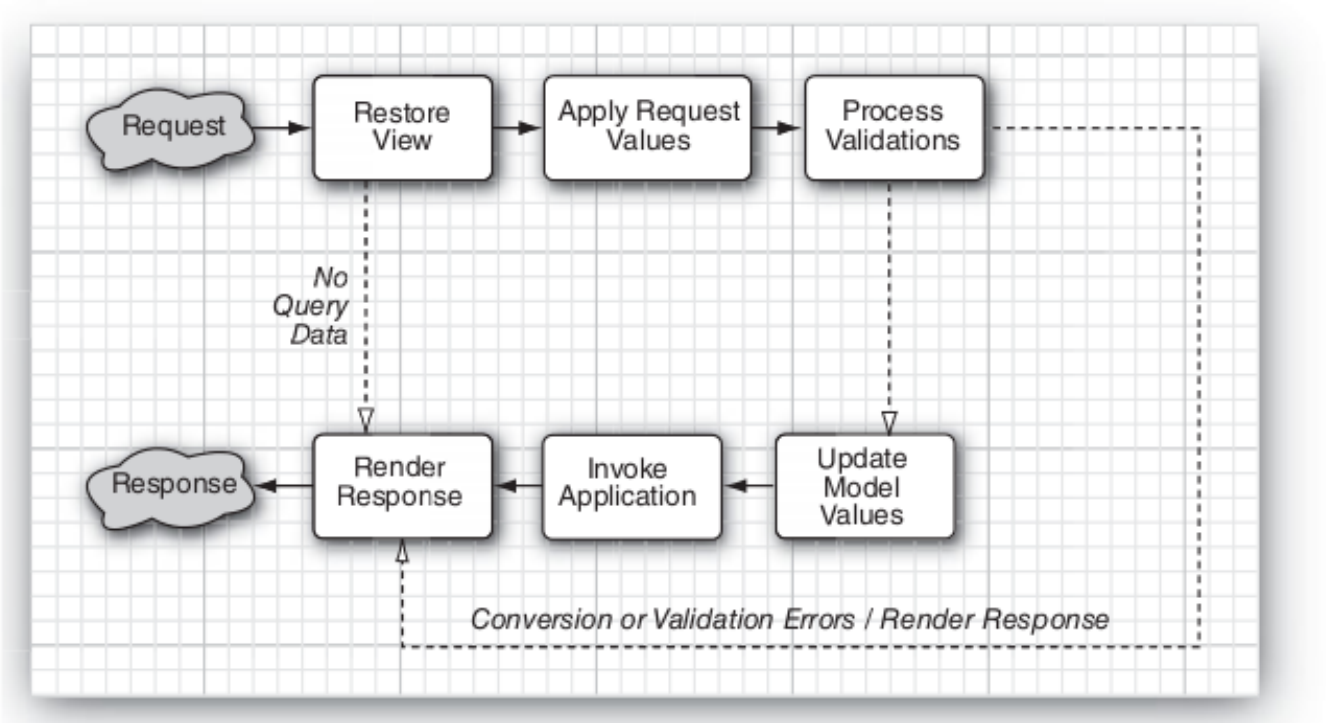
JavaServer Faces 2.2 (JSR 344)

- ¿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.

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- Ciclo de vida en JSF



JavaServer Faces 2.2 (JSR 344)

- 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.

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- Managed Beans

```
 */
@ManagedBean(name = "miFormulario")
@SessionScoped
public class MiFormularioBean implements Serializable{

    @Size(max = 5)
    private String nombre;
    private String apellido;
    int contador;

    public String procesarDatosForm(){
        /**
         * Controlar cualquier cosa que sea neceario en su procesamiento...
         */
        nombre=nombre.toUpperCase();
        //
        return "resultado?faces-redirect=true";
    }

    public int getContador() {
        return contador;
    }

    public void setContador(int contador) {
        this.contador = contador;
    }
```

Definición

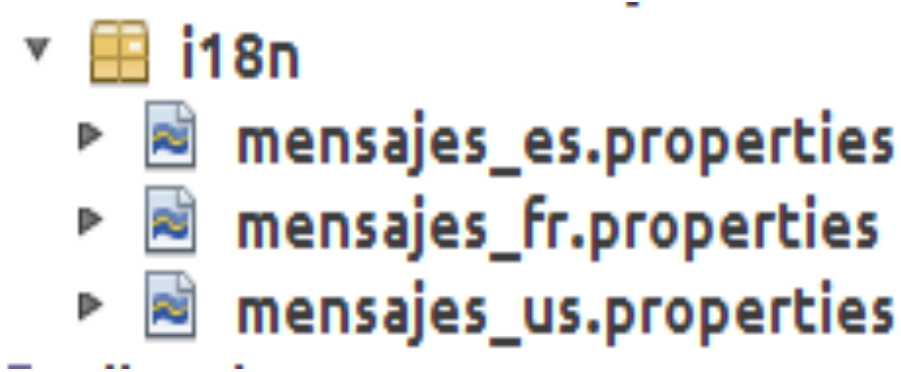
Propiedades

Set & Get

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- I18n - Internacionalización
  - Permite construir aplicaciones multi-idiomas de una manera limpia y práctica.
  - Se apoya de archivos de propiedades para estos fines.



JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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"));
   }
```

JavaServer Faces 2.2 (JSR 344)

- 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.

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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>
```

JavaServer Faces 2.2 (JSR 344)

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

JavaServer Faces 2.2 (JSR 344)

- 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

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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>
```

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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>
```

JavaServer Faces 2.2 (JSR 344)

- 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

JavaServer Faces 2.2 (JSR 344)

- JSF 2.2
  - Pertenece al JEE 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.

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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>
```

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- JSF 2.2 - File Upload

```
 */
@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;
    }


}
```

JavaServer Faces 2.2 (JSR 344)

- JSF 2.2 - HTML5
    - Se incluye el taglib: xmlns:pt="http://xmlns.jcp.org/jsf/passthrough para traspasar propiedades.
    - Se incluyo la etiqueta f:passThroughAttribute y f: passThroughAttributes para combinar con las etiqutas JSF.
    - Incluye el traspaso de elementos.

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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>
```

JavaServer Faces 2.2 (JSR 344)

MAKE THE FUTURE JAVA

- 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 EE 7 APIs

- 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)

Contexts and Dependency Injection 1.1 (JSR 346)

La **inyección de dependencias** consiste en cargar las propiedades de los beans mediante su constructor o sus setters en el momento de iniciar la aplicación.

Sin inyección de dependencias, cada clase llama al objeto que necesita en **tiempo de ejecución**.

Mientras que con **inyección de dependencias**, cada objeto es cargado en cada clase que lo necesita en tiempo de inicialización.

La forma habitual de implementar este patrón es mediante un "Contenedor DI"

Contexts and Dependency Injection 1.1 (JSR 346)

En Java EE 6, era necesario agregar explícitamente un archivo beans.xml con el fin de habilitar escaneo CDI.

La DI (**Inyección de dependencia**) se considera correctamente una preocupación generalizada, los desarrolladores de Java EE a menudo se encuentran con este requisito confuso.

Uno de los cambios más significativos de CD 1.1/Java EE 7 es que por petición popular, el **CDI** está **activada** de **forma predeterminada**.

Contexts and Dependency Injection 1.1 (JSR 346)

Esto significa que no hay necesidad de añadir explícitamente un archivo beans.xml para permitir DI.

Sin embargo, también es muy importante entender que la CDI ahora también proporciona un control más preciso sobre la exploración de componente mediante el atributo 'bean-discovery-mode'.

Este atributo tiene tres valores posibles:

Contexts and Dependency Injection 1.1 (JSR 346)

'annotated' - traducido libremente, significa que sólo se procesan los componentes con una anotación de nivel de clase.

'all' - todos los componentes son procesados, al igual que lo fueron en Java EE 6, con la beans.xml explícito.

'none' - CDI estará desactivada.

De forma predeterminada, si no se especifica nada y no hay beans.xml, el modo bean-discovery 'annotated' es asumido, en lugar de suponer "all".

Contexts and Dependency Injection 1.1 (JSR 346)

Ejemplo básico

```
51  /**
52   *
53   * @author arungup
54   */
55  @WebServlet(urlPatterns = {"/ServerServlet"})
56  public class ServerServlet extends HttpServlet {
57
58      @Inject MyRequestScopedBean requestBean;
59      @Inject MyRequestScopedBean requestBean2;
60
61      @Inject MySessionScopedBean sessionBean;
62      @Inject MySessionScopedBean sessionBean2;
63
64      @Inject MyApplicationScopedBean applicationBean;
65      @Inject MySingletonScopedBean singletonBean;
66
67
68      /**
69       * Processes requests for both HTTP
70       * <code>GET</code> and
```

Java EE 7 APIs

- 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)

Bean Validation 1.1 (JSR 349)

Bean Validation define un modelo de metadatos y una interfaz para la validación de **JavaBeans**.

La fuente de los metadatos consiste en anotaciones, con la posibilidad de sobreescribir y extender estos metadatos por medio del uso de descriptores de validación en formato XML.

Java Bean Validation (JSR 303) fue aprobado por el JCP al 16 de noviembre de 2009, siendo aceptado como parte de la especificación Java EE 6.

Bean Validation 1.1 (JSR 349)

Bean Validation 1.1 se estuvo enfocado en los siguientes temas:

- Apertura de la especificación y su proceso
- Validación a nivel de método (validación de parámetros o valores de retorno)
- Inyección de dependencias para los componentes Bean Validation
- Integración con el contexto y la inyección de dependencias (CDI)
- Conversión de grupo
- Interpolación de mensaje de error utilizando las expresiones EL

Bean Validation 1.1 (JSR 349)

MAKE THE FUTURE JAVA

Validación de métodos

```
public String sayHello(@Size(max = 3)String name) {
    return "Hello " + name;
}

@Future
public Date showDate(boolean correct) {
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DAY_OF_MONTH, correct ? 5 : -5);
    return cal.getTime();
}

public String showList(@NotNull @Size(min=1, max=3) List<String> list, @NotNull String prefix) {
    StringBuilder builder = new StringBuilder();

    for (String s : list) {
        builder.append(prefix).append(s).append(" ");
    }

    return builder.toString();
}
```

Bean Validation 1.1 (JSR 349)

Integración con CDI

```
class ZipCodeValidator implements ConstraintValidator<ZipCode, String> {

    @Inject @France
    private ZipCodeChecker checker;

    public void initialize(ZipCode zipCode) {}

    public boolean isValid(String value, ConstraintValidationContext context) {
        if (value==null) return true;
        return checker.isZipCodeValid(value);
    }
}
```

Java EE 7 APIs

- 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)

Batch Applications - JSR 352

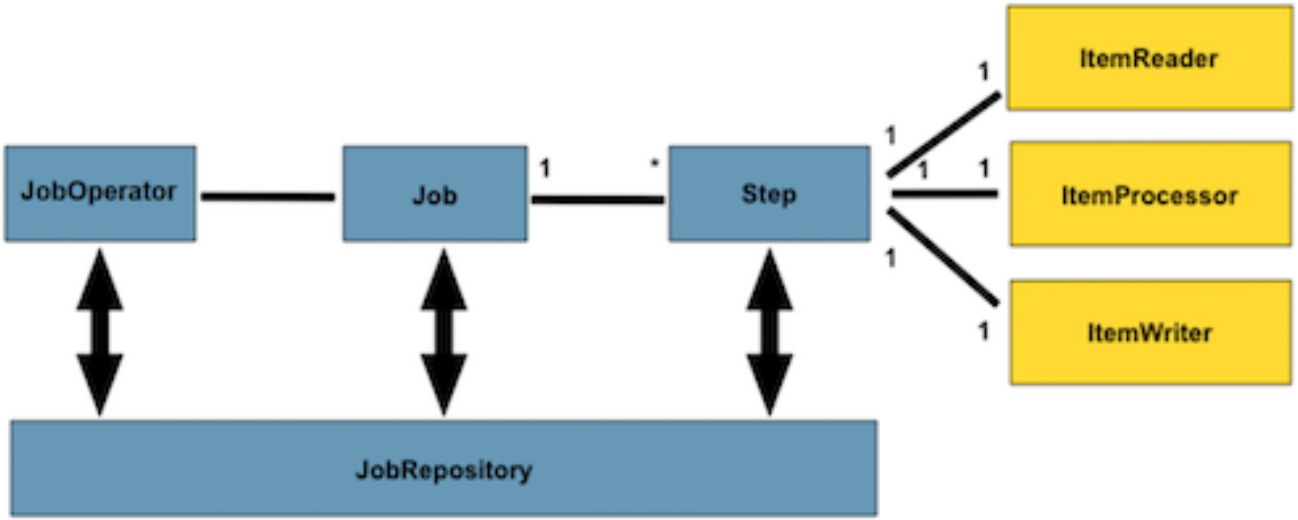
- Concepto viejo y utilizando por décadas.
- La especificación busca normalizar la construcción y ejecución aplicaciones que necesitan procesos Batch.
- Se enfoca en:
  - Modelo de programación de los Batch.
  - Lenguaje para definir los Jobs.
  - Controlador de los jobs en proceso. (Runtime).

Batch Applications - JSR 352

MAKE THE FUTURE JAVA

Java

Diagrama Batch



Batch Applications - JSR 352

MAKE THE FUTURE JAVA

Java

Definición del Job-Xml

```
<job id="myJob" xmlns="http://batch.jsr352/jsl">
<step id="myStep" >
<chunk reader="MyItemReader"
writer="MyItemWriter"
processor="MyItemProcessor"
buffer-size="5"
checkpoint-policy="item"
commit-interval="10" />
</step>
</job>
```

Batch Applications - JSR 352

MAKE THE FUTURE JAVA

Java

ItemReader

```
@ItemReader
public class MyItemReader {
 private static int id;
 MyCheckPoint checkpoint = null;

 @Open
 void open(MyCheckPoint checkpoint) {
 this.checkpoint = checkpoint;
 System.out.println(getClass().getName() + ".open: " + checkpoint.getItemCount());
 }

 @ReadItem
 MyBatchRecord read() {
 checkpoint.incrementByOne();
 return new MyBatchRecord(++id);
 }

 @CheckpointInfo
 MyCheckPoint getCheckPoint() {
 return checkpoint;
 }
}
```

Batch Applications - JSR 352

Procesador

```
@ItemProcessor
public class MyItemProcessor {
 @ProcessItem
 MyBatchRecord process(MyBatchRecord record) {
 return (record.getId() % 2 == 0) ? record : null;
 }
}
```

Batch Applications - JSR 352

MAKE THE FUTURE JAVA

Java

ItemWriter

```
@ItemWriter
public class MyItemWriter {
 MyCheckPoint checkpoint = null;

 @Open
 void open(MyCheckPoint checkpoint) {
 this.checkpoint = checkpoint;
 System.out.println(getClass().getName() + ".open: " + checkpoint.getItemCount());
 }

 @WriteItems
 void write(List<MyBatchRecord> list) {
 System.out.println("Writing the chunk...");
 for (MyBatchRecord record : list) {
 System.out.println(record.getId());
 }
 checkpoint.increment(list.size());
 System.out.println("... done.");
 }

 @CheckpointInfo
 MyCheckPoint getCheckPoint() {
 return checkpoint;
 }
}
```

Batch Applications - JSR 352

MAKE THE FUTURE JAVA

Java

CheckPoint

```
public class MyCheckPoint {
 int itemCount;

 public int getItemCount() {
 return itemCount;
 }

 public void setItemCount(int itemCount) {
 this.itemCount = itemCount;
 }

 void incrementByOne() {
 itemCount++;
 }

 void increment(int size) {
 itemCount += size;
 }
}
```

Java EE 7 APIs

- 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)

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)

Java API for JSON Processing 1.0 (JSR 353)

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" }
   ]
}
```

Java API for JSON Processing 1.0 (JSR 353)


```
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();
```

Escribiendo en un archivo

Java API for JSON Processing 1.0 (JSR 353)

MAKE THE FUTURE JAVA


```
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;
   }
}
```

Leyendo desde un archivo con formato JSON

Java EE 7 APIs

- 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)

Java API for WebSocket 1.0 (JSR 356)

WebSocket:
Es un nuevo protocolo derivado de HTTP.

HTTP es el protocolo estándar para la Web, es muy efectivo para una gran cantidad de casos de uso pero, sin embargo, tiene algunos inconvenientes en el caso de aplicaciones Web interactivas:

half-duplex: basado en el modelo de solicitud / respuesta, el cliente envía una petición y el servidor realiza el procesamiento antes de enviar una respuesta, el cliente se ve obligado a esperar una respuesta del servidor

Java API for WebSocket 1.0 (JSR 356)

verbose: una gran cantidad de información se envía en las cabeceras HTTP asociados con el mensaje, tanto en la solicitud HTTP y en la respuesta HTTP

con el fin de añadir un modo de inserción en el servidor, es necesario utilizar solución (poll, long poll, Comet / Ajax), ya que no existe un estándar.

Este protocolo no está optimizado a escala en aplicaciones de gran tamaño que tienen importantes necesidades de comunicación en tiempo real bidireccional.

Java API for WebSocket 1.0 (JSR 356)

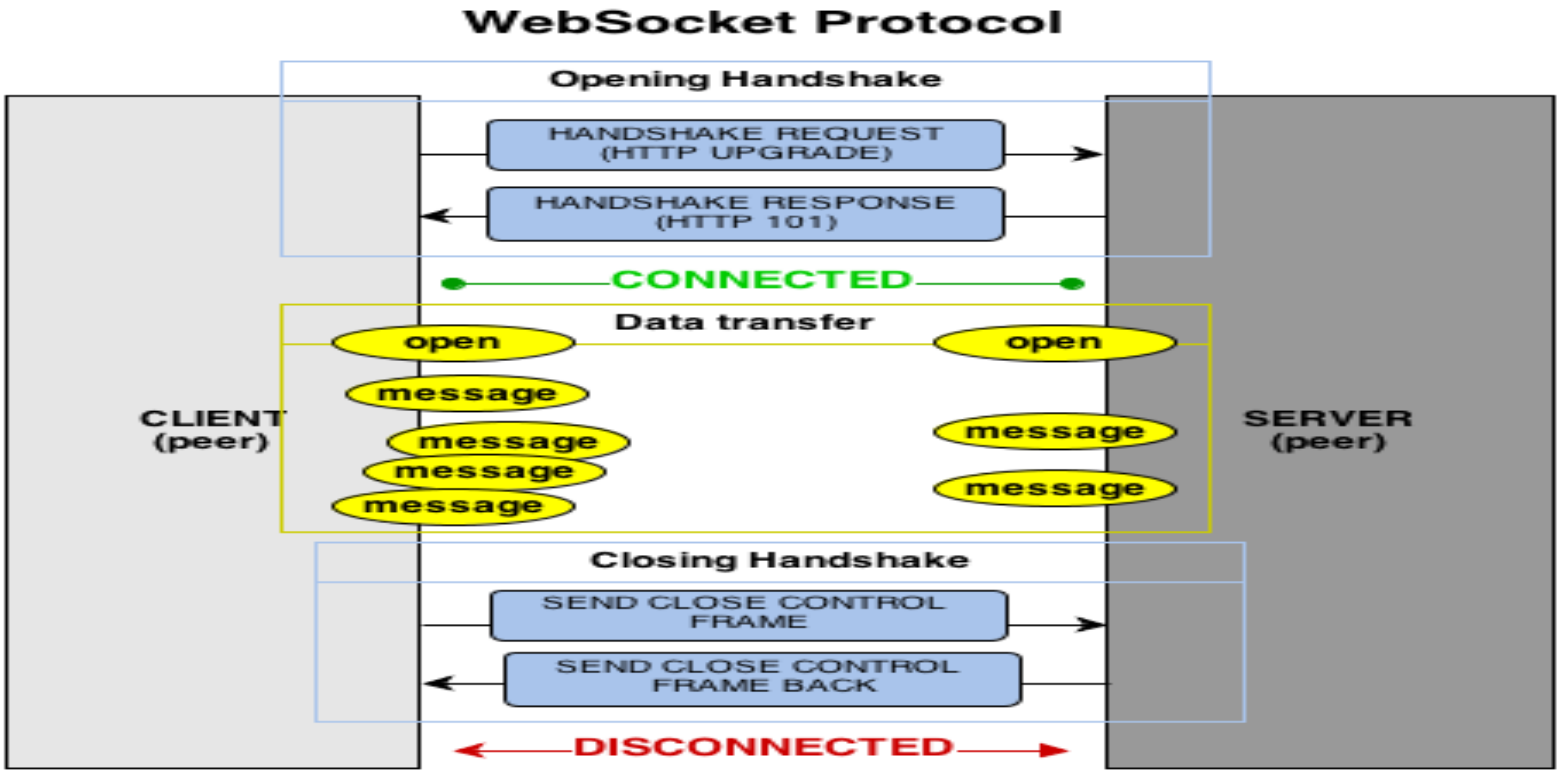
Por ello, el nuevo protocolo WebSocket ofrece funciones más avanzadas de HTTP porque es:

Basado en 1 conexión TCP única entre 2 pares (mientras que cada solicitud / respuesta HTTP necesita una nueva conexión TCP)

Bidireccional: cliente puede enviar el mensaje al servidor y el servidor también puede enviar un mensaje al cliente

full-duplex: cliente puede enviar varios mensajes al servidor, así como el servidor al cliente sin esperar una respuesta de sí

Java API for WebSocket 1.0 (JSR 356)



Java API for WebSocket 1.0 (JSR 356)

MAKE THE FUTURE JAVA

WebSocket Server Endpoint

JAVA

```java
import javax.websocket.OnMessage;
import javax.websocket.ServerEndpoint;

@ServerEndpoint("/echo") (1)
public class EchoServer {

        @OnMessage (2)
        public String handleMessage(String message){
                return "Thanks for the message: " + message;
        }

}
```


Java
Dominicano

Java Dominicano

Es un grupo de entusiastas de la Plataforma Java, enfocados en promover el conocimiento Java en el presente y futuras generaciones de programadores.

A través de nuestros foros y reuniones periódicas que pueda mantenerse en contacto con los últimos desarrollos de la industria, aprender nuevas tecnologías de Java (JVM y otros), conocer a otros desarrolladores, discutir temas técnicos / no técnicos y de la red aún más en toda la Comunidad Java.

Enfoque técnico

- Plataforma Java (JSE).
- Técnicas y Herramientas de desarrollo.
- Lenguajes emergentes en la JVM.
- Java Enterprise Edition (JEE).
- Java Embebidos(Java ME, Java Card, etc).
- JavaFX y RIA.
- Frameworks Java.
- IDE

Redes sociales

Contactos:
info@javadominicano.org

Twitter:
@javadominicano

Facebook:
Grupo Java Dominicano

Google Groups:
@letsrockjava

¡Gracias por acompañarnos!

Referencias

https://javaee7.zeef.com/arjan.tijms

https://github.com/javaee-samples/javaee7-samples

http://javaee-samples.github.io/

http://mgreau.com/posts/2013/11/11/javaee7-websocket-angularjs-wildfly.html

http://www.calcey.com/calcey/conforming-to-jpa-2-1-for-persistence-a-practical-experience/

http://java.dzone.com/articles/whats-new-jax-rs-20

http://www.infoq.com/news/2013/06/Whats-New-in-JAX-RS-2.0

http://www.slideshare.net/edburns/jsf-22-26091922