Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

16

Consumiendo Web Services

Encriptación - Introducción a WSE

Seguridad

Amenazas en aplicaciones Web

STRIDE - DREAD

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ISBN 978-987-1347-43-8
00016
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Creación de consultas con parámetros

Parametrizar las consultas es una manera muy rápida de obtener resultados con filtros sin necesidad de crear procedimientos amacenados en el motor. Esto puede lograrse a través de la inserción de parámetros en cadenas de texto.

En el siguiente ejemplo, tomaremos una tabla llamada Usuario, con un campo ID, del tipo numérico, para la identificación única de registros, y utilizaremos un filtro o condición Where en nuestra consulta para el método Fill. A su vez, pasaremos el valor por filtrar a través de una variable, en OleDbParameter:

```
'Codigo Visual Basic
'Utilizaremos como ejemplo el objeto MyConn
creado anteriormente
Public Function ObtenerUsuariosPorId(ByVal Id
as Integer) as DataSet

  Dim MyFilterSelectCommand as String =
 "SELECT * FROM Usuario WHERE ID = ?"
  Dim MyFilterAdapter as new OleDbDataAdapter
  (MyFilterSelectCommand, MyConn)
  MyFilterAdapter.SelectCommand.Parameters.
  Add("@ID",OleDbType.integer,10,Id)
  Dim MyFilterDataSet as DataSet
  MyFilterAdapter.Fill(MyFilterDataSet)
  Return MyFilterDataSet

End Function

//Codigo C#
public DataSet ObtenerUsuariosPorId(int Id) {
        string MyFilterSelectCommand =
        "SELECT * FROM Usuario WHERE ID = ?";
        OleDbDataAdapter MyFilterAdapter =
        new OleDbDataAdapter(MyFilter
        SelectCommand, MyConn);
        MyFilterAdapter.SelectCommand.
        Parameters.Add(ID, OleDbType.integer,
        10, Id);
        DataSet MyFilterDataSet;
        MyFilterAdapter.Fill
        (MyFilterDataSet);
        return MyFilterDataSet;
    }
```

En el código anterior podemos observar cómo, mediante el uso del carácter "?", indicamos el pasaje de parámetros (en caso de existir más de un parámetro, se asignarán por orden de agregado), para luego añadirlo mediante el método Add de la colección de parámetros. Este método reemplaza al equivalente en un procedimiento almacenado con parámetros establecidos desde el motor de base de datos.

Actualizando datos

Si obtenemos datos desde un origen con el objeto DataAdapter, podremos indicar también

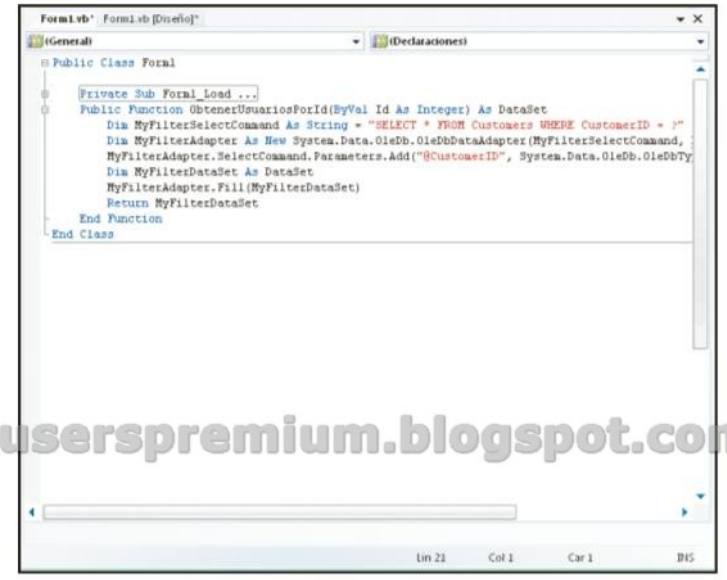


FIGURA 010 | Desarrollo de la función de la consulta mediante parámetros en VB.NET.

El objeto DataAdapter posee lógica embebida para el manejo de errores comunes en la actualización de datos.

qué comportamiento tendrá que realizar al momento de la actualización. Dicha tarea será efectuada mediante el llamado comando de actualización o UpdateCommand del DataAdapter.

Como vimos anteriormente, mediante el uso del carácter "?", podemos indicar la existencia de un parámetro para el comando. Veremos cómo establecer su valor con el dato contenido en una celda específica de nuestra colección en memoria (DataSet), en el siguiente fragmento de código:

```
'Codigo Visual Basic.NET
'Utiizaremos El mismo ejemplo que antes, para
actualizar
Public Function ActualizarUsuario(ByVal Datos
As DataSet)
  Dim MyUpdateCommand as String = "UPDATE
  Usuario SET ID = ?, Nombre = ? WHERE ID = ?"
  Dim MyUpdateAdapter as new OleDbDataAdapter
  (nothing, MyConn)
  MyUpdateAdapter.UpdateCommand.Parameters.
  Add("@IDNuevo", OleDbType.Integer, 10, "ID")
  MyUpdateAdapter.UpdateCommand.
  Parameters.Add("@Nombre", OleDbType.
  VarChar, 40, "Nombre")
  MyUpdateAdapter.UpdateCommand.
  Parameters.Add("@IdOriginal", OleDbType.
  Integer, 5, "ID").SourceVersion =
  DataRowVersion.Original
  MyUpdateAdapter.Update(Datos)
End Function

//Codigo C#
//Utiizaremos El mismo ejemplo que antes,
para actualizar
    public void ActualizarUsuario(DataSet
    Datos) {
        string MyUpdateCommand = "UPDATE
        Usuario SET ID = ?, Nombre = ? WHERE
        ID = ?";
        OleDbDataAdapter MyUpdateAdapter =
        new OleDbDataAdapter(null, MyConn);
        MyUpdateAdapter.UpdateCommand.
        Parameters.Add("@IDNuevo", OleDbType.
        Integer, 10, "ID");
        MyUpdateAdapter.UpdateCommand.
        Parameters.Add("@Nombre", OleDbType.
        VarChar, 40, "Nombre");
        MyUpdateAdapter.UpdateCommand.
        Parameters.Add("@IdOriginal",
        OleDbType.Integer, 5, "ID").
        SourceVersion = DataRowVersion.
        Original;
        MyUpdateAdapter.Update(Datos);
    }
```

FIGURA 011 | Los tips de VB.NET nos alertarán en el caso de desarrollar mal una función.

Debemos prestar especial atención a la propiedad establecida, SourceVersion, que indicará al adaptador qué valor utilizar para la actualización original o el modificado; por defecto, será este último.

Procedimientos almacenados

Pueden sernos de utilidad a la hora de programar lógica de negocio compleja o en caso de necesitar alto rendimiento.

La posibilidad de programar fragmentos de código en un lenguaje particular (PL/SQL, T-SQL, etc.) y acceder a ellos desde nuestra aplicación abre un abanico de posibilidades en cuanto a distribución de la lógica de negocio.

Código almacenado vs. código fuente

A la hora de elegir la ubicación de una función de negocio, entran en juego diversas variables y condiciones, como volumen de datos por procesar, cantidad de tablas relacionadas en la función y complejidad de consulta. Cuanto mayores sean estos valores, más motivos tendremos para ubicar esa función en un procedimiento almacenado en lugar del código (C#, VB.NET, etc.).

Optimización

Cualquier motor de base de datos actual respetable soporta el uso de procedimientos almacenados, y su optimización representa un beneficio significativo de rendimiento frente al uso de consultas dinámicas o generadas por código fuente. Cada motor posee técnicas particulares de optimización, que incluyen la precompilación, cacheo de resultados, simplificación, etc.

Creación de un procedimiento almacenado

La creación de un procedimiento almacenado es una tarea relativamente sencilla, pero un tanto incómoda desde el punto de vista del programador. Para hacerlo, es necesario correr una consulta de creación/modificación (en caso de que el procedimiento ya exista), con la definición del procedimiento almacenado, y declarar un nombre particular para él, sus correspondientes parámetros (de entrada y salida) y el cuerpo propiamente dicho, como si se tratase de cualquier función en algún lenguaje estructurado. El siguiente código T-SQL crea un procedimiento almacenado hacia una tabla imaginaria, recibiendo un parámetro para filtrar datos (ID) y devolviendo las columnas específicas que precisamos mostrar en pantalla:

```
CREATE/ALTER Procedure [Usuario_Select]
@Id int = null
AS
 SELECT Nombre, Apellido, Direccion, Telefono
 FROM
  [Usuario]
 WHERE Id=@Id
```

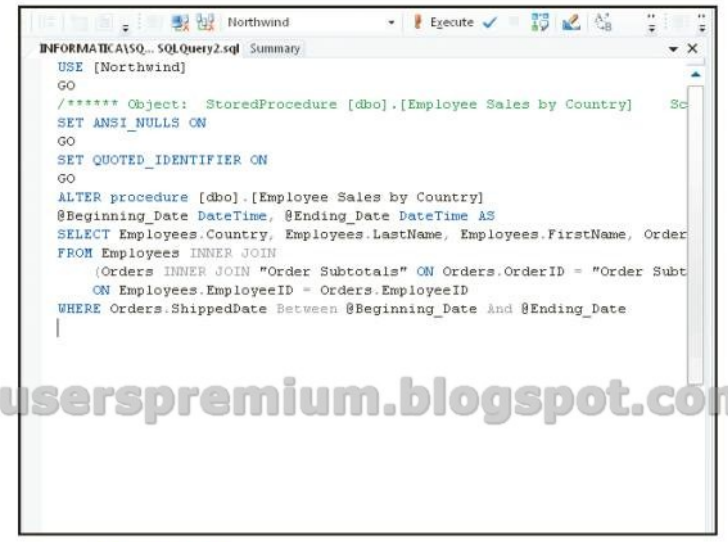


FIGURA 012 | Stored procedure editado de la base de datos Northwind en SQL Server 2005.

Capa de acceso a datos

La capa de acceso a datos cumple un rol importante en todo desarrollo o software. Veamos cuáles son sus funciones.

Comúnmente conocida como *Data Access Layer*, esta capa debe encapsular todas las funciones de acceso a datos en nuestro sistema o desarrollo, y automatizar todo lo posible la interacción con cualquier base o motor.

Capas lógicas, capas físicas

Al hablar de capas en proyectos de software, siempre se generan confusiones, sobre todo, debido a las diversas acepciones que posee el término y a sus distintas traducciones del inglés (Layer, Tier, etc.). Para este caso, diferenciaremos la capa física de acceso a datos (puede ser un servidor Web), de la capa donde se encuentra físicamente el servidor de base de datos. Como es lógico, estas capas pueden estar ubicadas en un mismo servidor o computadora. También vamos a distinguir la capa lógica de acceso a datos del proyecto Web en sí, creando un proyecto separado de la interfaz de usuario.
El objetivo de nuestra capa es reducir la cantidad de código necesaria para acceder al motor, identificando aquella funcionalidad invariable, y parametrizando los puntos que varían entre una consulta y otra. Todo el código aquí desarrollado debería ser reutilizable desde cualquier plataforma (Windows, Mobile, etc.) y, a su vez, permitirnos el intercambio del motor de base de datos sin recompilar las capas superiores.
Para agregar la capa de acceso a datos comenzaremos por incluir un nuevo proyecto en nuestra solución de ejemplo, desde Agregar Nuevo Proyecto, en el menú Archivo.
Una vez en el asistente, seleccionamos el tipo Librería de Clases y colocamos el nombre DAL (por sus siglas en inglés).

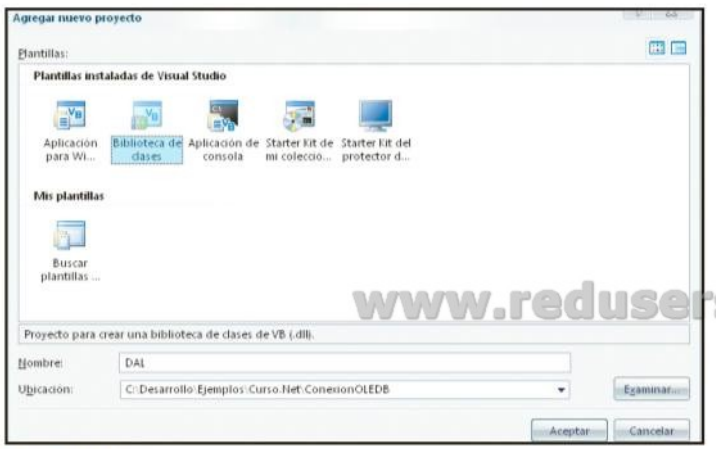


FIGURA 013 | Agregando la librería de clases a nuestro proyecto ConexionOleDB en VB.NET.

Generar las clases

Nuestro objetivo, como ya mencionamos, es la creación de una o más clases reutilizables para el acceso al motor de base de datos. La siguiente estructura de clase nos permitirá traer valores, ejecutar procedimientos almacenados, cargar objetos en memoria con los datos y actualizar/insertar en el motor, desde un único punto:

```
'Codigo Visual Basic
Public Class DataAccess
    Private _connectionString As String
    Public Enum Tablas
        Usuarios
        Productos
        Paises
    End Enum
    Public Sub New()
        _connectionString = System.
```

```
            Configuration.ConfigurationManager.
            AppSettings("ConnectionString")
        End Sub
Public Function Ejecutar(ByVal NombreComando
As String) As DataTable
End Function

        Public Function Ejecutar(ByVal Nombre
        Comando As String, Parametros as List(Of
        SqlClient.SqlParameter) As DataTable

End Function

        Public Function Cargar(ByVal NombreDeTabla
        As Tablas, ByVal Obj As DataSet)

End Function

        Public Function Actualizar(ByVal Datos As
        DataSet, ByVal tabla As Tablas)

End Function

        Public Function Agregar(ByVal Datos As
        DataRow, ByVal tabla As Tablas)

End Function

End Class


//Codigo C#
public class DataAccess {

    private string _connectionString;

    public DataAccess() {
        _connectionString = System.
         Configuration.ConfigurationManager.
         AppSettings("ConnectionString");
    }
    public DataTable Ejecutar(string
    NombreComando) {
    }

    public DataTable Ejecutar(string
    NombreComando, List[] Parametros, void
    Of, void SqlClient.SqlParameter) {
    }
    public void Cargar(Tablas NombreDeTabla,
    DataSet Obj) {
    }
     public void Actualizar(DataSet Datos,
    Tablas tabla) {
    }
    public void Agregar(DataRow Datos, Tablas
    tabla) {
    }
     public enum Tablas {
        Usuarios,
        Productos,
        Paises,
    }
}
```

En el código anterior, podemos ver el uso de enumeraciones para la diferenciación de tablas, recurso que también puede ser utilizado para la identificación inequívoca de procedimientos almacenados.

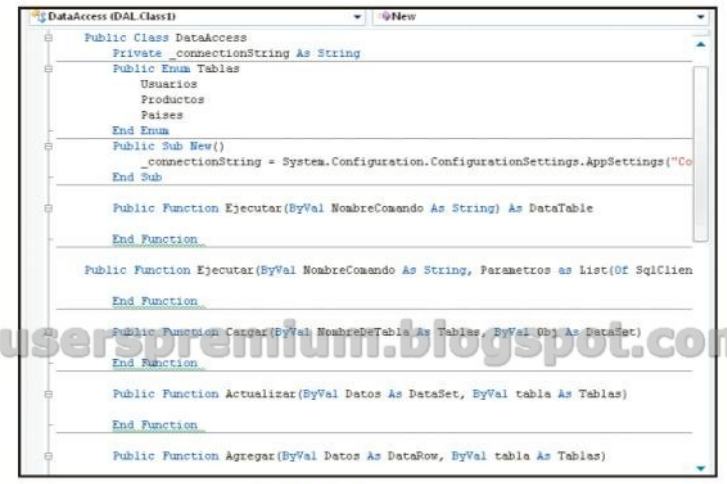


FIGURA 014 | Así creamos las clases necesarias para la reutilización de código.

Asegurar la base de datos

Debemos comprender el concepto de seguridad para todas nuestras aplicaciones, más aún cuando utilizan bases de datos.

Ya sea en una aplicación personal, un sistema a medida o un producto para una corporación, la seguridad de la base de datos juega un papel clave en estos tiempos, en los que el robo de información es moneda corriente y las herramientas de piratería informática son más avanzadas que nunca.

Restricción de usuarios

Es común utilizar el superusuario o administrador del sistema "sa" para conectar aplicaciones a bases de datos, sobre todo, por el hecho de que el usuario siempre tendrá acceso a cualquier base/tabla/vista y no hace falta crearlo. Esta solución parece un camino fácil y rápido para acceder a nuestro motor. Sin embargo, este tipo de error hace completamente vulnerable al sistema ante cualquier ataque o inyección de código.

FIGURA 015 | Uso de la autenticación de nuestro propio usuario en Management Studio.

Crear SP

Una de las mejores maneras de restringir el acceso a nuestros datos es mediante procedimientos almacenados. Es preferible permitir a la aplicación ejecutar ciertos procedimientos con parámetros variables, antes que permitir la lectura de todas las tablas del sistema. Esto puede observarse con más detalle en la sección relativa a los procedimientos almacenados y su creación.

Crear usuarios

Utilizar un usuario específico para consultas desde código fuente nos permite manejar con detalle el nivel de acceso a las tablas del sistema, y determinar qué objetos serán visibles, qué operaciones será posible ejecutar, y denegar, sobre todo, la modificación de tablas con datos sensibles.

Buenas prácticas

Analicemos algunas prácticas comunes referidas al uso de aplicaciones con base de datos. Estos lineamientos no deben tomarse como reglas obligatorias de programación, sino como sugerencias aplicables a nuestro proyecto.

Nombres de Stored Procedures

Es importante mantener un orden adecuado con nuestros procedimientos almacenados en el motor de base de datos. El uso de prefijos

permite encontrar fácilmente los SP relacionados entre sí, y es indispensable si el volumen de éstos supera los 20 o 30. Podremos agruparlos por tipo de operación, con prefijos del estilo SELECT_, DELETE_, UPDATE_, etc.; o bien por unidades de negocio, al estilo de CLIENTES_, EMPLEADOS_, etc.

> Debemos evitar a toda costa el uso de prefijos reservados por el sistema (SP_).

Optimizar consultas

Cada sentencia de una consulta influye notablemente en el tiempo de ejecución. Por cada operación que dejemos librada al motor se sumarán milésimas de segundos, que, en cantidad, pueden resultar un tiempo considerable para el usuario final. Una manera práctica de quitarle trabajo al servidor de base de datos al hacer consultas (SELECT) es indicar, específicamente, la o las columnas por traer. Es decir, en vez de ejecutar la línea:

```
SELECT * FROM Usuario;
```

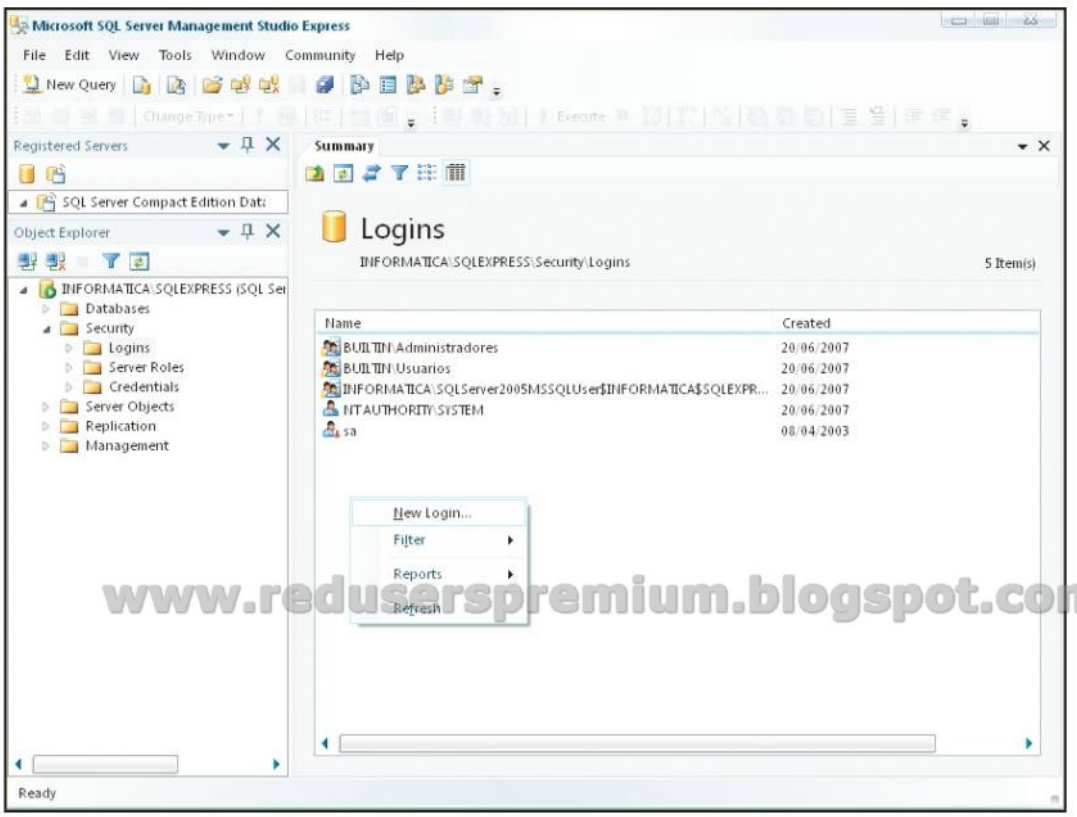
Conviene incluir cada uno de los nombres de columna presentes en la tabla, y si bien esto lleva una cantidad mayor de tiempo como programadores, simplifica gran parte del trabajo del motor a la hora de analizar y ejecutar la consulta:

```
SELECT Nombre, Apellido, Direccion, Telefono
FROM Usuario;
```

Cadena de conexión en Web.Config

En aplicaciones Web es frecuente utilizar el archivo Web.Config para almacenar opciones

FIGURA 016 | Agregando un nuevo usuario en SQL Management Studio 2005 Express.



de sólo lectura, en formato Clave-Valor. Esta capacidad nos permite establecer la cadena de conexión a nuestro motor de base de datos como un parámetro de la aplicación Web, y acceder al valor desde la capa de acceso a datos, por estar en el mismo espacio de memoria. El siguiente fragmento de código representa un archivo Web.Config estándar, con un parámetro establecido para la cadena de conexión:

```
<?xml version="1.0"?>
<configuration>
   <appSettings>
   <add key="CadenaDeConexion"value="Provider=
   SQLOLEDB;Data Source=localhost;Initial
   Catalog=MiBaseDePruebas;Integrated
   Security=SSPI;"/>
   </appSettings>
      <connectionStrings/>
      <system.web>
      </system.web>
</configuration>
```

FIGURA 017 | Conexión a SQL Server dentro del archivo web.config.

§ Nueva sección en Web.Config

ASP.NET 2.0 ofrece una sección exclusiva dentro del archivo Web.Config para el almacenamiento de cadenas de conexión a fuentes de datos, llamada ConnectionStrings. Es sólo una subdivisión de la categoría Settings, y es accesible a través de una propiedad específica de la clase Configuration Manager.

Acceso al archivo Web.Config

Existen diversos mecanismos para acceder a los valores almacenados en el archivo Web.Config. Las siguientes líneas de código nos devolverá el valor almacenado en el archivo de configuración, desde cualquier librería referenciada desde el proyecto Web. En este caso, la capa de acceso a datos será la encargada de acceder al valor y de instanciar un objeto Connection con dicha cadena. En las versiones 1.0 y 1.1 del framework .NET, la clase utilizada para esta tarea era la llamada AppSettings, ubicada en el espacio de nombres System.Configuration. ConfigurationSettings. Sin embargo, con la llegada del framework .NET 2.0, se simplificó el acceso a todos los parámetros de la aplicación dentro de la clase Configuration Manager:

```
'Codigo Visual Basic.NET
Dim miCadena as String = System.Configuration.
ConfigurationManager.AppSettings("CadenaDe
Conexion")
```



```
'Codigo C#
String miCadena = System.Configuration.
ConfigurationManager.AppSettings("CadenaDe
Conexion");
```

Web Services

Creación de aplicaciones distribuidas

9

Contenidos

En este capítulo aprenderemos todo lo necesario para trabajar con servicios Web: desde los fundamentos y su infraestructura, pasando por los estándares que se utilizan hasta ejemplos prácticos para crear, publicar y consumir servicios Web.

Temas tratados

- Fundamentos
- Infraestructura
- WSDL y UDDI
- Creando un Web Service
- Publicación y testeo de Web Services
- Consumir un Web Service
- Encriptación
- Introducción a WSE

Web Services

Los Web Services son una tecnología orientada a servicios, lo que nos permite crear aplicaciones distribuidas.

Fundamentos

Comenzaremos aprendiendo qué es y cómo se usa un servicio Web. Conoceremos además las tecnologías involucradas.

- Qué es un Web Service
- Qué estándares maneja
- Cómo consumirlo
- Navegando un Web Service

Infraestructura

Aprenderemos con más detalle las tecnologías que se utilizan e interactúan durante el consumo de un servicio Web.

- Las tecnologías que más se utilizan
- Entidades involucradas
- Roles
- Aspectos y normas por cumplir

WSDL y UDDI

Conoceremos en detalle cómo encontrar un Web Service y reconocer sus métodos, parámetros y datos devueltos mediante la lectura de documentos XML.

- Significado de WSDL
- Partes de un documento WSDL
- Significado de UDDI

Creación de un Web Service

En este punto, repasaremos lo visto hasta aquí, y aprenderemos todo lo necesario para crear un servicio Web.

- Repasos generales
- Estructura de código
- Atributos de clase y métodos
- Web Services transaccionales

Publicación y testeo de Web Services

Conoceremos las formas y métodos prácticos para publicar y probar nuestros servicios web, como así también consideraciones a tener en cuenta cuando realizamos pruebas.

- Proceso de publicación
- Proceso de prueba
- Prueba final

Fundamentos

A continuación, conoceremos desde cero para qué sirven y cómo aplicar Web Services.

Lo primero que hay que conocer es para qué sirve un Web Service, y podemos notar dos escenarios bastante comunes hoy en día: integrar aplicaciones realizando tareas remotas y (el más simple) compartir información.
Las ventajas que éstos suponen a la hora de compartir información es que no requieren intervención de un usuario, lo que aumenta la velocidad de proceso o de respuesta.
Se manejan diferentes estándares, como HTTP, SMTP, XML y SOAP, los cuales hemos ido viendo a lo largo de esta obra. La forma en que se consume un Web Service es muy sencilla y requiere de algunos pasos: primero el cliente consulta al servidor Web con el fin de conocer la ubicación de dicho servicio, luego lo busca, solicita la descripción WSDL y, por último, lo consume. Esta descripción WSDL es la que especifica qué datos recibe y qué devuelve. Ahora bien, para consumirlo, está claro que nos debemos comunicar con él, y esto se logra por medio de HTTP-GET, HTTP-POST y SOAP (*Simple Object Access Protocol*), que permite la comunicación utilizando XML, seguridad, etc.
Un Web Service se crea dentro de un directorio Web, pero no necesita estar "atado" a una aplicación Web. Es independiente de ella, y no es más que una clase con una extensión especial ASMX (*Active Server Message eXtension*).
Dentro de esta clase, se definen propiedades, métodos, etcétera, como en cualquier otra, pero los métodos que desean ser publicados deben marcarse con el atributo WebMethod. Por

Consumo de un servicio Web

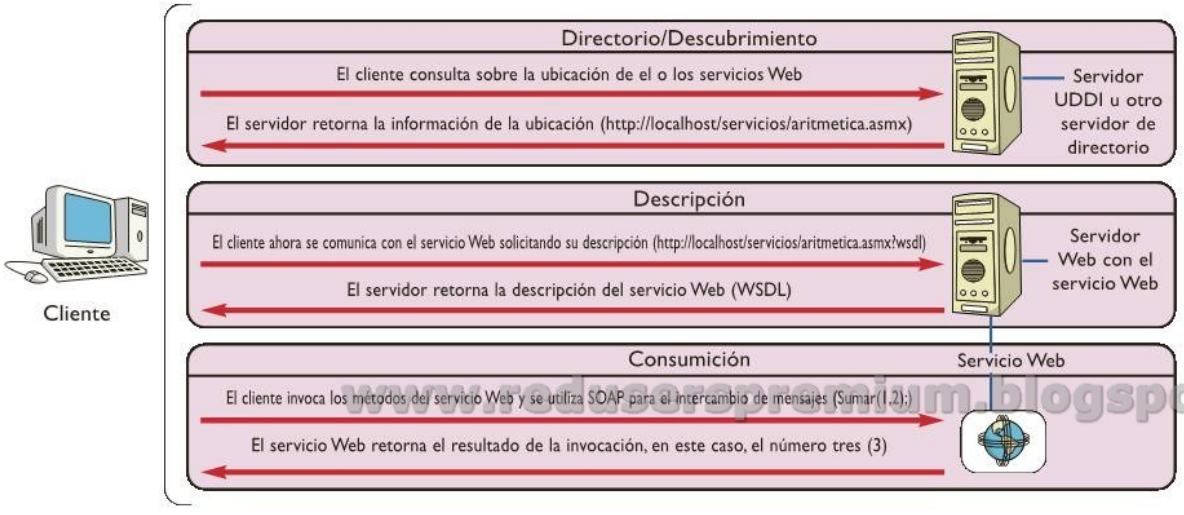


FIGURA 001 | Debemos seguir ciertos pasos necesarios para realizar la operación de forma correcta.

Para consumir un Web Service está claro que nos debemos comunicar con él. Esto se logra por medio de HTTP-GET, HTTP-POST y, por último, SOAP.

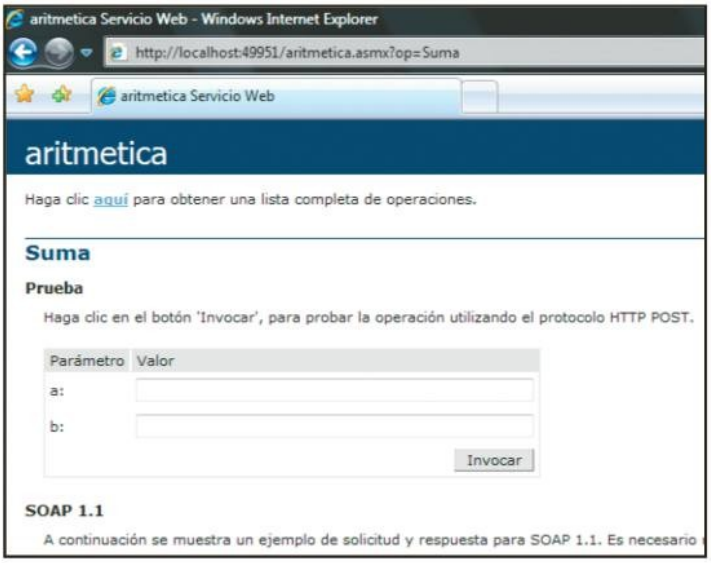


FIGURA 002 | Cuando probamos un servicio web usando un navegador, .NET muestra la descripción de este.

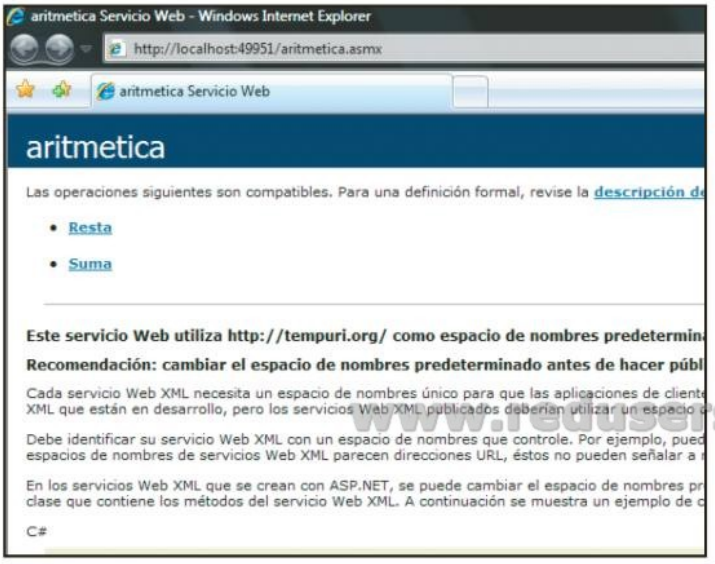


FIGURA 003 | Cuando visualizamos un método, .NET crea una "mini aplicación", a través de la cual podremos probarlo.

ahora no vamos a ver cómo se codifica un Web Service, pero sí cómo se ve su exposición a través de la Web. El caso que vamos a analizar es el de un servicio Web denominado aritmética, que expone dos métodos, uno para sumar y otro para restar.

Cuando intentamos navegar un Web Service, lo primero que notamos es que muestra la información de cuáles son sus métodos expuestos, y un enlace a la definición formal o descripción del servicio, que no es otra cosa que un documento XML. Por otro lado, podemos consumir este servicio desde el mismo explorador haciendo clic en el método que nos interesa. Cuando hayamos entrado en él, veremos que en cuadros de texto nos pide el valor de cada parámetro que requiera (en caso de que los tenga), y un botón con la leyenda Invocar. No tenemos más que ingresar nuestros valores y presionar dicho botón para que se abra una nueva ventana del navegador con un XML que tiene el resultado obtenido. Para el ejemplo del servicio Web de suma y resta (de enteros), si cargamos los valores 1 y 2, nos devolverá un XML como el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">3</int>
```

En este XML veremos que nos retorna un int (recordemos que así puede codificarlo) y el valor 3. Estos tipos devueltos podrían ser objetos, documentos XML, imágenes, etc.

Infraestructura

La primera de estas tecnologías es **XML** y es la que se utiliza durante todo el proceso. Ya sabemos que **XML es un lenguaje de marcas extensibles** (*eXtensible Markup Language*), y que es un medio para almacenar y transportar información en forma estructurada, cuyo estándar está supervisado por la **W3C**.

Luego tenemos **UDDI** (*Universal Description, Discovery and Integration*), que es un servicio mediante el cual se pueden publicar y/o describir los servicios Web. Ya hemos visto que al explorar un servicio Web, también podemos acceder a su descripción. Por lo tanto, **UDDI** nos proporciona una forma de búsqueda y publicación.
A continuación, está **DISCO** (diminutivo de Discovery), que nos brinda la posibilidad de hallar un servicio Web dentro de un sitio.
Para concluir, vemos que el medio o protocolo sobre el cual se comunica el Web Service es **SOAP** (*Simple Object Access Protocol*), que no es más que un soporte liviano para el transporte de la información entre entornos distribuidos y descentralizados, también manejado por el **W3C**.
Con el propósito de ampliar estos conocimientos, vamos a desarrollar el ciclo de vida de una consulta a un Web Service y las partes que intervienen en el proceso.
Podemos citar tres entidades principales, como el cliente del servicio Web, el servicio de directorios (UDDI) y el propio servicio Web. Lo primero que realiza el cliente es consultar al servicio de directorios con el fin de localizar el servicio Web deseado, y recibe un enlace al **Discovery Document**. Con este enlace no hace otra cosa que solicitar dicho documento, pero directamente al servicio Web (el enlace le proporciona el trayecto a él). En este momento, el cliente dispone de este documento, que está en formato XML. A continuación, por medio del WSDL, solicita la descripción del servicio, también en formato XML. Con todo esto, el cliente se encuentra en condiciones de consumirlo llamando al método deseado, y de obtener la respuesta correspondiente a dicha solicitud.
Como ya sabemos, hoy en día la tecnología Web no se basa sólo en Internet, sino que son cada vez más las empresas que la aplican a sus propias intranets. En este aspecto, los servicios Web cumplen un rol de infraestructura muy importante, ya que permiten que la empresa soporte un entorno de negocios colaborativo (compartiendo la capa de negocios), y variadas tecnologías y lenguajes de desarrollo. Tengamos en cuenta, también, que un servicio Web no sólo es accesible mediante una aplicación Web, sino que una Windows puede consumirlo libremente, lo que le da una mayor importancia a esta clase de servicios. Y, por supuesto, puede ser consumido desde otro servicio Web.

Tecnologías

Directory: Publish & Find Services:	UDDI
Inspection: Find Services on server:	DISCO
Description: Format Service Descriptions:	WSDL
Wire Format: Service Interactions:	SOAP
Universal Data Format:	XML
Communications:	Internet

FIGURA 004 | Tecnologías utilizadas durante el consumo de un Web Service.

Para ampliar el concepto de cómo trabaja un servicio Web y en qué medios existe, debemos mencionar el término **SOA** (*Service Oriented Architecture*), que define una arquitectura de desarrollo o trabajo orientada a servicios. Podemos decir que los servicios Web y SOA en Internet van de la mano, ya que los primeros son los elegidos para resolver este tipo de arquitectura. Es más, SOA fue creciendo y cobrando importancia a medida que la tecnología de los servicios Web fue madurando. Hoy en día, los Web Services son el estandarte de SOA.

Los servicios Web, como toda la tecnología orientada a servicios, debe pretender cumplir con ciertos aspectos o normas, como **reusabilidad, contrato formal, bajo acoplamiento, posibilidad de composición, autonomía, carencia de estados y posibilidad de descubrirlo o encontrarlo**. Veamos esto en detalle.

- La **reusabilidad** se basa en la posibilidad de utilizar el mismo proceso de negocios dentro de la misma aplicación o diferentes aplicaciones. A modo de ejemplo, podemos mencionar que si calculamos el sueldo de un empleado con cinco aplicaciones diferentes, todas deben invocar al mismo proceso. Esto garantiza que cuando dicho cálculo deba ser modificado, en forma inmediata todas nuestras aplicaciones verán el reflejo del cambio.
- El hecho de proporcionar un **contrato formal** queda definido en poseer o brindar un nombre de servicio, una forma determinada de acceso, las funciones ofrecidas, y los parámetros y valores devueltos de cada una de dichas funciones. Esto se logra mediante WSDL.
- El **bajo acoplamiento** se basa en la independencia para que el contrato se maneje entre el consumidor y el servicio.

Ciclo de vida

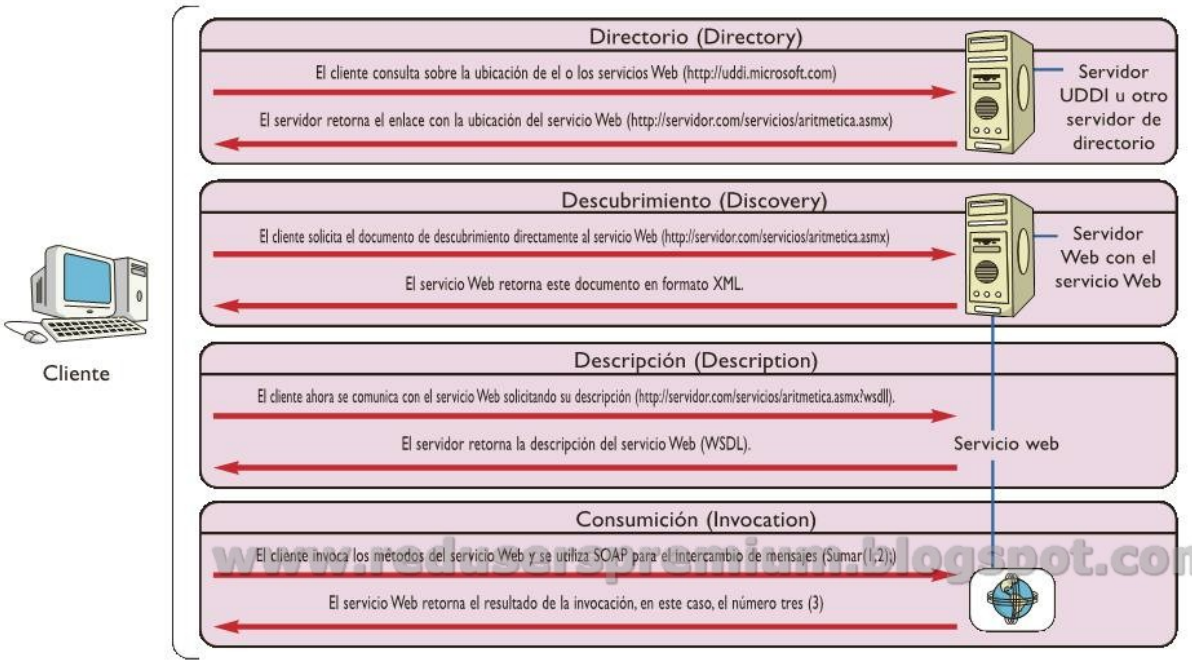


FIGURA 005 | Ciclo de vida de la consulta de un servicio Web.

- La **posibilidad de composición** radica en crear servicios generales en los que se basen los más específicos o de alto nivel. En el caso de los servicios Web, se puede instrumentar acerca de dos aspectos que sólo vamos a mencionar: protocolos de orquestación (WS-BPEL) y coreografía (WS-CDL).
- La **autonomía** requiere que cada servicio cuente con su propio espacio de ejecución. De esta manera, es independiente y nos aseguramos de que pueda ser reutilizado. En otras palabras y a modo de ejemplo, el servicio está en un servidor "de servicios Web", y lo consumen un sitio y una aplicación Windows. Si estuviera dentro de uno de los dos y ése quisiera ser migrado a otra plataforma, habría que migrar/cambiar dicho servicio.
- Los servicios **no deben mantener estados**. Esto quiere decir que, terminada su función, no deben mantener información de ningún tipo, con el fin de no producir inconsistencia de los datos. Una aplicación o conjunto de aplicaciones invocan o utilizan variados servicios y, por lo tanto, éstos no deben mantener la información para que entre ellos no se produzca inconsistencia.
- Obviamente, debe de existir la **posibilidad de descubrimiento**, es decir que una aplicación pueda encontrar el servicio para así poder utilizarlo. Esto se resuelve con UDDI.

Aspectos o normas que debe cumplir un servicio Web

Reusabilidad

Posibilidad de utilización dentro de una misma aplicación o varias aplicaciones.

Contrato formal

Brindar un nombre de servicio.
Brindar una forma determinada de acceso.
Brindar las funciones ofrecidas y sus parámetros, y los valores de retorno.

Bajo acoplamiento

Ser independiente de cualquier capa.

Composición

Crear servicios generales en los que se basen los más específicos.
Protocolos de orquestación (WS-BPEL).
Protocolos de coreografía (WS-CDL).

Autonomía

Debe contar con su propio espacio de ejecución

Estados

Terminada su función, no deben mantener información de ningún tipo.

Descubrimiento

Debe ofrecer una forma clara para su consumición.

FIGURA 006 | Resumen de los aspectos o normas a cumplir por un servicio web.

WSDL y UDDI

Veremos cómo encontrar un Web Service, y reconocer sus métodos, parámetros y datos devueltos.

Repasemos el significado de WSDL o *Web Service Description Language*. Esto indica que es un lenguaje mediante el cual se describe un servicio Web, y describir un servicio Web no es más que especificar qué métodos posee, qué parámetros recibe y qué valores retorna cada llamada que hagamos a él. Por supuesto que también (por medio del nombre del servicio y del método que se invoca) sabremos qué es lo que hará, aunque quizás no conozcamos exactamente cómo.
Este documento no es de carácter obligatorio, pero sí es estándar, ya que es la única manera en que se puede acceder a los servicios Web en forma dinámica. En una etapa de conocimiento de los Web Services no se consideraba relevante saber construir un documento WSDL, aunque resulta de suma importancia poder entenderlo.

En este caso, este XML en particular es el que describe o define la interfaz de los servicios, y está dividido en dos partes: la concreta, que define el cómo y el dónde; y la abstracta, es decir, qué hace a través de los mensajes que recibe y retorna.

La primera etiqueta con la que nos encontramos es **types**, que define las estructuras de datos que

Documento WSDL

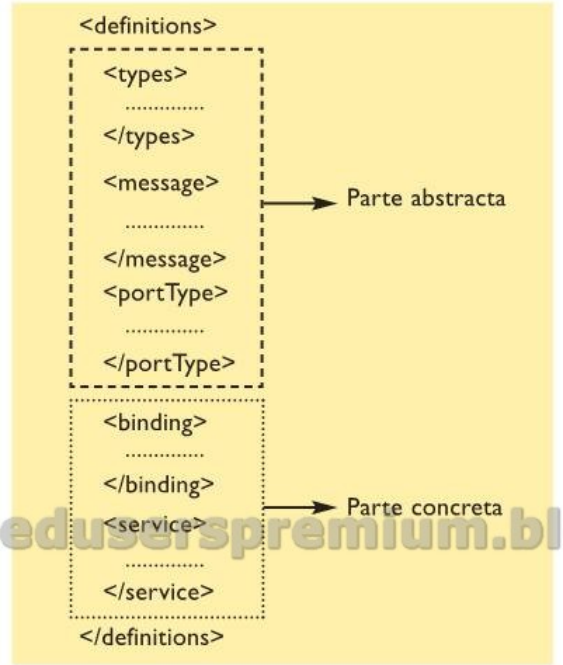


FIGURA 007 | Partes que debe contener para ser un documento válido.

se manejarán, tanto para los datos de entrada como para los de retorno.
También hay otro tipo de etiquetas, como **message**, que describe qué mensajes se intercambiarán entre el cliente y el servicio.
La última etiqueta de la parte concreta es la denominada **portType**, que establece el conjunto de operaciones antes mencionadas que puede resolver dicho servicio Web. Éstas pueden ser de cuatro tipos diferentes: **unidireccional**, en la que el servicio recibe algún dato pero no emite respuesta; **petición - respuesta**, en la que el servicio recibe un mensaje y devuelve otro; **solicitud - respuesta**, en la que el servicio envía un mensaje y recibe otro; y por último, **notificación**, en la que envía un mensaje pero no recibe respuesta.

Dentro de la parte abstracta, encontramos la etiqueta **binding**, que describe cómo formatear los mensajes para interactuar con el servicio en cuestión. Por último, está la etiqueta **services**, que indica dónde se encuentra el servicio.

UDDI o *Universal Description, Discovery and Integration*, nos permite no sólo describir un servicio Web, sino también localizarlo en forma dinámica para que los clientes y otros servicios lleven a cabo la creación de una plataforma interoperable. UDDI se monta sobre SOAP, y tanto al creador como al consumidor básico de servicio Web les será totalmente transparente.
Para entender este tema, mencionaremos que, al tratar de agregar una referencia Web o a un servicio Web dentro de nuestro proyecto, aparece una ventana en la que debemos ingresar la URL a nuestro servicio Web o bien escoger entre algunas acciones predeterminadas. Una de ellas es buscar los servicios Web del equipo local, y en nuestro caso, nos ofrecerá la posibilidad de ver nuestro servicio Web, denominado aritmética. Luego, si hacemos clic en él, veremos los diferentes métodos que expone.

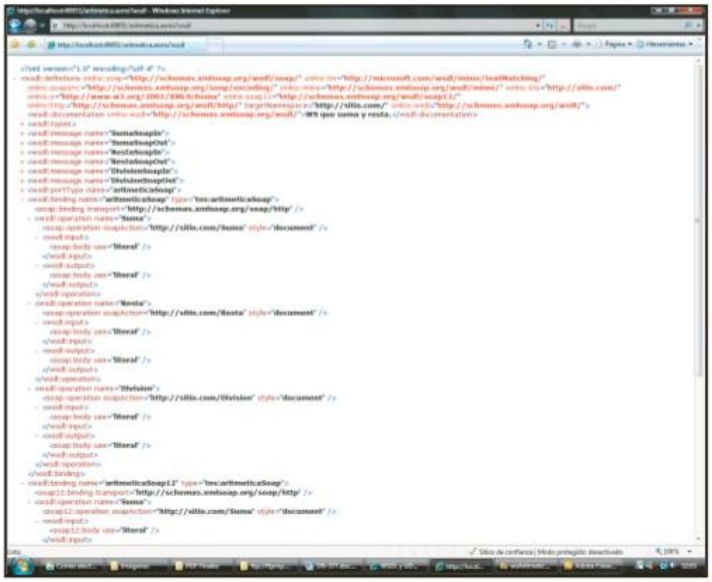


FIGURA 008 | Detalles de los nodos binding de un documento WSDL.

La evolución de WSDL

En junio del año 2007 se liberó formalmente el estándar WSDL 2.0, que incorpora importantes mejoras con respecto a su antecesor WSDL 1.1. Entre ellas encontramos funciones de importación, herencia, fallos/errores y compatibilidad total con HTTP y SOAP. Además se incorporaron las mejoras de WSDL 1.1 que refieren al perfil básico de WS-I (*WebService Interoperability*).
Si utilizamos WSDL 2.0, podremos crear y consumir servicios Web que permitan la llamada a métodos de forma asincrónica. Esto quiere decir que podemos implementar un servicio Web, llamar a uno de sus métodos y recibir una respuesta luego de un tiempo, sin necesidad de interrumpir la ejecución secuencial de nuestro programa. Todas las herramientas necesarias para trabajar con estos estándares estarán disponibles en la próxima versión de Microsoft Visual Studio, la 2008 (actualmente existen "agregados" para Visual Studio 2005 que permiten utilizar estos estándares, pero están avocados al .NET Framework 3.0.

Creación de un Web Service

En las siguientes páginas veremos cómo compartir información mediante servicios Web.

Repasemos, entonces, qué es un servicio Web. Éste permite la comunicación entre procesos o aplicaciones; es independiente del lenguaje de programación, del protocolo de comunicación y de la plataforma; y no almacena estados.

El primer paso para crear un servicio Web con **Visual Studio .NET 2005** es agregar un proyecto del tipo **servicio web asp.net**. Esto creará, entre otras cosas, una clase con extensión ASMX (extensión propia de los servicios Web) y otros archivos propios de la publicación Web, como **web.config** y **global.asax**. Presentaremos el código de la clase de nuestro servicio Web denominado wsAritmetica y lo analizaremos por partes:

```
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.ComponentModel;

namespace wsAritmetica {

    /// <summary>
    /// Descripción breve de aritmetica
    /// </summary>
    [WebService(Namespace = "http://tempuri.
    org/")]
    [WebServiceBinding(ConformsTo =
    WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class aritmetica : System.Web.
    Services.WebService
    {

        [WebMethod]
        public int Suma(int a, int b)
        {
            return a + b;
        }

        [WebMethod]
        public int Resta(int a, int b)
        {
            return a - b;
        }

    }

}
```

Se eliminaron comentarios que se agregan automáticamente por el entorno al momento de crear la clase, con el fin de compactar el código y lograr que su explicación resulte más simple. Entre los namespace o espacios de nombres citados vemos dos denominados **System.Web**, propio de todo lo referente a aplicaciones Web; y **System.Web.Services**, el que contiene todo lo relativo a los servicios Web.

La clase debe heredar de un WebService, lo que la convierte en una especialización de WebServices. Hereda de **WebService** del espacio de nombres **System.Web.Services**. A diferencia de versiones anteriores de Visual Studio .NET 2005, no es necesario declarar un constructor o destructor dentro de la clase; para nuestro servicio en particular se han declarado dos métodos, llamados **Suma** y

Resta, que se ven como métodos comunes como los de cualquier clase, con la única diferencia de que están encabezados por un atributo **WebMethod**. Con esto tenemos un Web Service ya creado y listo para ser consumido. En definitiva, podemos ver que estos servicios proponen una nueva forma de intercomunicar las capas de negocio basándose en protocolos abiertos, como XML y SOAP.

Para que al consumidor de nuestro servicio Web le resulte claro conocer qué hace dicho servicio, se le puede agregar, a modo de atributo de clase (aplica al servicio), la descripción de su funcionalidad.

Esto se realiza de la siguiente manera:

```
...
   /// <summary>
   /// Descripción breve de aritmetica
   /// </summary>
   [WebService(Description = "WS que suma y
   resta.", Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo =
   WsiProfiles.BasicProfile1_1)]
```

> El primer paso para la creación de un servicio Web es agregar un proyecto del tipo servicio web asp.net.

```
   [ToolboxItem(false)]
   public class aritmetica : System.Web.
   Services.WebService
   {
...
```

Como se puede apreciar, se ha agregado un atributo de clase denominado **WebService**, y se estableció un **string** a la propiedad **Description**. Esto lo verá nuestro consumidor (por ejemplo, en Internet Explorer) a continuación del nombre correspondiente.

De manera análoga, una vez que el servicio Web fue puesto en producción, podría cambiarse el **Namespace** o ambiente en el que será invocado estableciendo la propiedad **Namespace** con la URL correspondiente.

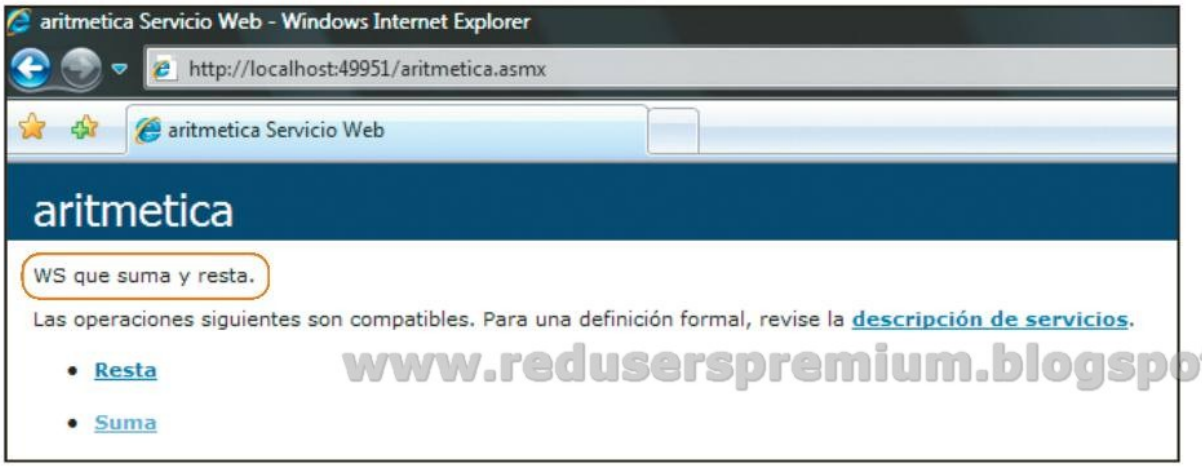


FIGURA 009 | La descripción puede ser visualizada cuando usamos nuestro navegador.

```
...
  [WebService(Description = "WS que suma y
  resta.", Namespace = "http://sitio.com/")]
...
```

Igualmente, esta propiedad puede no aparecer, y nuestro **Namespace** sería **http://tempuri.org**, que no es más que el valor por defecto. De la misma forma se puede colocar un valor a la propiedad **Description** del atributo **WebMethod**:

```
...
    [WebMethod(Description = "suma de dos
    enteros.")]
    public int Suma(int a, int b)
...
```

En este momento, lo que puede visualizar el consumidor es una información más detallada en cuanto a la funcionalidad del servicio Web y de cada método en particular.

Las propiedades posibles para un **WebServiceAttribute** son: **Description**, para especificar una descripción del servicio; **Name**, con el objeto de darle un nombre; **Namespace**, paraa establecer el espacio en que el servicio reside; y **TypeId**, que se implementa cuando es una clase derivada, y lo que hace es obtener un identificador único para este atributo. Ahora continuemos viendo las posibles propiedades para **WebMethodAttribute**, que son igualmente sencillas. Comencemos por **BufferResponse**, que indica si la respuesta a la solicitud del método se almacena o no en el buffer. Luego tenemos otra propiedad denominada **CacheDuration**, que determina la cantidad de segundos durante los cuales la respuesta es mantenida en el buffer. Ya analizamos **Description**. Con el fin de conocer si está habilitado el estado de sesión para este método, contamos con la propiedad **EnableSession**. **MessageName** establece el nombre que utiliza el servicio Web en los datos que se transportan o pasan, y que se devuelven. También hay una propiedad llamada **TransactionOption**, mediante la cual se indica la compatibilidad con transacciones para un método de servicio Web. Por último, mencionaremos la propiedad **TypeId**, de comportamiento idéntico a su par de **WebServiceAttributes**.

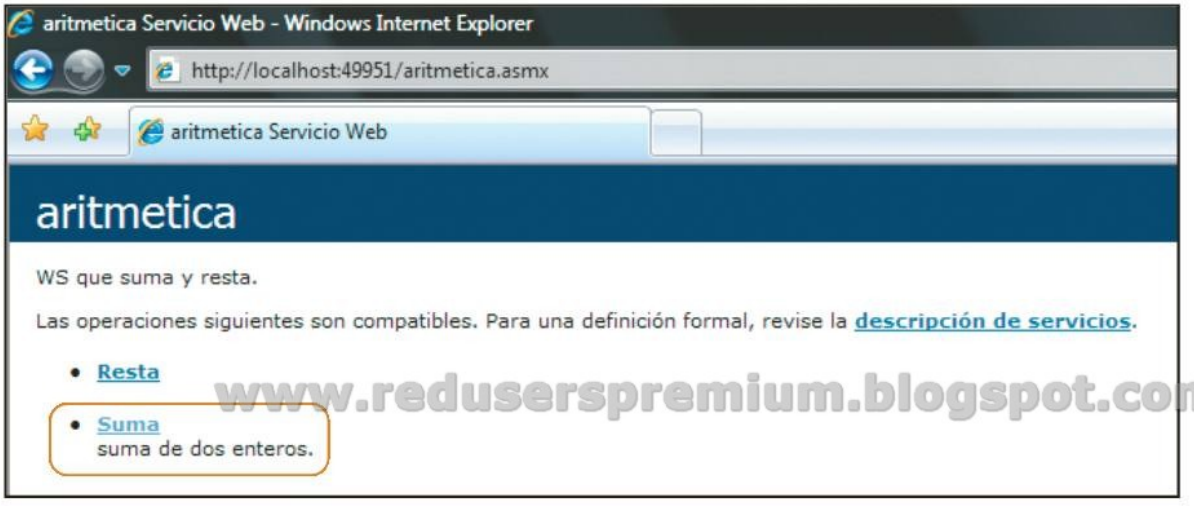


FIGURA 010 | La descripción de un método también puede ser visualizada en el navegador.

Regresemos a la primera propiedad, **BufferResponse**, y digamos que el valor por defecto es **true**. Por lo general, se recomienda que se establezca en **false** sólo cuando el método debe devolver grandes cantidades de información. Cuando está establecido en **true**, el método serializa la respuesta en el buffer de memoria hasta que haya terminado de completar la información, y desde ahí la envía, liberándose el buffer de memoria o bien hasta que éste se llene. Hay que tener en cuenta que cuando está en **false**, se dehabilitan las extensiones SOAP. La performance para pequeñas cantidades de información es mantener esta propiedad en **true**.

Ahora veamos **CacheDuration**, cuyo valor por defecto 0 indica que la información por enviar no queda almacenada en memoria. Esta propiedad puede establecerse en un valor en segundos, y si el consumidor vuelve a llamar al método con idénticos parámetros antes de ese límite de tiempo, **se le devuelve el valor serializado en memoria al instante**. Teniendo en cuenta que el objetivo de los servicios Web es mantener una cantidad importante de servicios y métodos que colaboren con la resolución de los procesos de negocios, y que éstos serán invocados por múltiples clientes en breves plazos y hasta, a veces, en forma simultánea, no se considera práctico que estas respuestas sean almacenadas en caché, ya que el consumo de memoria es importante y los tiempos de respuesta disminuyen, lo que produce una baja de performance significativa.
Para el caso de la propiedad **Description**, no hay mucho más que agregar, salvo que su valor por defecto es **String.Empty**.
Ahora bien, la propiedad **EnabledSession**, como su nombre lo indica, señala si se encuentra habilitado el estado de sesión o no. El valor por defecto es **false**. Si no hace falta habilitarlo, es mejor dejarla con su valor predeterminado, ya que esto mejorará la performance.
El valor predeterminado de la propiedad **MessageName** es el nombre del mismo método del servicio. Esta propiedad suele cambiarse para establecer alias y no modifica la performance. Sólo hay que tener en cuenta que este alias debe ser único entre todos los métodos Web del servicio. Esta propiedad suele modificarse en los casos pertinentes de polimorfismo, tema que no nos incumbe en este momento.

> El valor predeterminado de la propiedad MessageName es el nombre del mismo método del servicio.

Otra de las propiedades citadas anteriormente es **TransactionOption**, cuyo valor por defecto es **Disabled**. Esta propiedad se modifica en caso de que el método realice una llamada a un objeto COM que precisa la compatibilidad transaccional. Los otros valores posibles son **Required**, para indicar que requiere de una transacción; y **RequiredNew**, si precisa una transacción nueva. Cuando se

Tabla 1 | Propiedades de un WebServiceAttribute

Description	Mensaje descriptivo para el servicio web.
Name	Nombre del servicio web.
Namespace	Espacio de nombres XML predeterminado que se empleará en el servicio Web.
TypeId	Cuando se implementa en una clase derivada, obtiene un identificador.

completa la llamada al método sin producirse ninguna excepción, la transacción se efectúa completamente; en caso contrario, todo el proceso es abortado. Se puede abortar a demanda llamando explícitamente a **SetAbort**.

En cuanto a **TypeId**, no hay mucho por mencionar, salvo que es un identificador único y se utiliza para discernir entre dos atributos del mismo tipo. Esta propiedad es del tipo **Object**. Para ampliar el concepto de transaccional, podemos citar un ejemplo del procesamiento de una orden de compra, en la cual nos interesa bajar la cantidad comprada del stock existente de un depósito. Para hacerlo, tenemos un método de un servicio Web que recibe una orden de compra, y otro método que es invocado N veces, de acuerdo con la cantidad de registros de nuestra orden, y que irá actualizando nuestra base de datos teniendo en cuenta el stock actual y la cantidad comprada de cada artículo.

Transaccional

Un servicio Web puede ser transaccional y manejar datos de sesión; por lo tanto, resulta una excelente opción para manejar la capa de negocios de cualquier empresa en un entorno de aplicaciones distribuidas.

Por lo tanto, tendremos un método Web que podríamos denominar como "procesar orden de compra" y otro "descontar stock". El primero será invocado por un consumidor de nuestro servicio Web, y el segundo, por el método anterior, por cada uno de los artículos comprados. Si uno de los artículos en cuestión contiene menor stock que el deseado, en la compra se abortará la transacción, y al estar en un entorno transaccional, se realizará un **rollback** de las transacciones anteriores, dejando automáticamente de procesar la orden de compra y regresando los datos a su estado primitivo.

Para realizar esto, deberemos de agregar el atributo correspondiente:

```
...
    [WebMethod(TransactionOption =
    System.EnterpriseServices.Transacti
    onOption.Required)]
    public int Suma(int a, int b)
...
```

Con el fin de abortar la carga transaccional, se debe de invocar el método:

```
...
System.EnterpriseServices.ContextUtil.Set
Abort();
...
```

Tabla 2 | Propiedades de un WebMethodAttribute

BufferResponse	Establece si la respuesta para esta solicitud se almacena en el búfer.
CacheDuration	Establece el número de segundos durante los cuales la respuesta es mantenida en el buffer.
Description	Mensaje descriptivo para el servicio Web.
EnableSession	Establece si se habilita el estado de sesión para un método del servicio Web.
MessageName	Establece el nombre que se utiliza para el método de servicios Web.
TransactionOption	Indica la compatibilidad con transacciones de un método de servicios Web.
TypeId	Cuando se implementa en una clase derivada, obtiene un identificador.

O bien, si toda la operación ha sido completada con éxito:

```
...
System.EnterpriseServices.ContextUtil.Set
Complete();
...
```

Pero aún nos falta algo que nos debe preocupar mucho y es el tema de la seguridad, ya sea que el servicio Web esté dentro de un intranet o de la Web. Hay varias maneras de crear un entorno seguro, y una de ellas es mantener ciertos datos en sesión. Por ejemplo, se comienza con un llamado a un método de autentificación que creará un nuevo **token** y lo retornará, mientras que, al mismo tiempo, queda en caché. Para las siguientes invocaciones a métodos Web, se deberá pasar como parámetro este **token**, y el servicio Web lo cotejará con su lista de **tokens** activos y supondrá que estamos autorizados a ejecutar ese método. Tengamos en cuenta que esta caché se va liberando con el tiempo y, por lo tanto, quizás haya llamado a un método habiendo expirado el **token** en cuestión, lo que equivaldría a una sesión de usuario expirada.

Publicación y testeo de Web Services

El proceso de publicación de un servicio Web **es idéntico al de una aplicación Web de ASP.NET**, ya que un servicio Web es también una aplicación de este tipo. Por lo tanto, hay que crear la carpeta correspondiente dentro de nuestro IIS (*Internet Information Server*) y copiar el servicio en ella.

Como cualquier aplicación Web, estos servicios también poseen un archivo denominado **web.config**, en donde podemos encontrar dos elementos principales: **webServices**, que se utiliza para configurar una o varias opciones, como **protocols**, donde se especifica a qué tipos de mensajes debe responder el o los servicios; y **sessionState**, cuya función es indicar cómo se deben manejar los datos almacenados en el objeto **Session**.

También es posible agregar parámetros propios

Comunicación basada en tokens

FIGURA 011 | Mediante este sistema, debemos autenticarnos contra el servidor y utilizar la respuesta que nos brinda para poder consumir el servicio web.

Deberíamos contemplar la posibilidad de probar los servicios Web desde otras PCs.

de configuración, como una cadena de conexión a una base de datos, etc. Como toda aplicación Web, Windows o todo lo que desarrollamos, un Web Service también debe probarse con el fin de verificar su correcto funcionamiento. Para llevar a cabo esta tarea, hay dos caminos factibles: uno es abordarlo y consumirlo en forma directa, y otro es consumirlo. Por el momento, nos dedicaremos al primero.

Como ya hemos visto, en un explorador de Internet podemos colocar la URL del servicio Web y probarlo. Ésta es la manera más simple de probar el servicio en cuestión. El problema surge cuando, en vez de recibir un dato de tipo simple, recibe un objeto. Para este caso, no nos queda más opción que desarrollar una pequeña aplicación de prueba y que ésta envíe al servicio Web (o reciba o ambas cosas) dicho objeto.

También tenemos la opción de probar nuestro servicio desde el entorno de desarrollo, que nos abrirá la pantalla del navegador tal como si lo navegáramos. La diferencia radica en que, de esta otra manera, podremos depurar nuestro servicio intentando identificar un problema, etc. Si navegáramos directamente nuestro servicio Web y quisiéramos depurarlo, deberíamos utilizar un método más complejo, que implica adjuntar un proceso a nuestro depurador, y esto es algo que escapa al alcance de este capítulo.

Ahora bien, hemos agregado a nuestro servicio Web de aritmética el método Web de división, con el código siguiente:

```
...
    [WebMethod]
    public double Division(int a, int b)
    {
        return (double)a / (double)b;
    }
...
```

Para cualquier ejemplo que tomemos, nos devolverá el resultado en formato de **double**, salvo cuando nuestro segundo valor (b) sea cero (0). En este caso, el servicio Web devolverá el XML con otro tipo de resultado como **INF**, que no es más que la abreviatura de infinito. Como todos sabemos, no es posible dividir por cero, y es por eso que se ha producido este resultado. Pero también podemos observar que no se ha elevado una excepción, ya que al ser procesos (invocación del servicio y el servicio en sí mismo) totalmente desligados el uno del otro, no podría hacerlo. De esta forma entonces nos devuelve INF (infinito) al dividir por 0 en nuestro Web Service aritmética.

```
<?xml version="1.0" encoding="utf-8" ?>
<double xmlns="http://tempuri.org/">INF</double>
```

De todas formas, la prueba final se debe llevar a cabo en un entorno similar al de producción. Esto quiere decir que hay que compilar los servicios Web en modo **Release** y alojarlos bajo la carpeta virtual correspondiente en el Internet Information Server. Luego, hay que navegarlos desde una ventana de exploración como ya hemos mencionado, verificando todos los métodos implementados. También habría que probar los casos que resultan particulares o excepcionales, como la división por cero (0).

En caso de estar trabajando sobre nuestra PC, deberíamos contemplar la posibilidad de probar dichos servicios Web desde otras, o bien desde otras ubicaciones externas a nuestra intranet, con el fin de poder verificar también los tiempos de respuesta, etc.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

16

Consumiendo Web Services

Encriptación - Introducción a WSE

Seguridad

Amenazas en aplicaciones Web

STRIDE - DREAD

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ISBN 978-987-1347-43-8
00016
9 789871 347438