



**Comunicacions d'empresa**  
(SQL, cgi, Servlets i ASP)

**JAVIER MOYA i PAYÀ**  
ETSITelecomunicacions  
UPV2005

**Comunicacions d'empresa** Servlets

**JAVIER MOYA i PAYÀ**  
**ETSITelecomunicacions**  
**UPV2005**

Las clases principales son **DriverManager**, **Connection**, **Statement** y **ResultSet**. Los métodos de esta clase permiten que el servidor se comunique con la base de datos; por su parte los métodos de la clase Servlet permiten que el cliente se comunique con el servidor (y este a su vez mediante el puente JDBC/ODBC con la información residente en sí).

Nombre de la clase	Funcionalidad	Métodos básicos	Ejemplo
DriverManager	Permite fijar qué driver de los residentes en el servidor se utilizará para comunicarse con la base de datos	-	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		static Connection getConnection( driver, user pass);	Connection conexion = DriverManager.getConnection("jdbc:ODBC:Nombre_BD", user, pass);
		void close();	conexion.close();
Connection	Es la clase que crea la conexión entre el servidor y la base de datos. Esta conexión utiliza los drivers que se hayan definido en DriverManager. Esta clase no ejecuta métodos SQL pero tiene métodos para generar los objetos que sí las ejecutan	<b>Estando declarada una referencia del tipo Connection</b> Connection conexion = DriverManager.getConnection("jdbc:ODBC:Nombre_BD", user, pass);	
		Satatement createStatement();	Statement orden = conexion.createStatement(); <i>Crea un Statement para poder realizar consultas SQL</i>
		void setAutoCommit( boolean activo)	conexion.setAutoCommit( false); <i>Activa el modo transaccional de forma que las intrucciones que haya entre esta instrucción y conexion.commit() se ejecutaran de forma atómica (si se llega precisamente hasta commit)</i>
		void commit();	conexion.commit(); <i>Ejecuta (en lote) todas las órdenes solicitadas que haya escritas cuando el modo transaccional haya sido activado. En ningún caso se ejecutan antes</i>
		void rollback();	conexion.rollback(); <i>Deja las variables modificadas de forma transaccional como estaban ("como si no hubiésemos hecho nada")</i>

Statement	Sus objetos contienen métodos para poder ejecutar las órdenes SQL	<b>Estando declarada una referencia del tipo Statement</b> Statement orden = conexion.createStatement();	
		int executeUpdate( String instruccionSQL);	int numRegistros = orden.executeUpdate( "SELECT * ..."); <i>Devuelve el número de registros que la orden SQL según instruccionSQL haya modificado</i>
		ResultSet executeQuery( String instruccionSQL)	ResultSet rs = orden.executeQuery( "SELECT * ..."); <i>Realiza una consulta a la base de datos y con los registros extraídos se crea una tabla temporal (objeto rs tipo ResultSet) que permite consultar a posteriori los datos extraídos</i>
ResultSet	Los objetos de esta clase contienen los datos que la 'orden' haya conseguido extraer de la base de datos. Es sobre estos objetos sobre los cuales ejecutamos órdenes de lectura de información	<b>Estando declarada una referencia del tipo ResultSet</b> ResultSet rs = orden.executeQuery( "SELECT * ...");	
		boolean next();	rs.next(); <i>Indica si hay más filas que recorrer en el ResultSet rs</i>
		String getString( int numClomuna); Long getLong( int numColumna); ... Xxx getXxx( int numColumna);	Xxx valorLeido = rs.getXxx( int numColumna); <i>Devuelve el valor (de tipo Xxx) que se encuentre en la intersección entre la fila a la que actualmente se esté apuntando en el ResultSet y numColumna</i>
		void close();	rs.close(); <i>"Vacía" la consulta. ¡¡¡IMPORTANTE!! Entre dos consultas, hay que cerrar siempre el rs, es decir: rs.executeQuery( SQL1) → rs.close(); → rs.executeQuery( SQL2);</i>

Un servlet es un programa escrito en Java que un cliente puede lanzar remotamente a través de una conexión TCP. El servlet, en su código, puede realizar infinidad de tareas a raíz del código que lo implemente; fundamentalmente serán instrucciones de acceso a bases de datos utilizando para nuestro caso SQL y JDBC. La utilización de servlets pasa por asumir los árboles de herencia **Servlet** → **HttpServlet** y **HttpServlet** → **HttpServletRequest**, **HttpServlet** → **HttpServletResponse**.

Nombre de la clase	Funcionalidad	Métodos básicos	Ejemplo
HttpServlet	Permite la creación de servlets y la creación de hilos para atender las peticiones de los clientes	void init( ServletConfig config);	Inicializa el servlet. El servlet se ejecuta una única vez, en cuanto el servidor arranca o cuando es invocado por un cliente por primera vez.
		void service( HttpServletRequest req, HttpServletResponse res);	void service( HttpServletRequest req, HttpServletResponse res) { //...}  <i>Rutina de gestión de cada llamada. Cada vez que se llame al servlet se ejecutará el código que este método indique. El servidor mantiene, asociada a cada llamada, una referencia para gestionar los datos entrantes que mande el cliente (res) y otra para gestionar los datos salientes. ¡¡IMPORTANTE!! Cada llamada se gestiona con un hilo, no con un proceso específico (proceso hay uno y tiene las propiedades adoptadas según init)</i>
HttpServletRequest	Permite manejar los datos que el cliente está enviando al servidor	<b>Estando declarada una referencia del tipo HttpServletRequest</b> void service( <u>HttpServletRequest</u> req, HttpServletResponse res) { //...}	
		public String getParameter( String parametro);	String edad = req.getParameter("EDAD");  <i>La referencia req contiene cadenas del tipo param1=valor1&amp;param2=valor2&amp;... En este caso se elegiría el valor que almacene "EDAD"</i>
HttpServletResponse	Permite manejar los datos que el servidor está enviando al cliente	<b>Estando declarada una referencia del tipo HttpServletResponse</b> void service( HttpServletRequest req, <u>HttpServletResponse</u> res) { //...}	
		void setContentType( String type);	res.setContentType( "text/html");  <i>Establece el tipo de contenido MIME que tendrá la respuesta http generada por el servlet. Recordemos que http es un protocolo sin estados, en el que el cliente realiza una petición y el servidor responde SIEMPRE con un mensaje MIME.</i>
		PrintWriter getWriter();	PrintWriter flujoSalida = new PrintWriter( res.getWiter()); flujoSalida.println("<HTML><BODY>Hola! </HTML></BODY>");  <i>Obtiene un flujo de salida para generar la respuesta http en modo texto</i>

		<pre>OutputStream getOutputStream();</pre>	<pre>OutputStream flujoSalida = new OutputStream(     res.getOutputStream()); flujoSalida.println("&lt;HTML&gt;&lt;BODY&gt;Hola! &lt;/HTML&gt;&lt;/BODY&gt;");</pre> <p><i>Obtiene un flujo de salida para generar la respuesta http en modo texto</i></p>
ResultSet	<p>Los objetos de esta clase contienen los datos que la 'orden' haya conseguido extraer de la base de datos. Es sobre estos objetos sobre los cuales ejecutamos órdenes de lectura de información</p>	<p><b>Estando declarada una referencia del tipo ResultSet</b>                  ResultSet rs = orden.executeQuery( "SELECT * ...");</p>	
		<pre>boolean next();</pre>	<pre>rs.next();</pre> <p><i>Indica si hay más filas que recorrer en el ResultSet rs</i></p>
		<pre>String getString( int numClomuna); Long getLong( int numColumna); ... Xxx getXxx( int numColumna);</pre>	<pre>Xxx valorLeido = rs.getXxx( int numColumna);</pre> <p><i>Devuelve el valor (de tipo Xxx) que se encuentre en la intersección entre la fila a la que actualmente se esté apuntando en el ResultSet y numColumna</i></p>
		<pre>void close();</pre>	<pre>rs.close();</pre> <p><i>"Vacía" la consulta. ¡¡¡IMPORTANTE!! Entre dos consultas, hay que cerrar siempre el rs, es decir: rs.executeQuery( SQL1) → rs.close(); → rs.executeQuery( SQL2);</i></p>

**Comunicacions d'empresa ASP**

**JAVIER MOYA i PAYÀ**  
**ETSITelecomunicacions**  
**UPV2005**

En un entorno **ASP** (Active Server Pages) y utilizando **VBScript** (Visual Basic Script) como sintaxis para los scripts ASP, la interacción entre el servidor y la base de datos necesita de un controlador específico llamado **ADO** (Active X Data Object). ADO contiene dos clases principales: **Connection** y **RecordSet**. Los objetos<sup>1</sup> que se creen y sus métodos permitirán la sustracción de información que podrá ser devuelta al cliente remoto.

Nombre de la clase	Funcionalidad	Métodos básicos / Propiedades básicas		Ejemplo
Connection	<p>Nos proporciona una conexión a una base de datos ODBC desde una página ASP. Esta conexión nos permitirá efectuar las operaciones que deseemos sobre la base de datos.</p> <p>Es el objeto primario de ADO, ninguno de los otros objetos puede existir si este no es declarado de forma explícita o implícita (en algunos de los ejemplos veremos que no existe una declaración del objeto Connection, pero debemos de tener en cuenta que siempre existe, si es necesario ADO lo declarará por sí mismo).</p> <p>La conexión terminará cuando nosotros la cerremos explícitamente con el método close o bien cuando termine la ejecución de la página ASP</p>	Estando residente en el sistema el objeto Server		<p>Set conexion = Server.CreateObject( "ADODB.Connection")</p> <p><i>El objeto Server crea el objeto conexion que le permite utilizar los métodos adecuados para manejar la conexión con la base de datos</i></p>
		Propiedades	ConnectionString	<p>conexion.ConnectionString="DSN=dsn; user=usr; pass=pwd"</p> <p><i>Establece la cadena anterior para poder identificar el origen de datos</i></p>
			Mode	<p>conexion.Mode=1</p> <p><i>Habilita un control de acceso a nivel de conexión: 1 = solo lectura, 2 = solo escritura, 3 = lectura/escritura</i></p>
		Métodos	Open( "DSN=NombreBD; user=usuario; pass=pwd", mode)	<p>conexion.Open("DSN=Libros; user=javiMoya; pass=jeje", 1)</p> <p><i>Abre una base de datos identificada por el DSN y con unos permisos de acceso estipulados (cuando proceda)</i></p>
BeginTrans, CommitTrans, RollbackTrans	<p>conexion.BeginTrans</p> <p><i>Activa el modo transaccional para cierta cantidad de operaciones transaccionales</i></p> <p>conexion.CommitTrans</p> <p><i>Realiza las operaciones de forma atómica y cierra automáticamente el modo transaccional</i></p> <p>conexion.RollBackTrans</p> <p><i>Deshace las operaciones antes del último lote transaccional</i></p>			

<sup>1</sup> Haciendo un símil con Java, en el que la creación de un objeto sigue la siguiente sintaxis Clase referencia = new Clase( parametrosConstructor), en VBScript se crean objetos de la siguiente forma: Set referencia = Server.CreateObject("ADODB.ClaseControlador"); donde "referencia es el objeto" del tipo ClaseControlador, ADODB es el controlador, ClaseControlador es el nombre de la clase en el controlador y Server indica que el objeto que se crea pertenece a su vez al objeto Server (ver objetos VBScript)

			Close	<p>conexion.Close</p> <p><i>Cierra la conexión con la base de datos</i></p>
RecordSet	<p>Representa una tabla o el resultado de una consulta ejecutada contra la base de datos. Va a ser nuestro interface natural contra la base de datos. Como en todo modelo relacional, los datos se nos presentaran en filas y columnas.</p>	Propiedades	Estando residente en el sistema el objeto Server	<p>Set rs = Server.CreateObject("ADODB.RecordSet")</p> <p><i>El objeto Server crea el objeto rs que le permite utilizar los métodos adecuados para extraer información de la base de datos</i></p>
			CursorType	<p><i>Establece el vínculo de acceso entre el RecordSet y la base de datos. Jerárquicamente por debajo de si la conexión es del tipo lectura, escritura o lectura/escritura</i></p> <p>rs.CursorType=0</p> <p><i>(Valor por defecto) Sólo se permite leer. La lectura es secuencial (un solo sentido). Si se actualizan datos en la BD, estos NO se reflejan en el RecordSet. Equivale al ResultSet de Java</i></p> <p>rs.CursorType=1</p> <p><i>Este cursor permite lectura en ambos sentidos y actualizaciones. Las actualizaciones que se realicen sobre el RecordSet se actualizaran en la BD pero el ResultSet no se actualizará si la BD cambia</i></p> <p>rs.CursorType=2</p> <p><i>Este cursor permite lectura en ambos sentidos y actualizaciones. Las actualizaciones que se realicen sobre el RecordSet se actualizaran en la BD y el ResultSet TAMBIÉN se actualizará si la BD cambia</i></p> <p>rs.CursorType=3</p> <p><i>Este cursor permite lectura en ambos sentidos pero ningún tipo de actualización</i></p>
			LockType	<p><i>Se utiliza para evitar que dos usuarios intenten escribir a la vez en la BD. Garantiza la exclusión mutua</i></p> <p>rs.LockType=1</p>

*(Valor por defecto) El RecordSet no puede bloquear la BD, y por tanto no puede modificarla*

			<p>rs.LockType=2</p> <p><i>Bloqueo exclusivo. Se bloquean en exclusividad los registros que se hayan capturado en el RecordSet, pero no el resto. ¿?????? Se utiliza para operaciones del tipo x = x+1</i></p> <p>rs.LockType=2</p> <p><i>Bloqueo cuando Update. La BD solo se bloquea cuando se ejecuta Update. Se pueden modificar registros en modo no bloqueante; cuando "se carguen" con Update los cambios, entonces la BD se bloqueará EL TIEMPO JUSTO</i></p> <p><b>Combinaciones CursorType/LockType</b>                  Lectura 0,1                  Lectura y acceso a RecordCount 3,1                  Modificaciones (no inserciones) 1,3                  Inserciones y modificaciones 2,3                  Operaciones Tipo (x=x+1) 1,2</p>
			<p>EOF</p> <p>rs.EOF</p> <p><i>Vale true si estamos en el último registro del RecordSet</i></p>
			<p>BOF</p> <p>rs.EOF</p> <p><i>Vale true si estamos en el primer registro del RecordSet</i></p>
			<p>RecordCount</p> <p>numRegistros=rs.RecordCount</p> <p><i>Devuelve el número de registros que tenemos en el ResultSet</i></p>
		Métodos	<p>Open instruccionSQL, conexion</p> <p>rs.Open "SELECT * FROM Tabla", conexion</p> <p>Open instruccionSQL, conexion, CursorType, LockType</p> <p>rs.Open "SELECT * FROM Tabla", conexion, 0, 1</p> <p><i>Genera un RecordSet rs que contiene el resultado de la consulta SQL según la instruccionSQL</i></p>
			<p><i>Lectura de un registro en la BD</i></p> <p>var=rs("NombreColumna")</p> <p><i>Escritura de un registro en la BD</i></p> <p>rs("NombreColumna")=valor</p>

Mètodes	<p>Move</p> <p>MoveFirst MoveLast MoveNext MovePrevious</p>	<p>rs.Move=5 rs.Move=-2</p> <p><i>Indica un desplazamiento en número de filas dentro de rs. Para el segundo caso el CursorType tiene que permitir lectura en ambos sentidos (CursorType={ 1,2})</i></p> <p>rs.MoveFirst, rs.MoveLast, rs.MoveNext, rs.MovePrevious</p> <p><i>Desplazan la posición del cursor según su nombre indica</i></p>
	<p>Close</p>	<p>rs.Close</p> <p><i>Cierra el RecordSet. ¡¡IMPORTANTE!! Hay que cerrar el RecordSet entre consultas SQL distintas (rs.Open SQL1 ... → rs.Close → rs.Open SQL2 ...)</i></p>
	<p>AddNew</p>	<p>rs.AddNew</p> <p><i>Desplaza el cursor del RecordSet hasta el registro de inserción. ¡¡IMPORTANTE!! Los datos que se introducen en el registro de inserción se actualizan en la BD, pero solo si se utiliza Update (antes de mover el cursor)</i></p>
	<p>Update</p>	<p>rs.Update</p> <p><i>Escribe los campos referidos al registro activo, ya sea en el registro de inserción (introducción de datos) o cualquier otro registro (modificación de datos)</i></p>
	<p>Delete</p>	<p>rs.Delete</p> <p><i>Elimina el registro activo</i></p>

El servidor www/SQL dispone residentes en el sistema de los objetos **Request, Response, Server, Application y Session**<sup>1</sup>. Los objetos Request y Response disponen de los métodos apropiados para intercambiar datos con el cliente; Server, Application y Response son el punto de interacción entre las órdenes del cliente y lo que el servidor ejecuta sobre la base de datos.

Nombre de la clase	Funcionalidad	Métodos básicos / Propiedades básicas		Ejemplo
Request	El objeto Request se utiliza para tener acceso a la información que se pasa en las peticiones HTTP. Entre dicha información se incluyen los parámetros que se pasan desde los formularios HTML mediante el método POST o el método GET	Métodos	Form("campo")	<p>valor=Request.Form("nombre")</p> <p><i>Si el cliente ha utilizado un formulario con METHOD=POST, este método sustrae el VALUE asociado al campo NAME=nombre</i></p>
			QueryString("campo")	<p>valor=QueryString.Form("nombre")</p> <p><i>Si el cliente ha utilizado un formulario con METHOD=GET (u omitido), este método sustrae el VALUE asociado al campo NAME=nombre</i></p>
		Propiedades	ServerVariables("variableEntorno")	<p>Var=Request.ServerVariables("REMOTE_ADDR")</p> <p><i>Obtiene la variable de entorno REMOTE_ADDR, que contiene la IP del servidor</i></p>
Response	El objeto Response se utiliza para controlar la información que se envía al usuario. Esto incluye el envío de información directamente al explorador, la redirección del explorador a otra dirección URL o el establecimiento de valores de las cookies	Propiedades	ContentType	<p>Response.ContentType="text/html"</p> <p><i>Establece que lo que se va a enviar al cliente son cadenas de texto del tipo html</i></p>
			Expires ExpiresAbsolute(#month day, year#)	<p>Response.Expires=maxMinutos</p> <p><i>Indica el máximo de tiempo que la página permanecerá en la caché del cliente. Response.Expires=0 obliga a que el cliente consulte el servidor cada vez y que no acceda a su caché (frescura de la información)</i></p> <p>Response.ExpiresAbsolute(#May 10, 2005#)</p>
		Métodos	Write("texto")	<p>Response.Write &lt;OPTION VALUE="&amp;rs("nombreAlumno)&amp;"&gt;</p> <p>Envia la cadena de resultado hacia el cliente, por ejemplo &lt;OPTION VALUE=Javi&gt;</p>

<sup>1</sup> Objetos integrados ASP

		Métodos	<p>Write(variable) ó =variable</p>	<p>Response.Write(var) ó =var <i>Escribe una variable</i></p>
		Métodos	<p>Redirect("URL")</p>	<p>Response.Redirect www.sport.es <i>El cliente se conectará a www.sport.es. Este método funciona únicamente si antes NO se le ha enviado nada al cliente</i></p>
		Métodos	<p>BinaryWrite("archivo")</p>	<p>Response.BinaryWrite img.jpg <i>Envía la imagen jpg hacia el cliente, byte a byte (utilizando un buffer de segmentos TCP)</i></p>
		Métodos	<p>End</p>	<p>Response.End <i>Da por finalizada la respuesta del servidor hacia el cliente</i></p>
<p>Server</p>	<p>El objeto Server proporciona acceso a los métodos y las propiedades del servidor. El método utilizado más frecuentemente es el que crea una instancia de un componente ActiveX (Server.CreateObject)</p>	Propiedades	<p>ScriptTimeout</p>	<p>Server.ScriptTimeout=120 <i>Indica que las operaciones que el servidor tiene que hacer en el script tienen que efectuarse en 120 segundos. Tras 120 segundos el servidor finaliza la ejecución del script. Permite controlar que el servidor no acceda a bucles infinitos</i></p>
		Métodos	<p>CreateObject("IdProg") <i>IdProg es el identificador del tipo de componente que queremos crear; nos viene suministrado por el fabricante del componente</i></p>	<p>Set rs=Server.CreateObject("ADODB.RecordSet") <i>Crea una instancia de un componente ActiveX en el servidor (crea un objeto). En este caso el servidor crea un objeto del tipo RecordSet</i></p>

Application	El objeto Application se utiliza para compartir información entre todos los usuarios de una aplicación	Métodos	Lock/Unlock	<p style="text-align: center;">Application.Lock                  Application("visitas")= Application("visitas") + 1                  Application.Unlock</p> <p style="text-align: center;"><i>Los métodos Lock y Unlock aseguran la exclusión mutua en el acceso a variables de aplicación</i></p>
		Ejemplo de creación de variables de Application	<p>En el objeto Application pueden almacenarse matrices, pero estas son almacenadas como un objeto, es decir, no podemos almacenar o recuperar un solo elemento de la matriz, si no que cargaremos o recuperaremos la variable con la matriz completa</p> <p>Ejemplo</p> <pre style="margin-left: 40px;">                 &lt;%Dim parametros(2)                 parametros(0) = "verde"                 parametros(1) = 640                 parametros(2) = 480                 Application.Lock                 Application("Param") =parametros%&gt;                  Application.UnLock             </pre> <p>con estas instrucciones almacenaríamos TODA la matriz en la variable de aplicación "Param"</p> <p>Para recuperar los valores de la matriz primero recuperamos esta en una variable normal</p> <pre style="margin-left: 40px;">                 &lt;%Apliparam=Application("Param")%&gt;             </pre> <p>Ahora podremos operar con los valores de la tabla en las variables Apliparam(0), Apliparam(1) y Apliparam(2)</p>	

Session	<p>El objeto Session permite almacenar la información necesaria para una determinada sesión de usuario. Las variables almacenadas en el objeto Session no se descartan cuando el usuario pasa de una página a otra dentro de la aplicación, si no que dichas variables persisten durante todo el tiempo que el usuario tiene acceso a las páginas de la aplicación. También puede utilizar los métodos de Session para terminar explícitamente una sesión y establecer el periodo de tiempo de espera de inactividad de las sesiones</p>	Propiedades	IdSession	<p style="text-align: center;">idSesion=Session.IdSession</p> <p>Cada sesión existente en el servidor es identificada internamente mediante un entero. En este caso este valor lo almacenamos en idSesion</p>
		Métodos	Abandon	<p style="text-align: center;">Session.Abandon</p> <p>Destruye todos los objetos y variables asociados a la sesión representada por Session</p>
		Ejemplo de creación de variables de Application	<p>En el objeto Session pueden almacenarse matrices, pero estas son almacenadas como un objeto, es decir, no podemos almacenar o recuperar un solo elemento de la matriz, si no que cargaremos o recuperaremos la variable con la matriz completa</p> <p>Ejemplo</p> <pre style="margin-left: 40px;">                 &lt;%Dim cestacompra(2)                 cestacompra(0) = 1                 cestacompra(1) = 8                 cestacompra(2) = 22                 Session("Cesta") =cestacompra%&gt;             </pre> <p>con estas instrucciones almacenaríamos TODA la matriz en la variable de sesión "Cesta"</p> <p>Para recuperar los valores de la matriz primero recuperamos esta en una variable normal</p> <pre style="margin-left: 40px;">                 &lt;%Micesta=Session("Cesta")%&gt;             </pre> <p>Ahora podremos operar con los valores de la tabla en las variables Micesta(0), Micesta(1) y Micesta(2)</p>	

El archivo Global.asa es un fichero de texto situado en el directorio raiz de nuestro servidor Web, es decir, en el directorio de comienzo de nuestras páginas. Es un archivo de comandos que nos permite la automatización de los cuatro eventos básicos de nuestro servidor. La estructura es siempre la misma:

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">

Sub Application_OnStart
.....
End Sub

Sub Application_OnEnd
.....
End Sub

Sub Session_OnStart
.....
End Sub

Sub Session_OnEnd
.....
End Sub

</SCRIPT>
```

Evento	Funcionalidad	Ejemplo de archivo Global.asa
Application_OnStart	El evento Application_OnStart se ejecuta antes de que se cree la primera nueva sesión; es decir justo cuando el primer cliente pide una pagina de nuestro servidor	<pre>&lt;SCRIPT LANGUAGE="VBScript" RUNAT="Server"&gt;  Sub Application_OnStart dim mitabla() redim mitabla(9) application("tabla")=mitabla End Sub  Sub Application_OnEnd  End Sub  Sub Session_OnStart paginaInicio="/appl/index.html" response.redirect paginaInicio End Sub  Sub Session_OnEnd  End Sub  &lt;/SCRIPT&gt;</pre>
Application_OnEnd	El evento Application_OnEnd se ejecuta cuando la aplicación termina	
Session_OnStart	El evento Session_OnStart se ejecuta cuando el servidor crea una nueva sesión; esta secuencia de comandos es ejecutada antes de enviar la página solicitada al cliente	
Session_OnEnd	El evento Session_OnEnd se ejecuta cuando se abandona o se supera el tiempo de espera de una sesión	