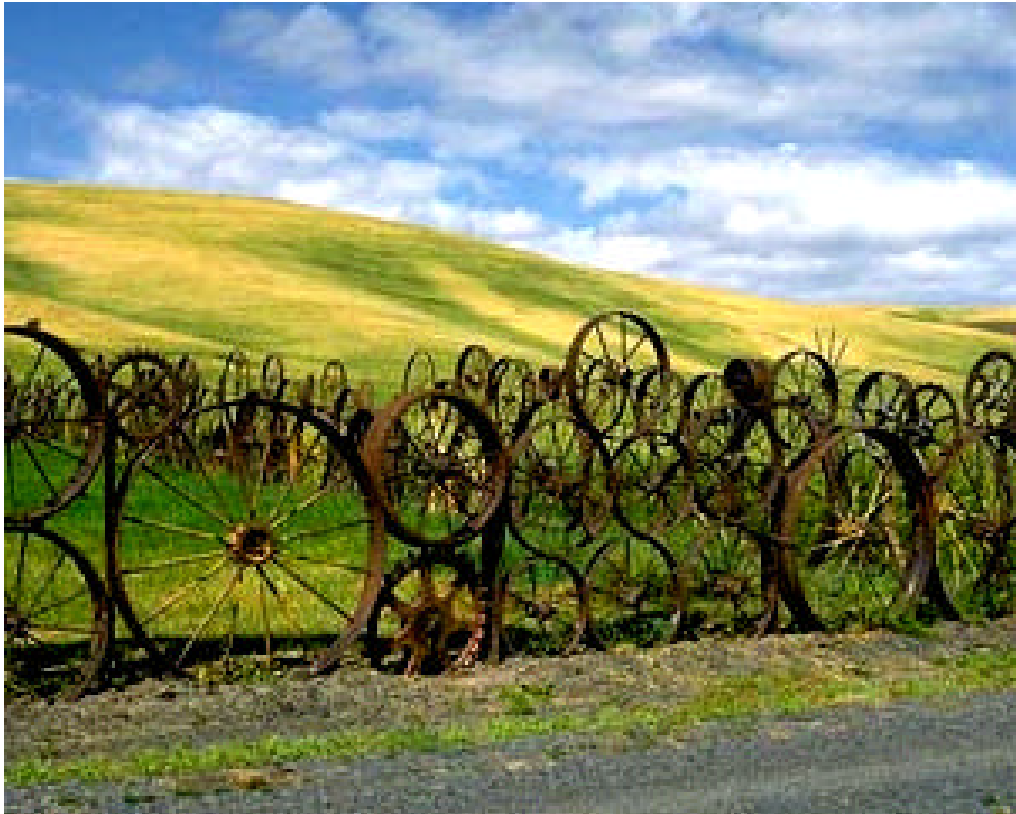


GUÍA DEL DESARROLLADOR DE APLICACIONES WEB CON

EXPRESSO FRAMEWORK™



por Michael Nash, Jcorporate Ltd.

traducido del inglés por: © [Gaspar González Oliva](#), 2001

<http://www.jcorporate.com>

VERSIÓN 1.0

TABLA DE CONTENIDO

CAPÍTULO 1. INTRODUCCIÓN.....	5
CAPÍTULO 2. VISTAZO GENERAL AL EXPRESSO FRAMEWORK.....	6
MARCO DE TRABAJO DE LA APLICACIÓN.....	6
SERVICIOS.....	7
ORGANIZACIÓN Y CONFIGURACIÓN DE LA APLICACIÓN.....	7
ACCESO A BASES DE DATOS Y OBJETOS DE BASES DE DATOS.....	8
OBJETOS DE BASE DE DATOS.....	8
CONEXIONES A BASES DE DATOS.....	9
ARQUITECTURA MODELO-VISTA-CONTROLADOR (MODEL VIEW CONTROLLER).....	10
MANIPULACIÓN Y ENCOLAMIENTO DE PROCESOS (JOB).....	11
SERVLETS.....	11
EVENTOS.....	12
UTILITARIOS.....	12
CAPÍTULO 3. DESARROLLANDO UNA APLICACIÓN CON EXPRESSO	14
CAPÍTULO 4. OBJETOS DE BASE DE DATOS.....	17
¿PORQUE USAR OBJETOS DE BASE DE DATOS?.....	17
CAPACIDADES DE OBJETOS DE BASE DE DATOS.....	18
NÚMERO MÁXIMO DE ARTÍCULOS.....	19
INFORMACIÓN DEL ESTADO.....	19
CAPACIDAD DE MÚLTIPLES BASE DE DATOS.....	19
INFORMACIÓN SOBRE LA TABLAS.....	19
RASTREO (LOGGING) DE CAMBIOS.....	20
BÚSQUEDAS Y CONJUNTOS DE RESULTADOS.....	20
INTEGRIDAD REFERENCIAL DECLARATIVA.....	20
ADICIONANDO ARTÍCULOS.....	20
MAPEO DE TIPOS.....	21
CAMPOS DE MÚLTIPLES VALORES.....	21
DESCRIPCIÓN DE CAMPOS.....	21
TABLAS GENERADAS AUTOMÁTICAMENTE.....	21
OBJETOS BUSCADOS (LOOK UP OBJECTS).....	22
CAMPOS DE SOLO LECTURA.....	22
NÚMEROS SECUENCIALES.....	22
PRUEBAS.....	22
USANDO LOS OBJETOS DE BASE DE DATOS.....	23
RECUPERANDO ARTÍCULOS.....	26
RECUPERANDO MÚLTIPLES ARTÍCULOS.....	28
MANIPULANDO DATOS ORDENADOS.....	29

MANIPULANDO GRANDES CONJUNTOS DE DATOS.....	30
USANDO RANGOS Y COMODINES.....	31
ACTUALIZANDO ARTÍCULOS.....	33
ADICIONANDO NUEVOS ARTÍCULOS.....	33
BORRANDO ARTÍCULOS.....	33
CLÁUSULAS “WHERE” PERSONALIZADAS.....	33
ESPECIFICANDO BASE DE DATOS.....	34
SEGURIDAD Y OBJETOS SECUREDDBOBJECT.....	35
USANDO DBMAINT.....	36
CONTROL DE TRANSACCIONES.....	37
OBJETOS MULTIDBOBJECTS.....	39
CAMPOS VIRTUALES.....	42
CAMPOS DE SOLO LECTURA.....	43
CAMPOS DE MÚLTIPLE VALORES.....	44
CAMPOS SECRETOS.....	45
VALIDACIÓN DE CAMPOS.....	45
LOS OBJETOS AUTODBOBJECT.....	46
PROCESO DE “CACHING”.....	47
OBJETOS DE BASE DE DATOS EN MÚLTIPLES BASE DE DATOS.....	48
CONTEXTO LOGGED-IN.....	48
MAPEANDO OBJETOS DE BASE DE DATOS.....	49
CAPÍTULO 5. USANDO LOS OBJETOS CONTROLADORES.....	53
VISTAZO GENERAL.....	53
¿PORQUÉ USAR CONTROLADORES?.....	53
CONTROLADORES BÁSICOS.....	55
ENTRADAS (INPUTS).....	55
SALIDAS (OUTPUTS).....	55
TRANSICIONES (TRANSITIONS).....	56
BLOQUES (BLOCKS).....	56
TIPOS DE CONTROLADORES.....	59
EXPLICACIÓN DE LA ACTIVIDAD DEL OBJETO CONTROLLER.....	60
ESTADOS (STATES).....	62
ESTADO INTERNOS.....	63
ESTADOS EXTERNOS.....	65
CAPÍTULO 6. LOS COMPONENTES PROCESO(JOB).....	68
ESCRIBIENDO OBJETOS “JOB”.....	68
LOS OBJETOS “JOB”. SU SEGURIDAD.....	69
ENCOLANDO LOS PROCESOS (OBJETOS “JOB”).....	69
CAPÍTULO 7. DESARROLLANDO PÁGINAS JSP.....	71
CAPÍTULO 8. DESARROLLANDO SERVLETS.....	72
SERVLET QUE NO SE CONECTAN A BASE DE DATOS.....	72
SERVLETS DE BASE DE DATOS.....	73
SERVLET DE MANTENIMIENTO DE LA BASE DE DATOS.....	74
U25 0..TD./F2.9..Tf.0.1250.0.1738..250.0.1c.0.ARRROLLANDO33.Tj 54.75 0 TD /F2 11.25 292 0.1275	

DBMAINT: UN EJEMPLO PASO POR PASO.....	77
CAPÍTULO 9. DESPLIEGUE DE COMPONENTES EXPRESSO PARA APLICACIONES.....	84
CAPÍTULO 10. MONITOREANDO Y VERIFICANDO LAS OPERACIONES DE LAS APLICACIONES EXPRESSO	85
PRUEBAS DE UNIDADES	85
PRUEBAS DE RENDIMIENTO.....	86
USANDO PRUEBAS DE RENDIMIENTO. CONJUNTO DE PRUEBAS DE RENDIMIENTO.....	87
CHEQUEO DE VITALIDAD (HEALTHCHECK).....	87
CONTROLADOR “RUNTEST”	89
OPTIMIZANDO EL RENDIMIENTO.....	90
FACTORES QUE INFLUYEN EN EL RENDIMIENTO.....	90
AJUSTE DE LA CACHE.....	90
OPTIMIZACIÓN DE LA BASE DE DATOS.....	93

CAPÍTULO 1. INTRODUCCIÓN

Bienvenido a la *Guía del Desarrollador de Aplicaciones Web con Expresso Framework*. Esta guía tiene como objetivo proveer toda la información a los desarrolladores con Java para poder usar todas las ventajas del Marco de Trabajo para el desarrollo de Aplicaciones Web **Expresso Framework** así como su biblioteca de Componentes.

Este le da en una forma clara y concisa, el resultado de miles de horas de tiempo de programación trabajando con Expresso y desarrollando de igual forma las mejores prácticas para su uso e implementación.

Esta guía es el resultado de muchas contribuciones, ideas y trabajo de la comunidad que usa Expresso Framework, y nosotros estamos abiertos a cualquier sugerencia y comentario sobre esta. Nuestra meta es continuar perfeccionando esta en la medida que Expresso progrese, para que esta continúe siendo una valiosa herramienta en manos de los desarrolladores profesionales.

Esta guía puede ser leída en orden secuencial, ya que comienza dando un vistazo general de todas la herramientas y el potencial que están contenido en Expresso Framework, luego usted puede continuar con los detalles de cada una de estas herramientas y como usarlas rápidamente en el desarrollo de sus propias Aplicaciones Web.

Esta guía pretende ser un suplemento a la documentación de Expresso, y la mejor forma de usarla es en conjunción con dicha documentación, principalmente el Documentación Java(JavaDoc) para Expresso. El código fuente es sí mismo, por supuesto, es la última referencia de lo que contiene Expresso y solo debe ser consultada cuando se requiera.

CAPÍTULO 2. VISTAZO GENERAL AL EXPRESSO FRAMEWORK.

Expresso es un Marco de Trabajo(Framework) que hace hincapié en la construcción de Aplicaciones Web robustas, seguras, confiables y ricas en características, todo esto de una manera rápida y sencilla.

MARCO DE TRABAJO DE LA APLICACIÓN

Para usar Expresso y sacarle el mejor provecho este deberá ser usado como un Marco de Trabajo para su Aplicación. Esto implica usar los mecanismos de Inicialización, Configuración y administración de la “Cache”, Objetos de Bases de Datos y sistemas de Seguridad, y su Controlador Modelo-Vista-Controlador (Model-View-Controller) así como los objetos de Estados (“State” object), todo esto en conjunción con los mecanismos de presentación de su selección tales como JSP (con la disponibilidad de la biblioteca de tags Struts), Webmacro y muchos otros.

Usando Expresso de esta forma aun deja espacio para la integración con otras herramientas de desarrollo, Marcos de Trabajo y bibliotecas de componentes. Expresso está diseñado para ofrecer una máxima extensibilidad y poder ser integrado de manera muy fácil con otras herramientas.

Mientras es absolutamente posible usar los componentes de Expresso sin usarlo completamente, el sistema hace la construcción de aplicaciones Web muy fácil cuando es usado como un Marco de Trabajo. Expresso impone un mínimo de restricciones en una aplicación Web. Este no esta casado con ningún método de navegación o estilo en particular, ni con ninguna arquitectura en particular. Las aplicaciones Web que sean construidas usando Expresso como su marco de trabajo pueden usar JSP, Servlets corrientes, Applets o Aplicaciones locales como su Interfaz de Usuario (User Interface - UI).

En los capítulos que siguen, discutiremos principalmente el uso de Expresso como un Marco de Trabajo, pero las secciones individuales que describen componentes como los objetos de Base de Datos, Controladores,

Procesos y así sucesivamente puede ser aplicado para el uso de estos componentes de manera individual.

Expresso depende en gran medida de al menos una Base de Datos relacional estando disponible para su operación y es más apropiado para aquellas aplicaciones en la cual la Base de Datos es una parte esencial de la aplicación.

SERVICIOS

Expresso provee un número de servicios que ayudan en la construcción de las aplicaciones Web. Las principales áreas que usan estos servicios son:

ORGANIZACIÓN Y CONFIGURACIÓN DE LA APLICACIÓN

La estructura fundamental de una Aplicación con Expresso está contenida en el objeto “Schema”. Este objeto extiende la clase “Schema”, y sirve como una lista de todos los otros objetos que constituyen una aplicación en particular.

Expresso usa para sí mismo una clase “Schema”

(com.jcorporate.expresso.core.ExpressoSchema) para describir las clases comunes a todas las aplicaciones que usan Expresso.

Expresso puede usar un objeto “Schema” de una aplicación para configurar automáticamente esta aplicación. El Servlet “DBCreate” lee la lista de todos los objetos “Schema” y puede:

- Crear todas las tablas definidas por los objetos de Bases de Datos en la aplicación.
- Proveer entradas de seguridad por defecto permitiendo al grupo de administradores acceder a todos los nuevos objetos de Bases de Datos que han sido creados.
- Proveer entradas de seguridad por defecto para cualquier objeto “Controller” en la aplicación.
- Proveer entradas de seguridad por defecto para cualquier objeto “Proceso” en la aplicación.

Una vez que la aplicación es inicializada, Expresso provee funciones de Administración para la aplicación como son:

- La entradas del fichero “Log” de la aplicación pueden ser vistas y sus fichero “Log” administrado por los controladores(objetos “Controller”) suministrados en Expresso.
- La seguridad de la aplicación puede ser administrada en forma de una Matriz de manera muy fácil, usando los Objetos “ControllerSecurityMatrix”, “DBObjSecurityMatrix” y “JobSecurityMatrix”.
- Los procesos en la aplicación (objetos “Job”) pueden ser puesto en cola por el controlador “QueueJob”, este controlador puede ser entonces extendido para poner en cola trabajos personalizados que requiera la aplicación.
- Todos los objetos de Bases de Datos en la aplicación pueden ser mantenidos (adicionar | actualizar | borrar | buscar) por el controlador por defecto DBMaint, simplemente pasando el nombre del objeto de Base de Datos.

ACCESO A BASES DE DATOS Y OBJETOS DE BASES DE DATOS

OBJETOS DE BASE DE DATOS

Los objetos de Base de Datos usan las conexiones a las Bases de Datos descritas arribas para proveer una forma de mapear desde los objetos la tablas de las Bases de Datos relacionales. Los objetos de Bases de Datos son similares y compatibles con los componentes Java empresariales (EJB) del tipo donde la persistencia de los datos es manejada por el componente (BMP Entity –ver el proyecto Expresso EJB-).

Ellos están diseñados en primer lugar para proveer los mecanismo de persistencia para los objetos de negocio, pero con frecuencia incluyen lógica de negocio también. Los Objetos de Bases de Datos proveen métodos para adicionar, actualizar y borrar, y un gran número de variados métodos de recuperación. El método “searchAndRetrieve” integra el proceso de búsqueda a la recuperación de artículos con una forma de acceder a los artículos recuperados, y es usado con frecuencia para aplicar algún tipo de procesamiento a un conjunto de artículos.

Una extensión de los “DBObject” básicos es el objeto de tipo SecuredDBObject el cual usa una serie de tablas que contienen información sobre los grupos y usuarios para suministrar seguridad a la Base de Datos al nivel de objetos. Los usuarios son agrupados en Grupos, y estos grupos solamente dan los permisos que ellos requieren para acceder a los Objetos de Bases de Datos. Los objetos de Bases de Datos que heredan de “SecuredDBObject” hacen uso de esta seguridad automáticamente, sin esfuerzos adicionales por parte de los desarrolladores.

CONEXIONES A BASES DE DATOS

Expresso provee facilidades para gestionar conexiones de Bases de Datos, abstrayendo el proceso de conexión aún mas que la interfaz de programación JDBC. Una sofisticada piscina de conexiones provee acceso a una o más Bases de Datos de una manera eficiente, proporcionando las siguientes facilidades:

- Piscinas de múltiples conexiones/Contextos
Las capacidades de piscina de conexiones provee acceso a varias piscinas de conexiones diferentes, la cuales potencialmente pueden trabajar en Bases de Datos separadas, aún siendo de proveedores diferentes. Por ejemplo, sus tablas principales pueden almacenarse en Oracle, pero usted puede tener una conexión alternativa a una Base de Datos en DB2/400 en un servidor diferente.
- Tamaño Máximo
Un valor definido puede especificar el tamaño máximo para la piscina de conexiones. Cuando la piscina alcanza este tamaño, el código que controla la piscina de conexiones trata de limpiar conexiones echadas a perder y rehusar estas si es posible, para satisfacer nuevas solicitudes de conexiones.
- Fueras de Tiempo
La piscina de conexiones está protegida contra errores accidentales donde las conexiones no son liberadas por los mecanismos de Fuera de Tiempo. Este mecanismo de Fuera de Tiempo retorna una conexión a

la piscina después de un cierto intervalo, en caso de que el programa cliente olvide liberar la conexión de una forma normal. Este mecanismo de Fuera de Tiempo puede ser sobrescrito para solicitudes que demorarán su ejecución.

- Conexiones Verificadas

La piscina de conexiones puede ser opcionalmente verificada antes de que esta sea suministrada al programa cliente corriendo una serie de pequeñas preguntas (query) con dicha conexión, por esa razón la manipulación de situaciones donde la Base de Datos pueda cerrar conexiones a esta se acabó y elimina que el programa cliente trabaje con una conexión cerrada. Este mecanismo usa ambos, una pregunta (query) de prueba y el método `isClosed()`, ya que `isClosed()` puede dar falsas indicaciones en algunas situaciones. Esto resulta en un incremento global en la flexibilidad y confiabilidad a sus aplicaciones escritas usando Expresso.

- Limpieza automática

La piscina de conexiones limpiará de manera automática las conexiones inactivas luego de un cierto período de tiempo, reduciendo el tamaño de la piscina a un mínimo especificado durante los períodos de inactividad.

- Configuraciones de Bases de Datos específicas

La piscina de conexiones permite opciones específicas para las Bases de Datos y serán especificadas como Propiedades. ver la documentación del fichero de Propiedades para más detalles.

ARQUITECTURA MODELO-VISTA-CONTROLADOR (MODEL VIEW CONTROLLER)

Expresso incluye un paquete de componentes para la creación de múltiples tipos de objetos controlador (“Controller” objects). Estos objetos Controladores encapsulan una serie de interacciones con el usuario, en una manera similar a los componentes Java empresariales de sesión (Session EJB), de hecho un Controlador puede ser un componente Java empresarial de sesión en un ambiente

donde estos componentes (EJB) sean soportados. El Controlador puede ser utilizado desde cualquier tipo de cliente: un “Servlet”, un “JSP”, un “Applet”, o una aplicación. Los Controladores son analizados en mas detalles en su propio capítulo en esta propia guía.

MANIPULACIÓN Y ENCOLAMIENTO DE PROCESOS (JOB)

Expresso contiene un facilidad para manipular tareas muy largas que un usuario normalmente no le gustará esperar (algunos estudios muestran que pueden ser tan bajas como solo 2 segundos). Estas facilidades proveen al cliente una forma de almacenar estas entradas en una cola de trabajos y un Proceso en el lado del servidor seleccionará estas entradas y llevará a cabo el procesamiento adecuado.

El Proceso es encolado por la creación de un objeto de Base de Datos “JobQueue” y sus entradas asociadas “JobQueueParam”, la forma de proveer los parámetros a los objetos es la misma en que se proveen estos a los métodos.

Un objeto en el lado del servidor que hereda de la clase “Job” es invocado cuando el manipulador de los procesos “JobHandler” comienza a hacerlo. Esta clase puede leer los parámetros dados a ella y procesar su controlador como requiera, generalmente reportándole al usuario que inició la tarea por vía correo electrónico que esta se completo.

Pueden haber múltiples procesos “JobHandler” todos trabajando en la cola “JobQueue” a la misma vez, y ellos sincronizarán sus solicitudes para que solo un proceso del servidor seleccione un trabajo determinado. Esto permite una tolerancia a fallos y escalabilidad, y mejoras en el rendimiento pueden ser adicionadas simplemente adicionando nuevos objetos “JobHandler” para manipular las solicitudes de manera más rápida.

SERVLETS

Expresso brinda una forma fácil y rápida para la creación de útiles “Servlet” y páginas “JSP”. La creación dos tipos de “Servlets” (los que usan

Bases de Datos y los que no la usan) son discutidos mas adelante. La creación de páginas “JSP” que usan Expresso son discutidas mas adelante de igual forma.

Una de las mas poderosas capacidades de los “Servlets” es que un Controlador (“Controller”) de mantenimiento de la Base de Datos (por ejemplo uno que permita adicionar | actualizar | buscar y borrar en la tabla en uso) puede ser creada simplemente pasando un parámetro al “Servlet DBMaint” donde el parámetro es el nombre del objeto de Base de Datos, el cual le dice al “Servlet DBMaint” todo lo que necesita conocer para mantener de manera automática de el objeto de Base de Datos.

Clases estándares en Expresso proveen la manera para que los Controladores reporten sus errores y excepciones al usuario de una forma uniforme, y proveen la manera de construir referencias a otros “Servlets” (ejemplo para las acciones de formas, botones, etc.).

EVENTOS

Expresso también define mecanismo para la definición de Eventos que ocurren dentro de una aplicación Web. Un ejemplo de un Evento es un error del sistema –este es un evento que viene predefinido con el sistema-. Este Evento puede tener una lista de usuarios asociados con el, los cuales serán notificados cuando este ocurra, en el caso de errores del sistema, usted probablemente deseará notificárselo a su administrador del sistema.

El desarrollador simplemente tiene que especificar que Evento será lanzado y las notificaciones correspondientes serán enviadas de manera automática por el sistema.

UTILITARIOS

Un conjunto de otras facilidades no caen en ninguna de las categorías anteriores, pero que pudieran ser muy útiles. Algunas de estas son:

- Manipulación de Ficheros: Utilitarios para copiar y mover ficheros, y llevar a cabo manipulaciones con los nombre de estos.

- Utilitarios de cadenas de caracteres: Algunas utilitarios para una manipulación más compleja que la que permite la interfaz de programación (API) “Java String”.
- Rastreo de situaciones (“Logging”): Clases que hacen papel de Interfaz con el proyecto Apache log4j, permitiendo personalizarse completamente para rastreos de múltiples niveles hacia ficheros, Bases de Datos, y otras destinos con bajo impacto en el rendimiento.
- Procesos del Sistema Operativo: Clases para encapsular las llamadas a programas ejecutables externos, y esperar por un intervalo de tiempo especificado a que estos terminen.

Una de las más detalladas referencias para Expresso es el “JavaDoc”, el cual usted puede bajar junto con el código fuente del Expresso. El código fuente sigue siendo la última herramienta de referencia.

El resto de los documentos en este paquete proveerán la información necesaria para comenzar a usar Expresso.

CAPÍTULO 3. DESARROLLANDO UNA APLICACIÓN CON EXPRESSO

Esta guía tiene como objetivo describir el proceso de desarrollo de una Aplicación con Expresso. Aunque hay muchas formas diferentes de usar los componentes de Expresso, los pasos descritos aquí son los óptimos desde el punto de vista del diseño del sistema, y le brindará los mejores resultados.

Desarrollar una aplicación con Expresso consiste de los siguientes pasos generales:

- Planeamiento de la Aplicación

Una aplicación Expresso consiste normalmente de una serie de objetos de Base de Datos (conteniendo el modelo y la persistencia de la aplicación), ver la sección sobre los objetos de Base de Datos para más información sobre los objetos de Base de Datos. Los objetos Controladores (proveen el aspecto controlador de la arquitectura Modelo-Vista-Controlador, y encapsulando todas las interacciones con los usuarios), objetos de Procesos (“Job”) (manipulando todas las tareas que demoran en su ejecución así como los procesos que deben ejecutarse en “background”), y un único objeto de “Schema” para mantener a todos juntos. Por supuesto, la aplicación puede tener muchos otros objetos que no caen en esta categoría, pero esta son las principales clases de objetos con las que Expresso trata y con a las cuales el puede ayudar. Por favor, ver la documentación específica en la “Guía del Desarrollador de Aplicaciones Web” con Expresso para cada uno de estos tipos de objetos y donde y como pueden ser mejor usados.

- Crear el paquete

Típicamente una aplicación Expresso reside en un solo paquete, con un número de sub-paquetes. Por ejemplo, si usted esta creando una aplicación para la administración de miembros, usted podría llamar a su paquete `com.nombredeasucompañia.miembros`. Dentro de los paquetes básicos estarán por lo general los siguientes paquetes:

- dbobj: Contiene los objetos de Base de Datos para la aplicación.
- controller: Contiene los objetos Controladores para la aplicación.
- server: Contiene objetos del Lado-del-Servidor para la aplicación, incluyendo Procesos (Jobs) y cualquier programa independiente (ejemplo de línea de comando o basados en Interfaces de Usuario) que estén relacionados con la aplicación.
- servlet: Cualquier Servlet personalizado requerido por la aplicación, generalmente es mejor y mas flexible depender en los objetos Controladores y el Servlet Struts tal como ActionServlet.

- Crear un objeto “Schema” para la aplicación

El objeto “Schema” para la aplicación permite a la aplicación entera ser referenciada por Expresso. Este le permite a Expresso administrar la seguridad de la aplicación y otro tipo de administración de manera automática cuando el objeto “Schema” es registrado con Expresso (ver la página “Aplicaciones con Expresso” para mas detalles del proceso de registrar los objetos “Schema”). Por lo general usted creará el objeto Schema en el paquete “raíz” de la aplicación.

- Crear cualquier objeto de Base de Datos requerido

Los objetos de Base de Datos son normalmente creados en el sub-paquete “dbobj” de la aplicación. Una aplicación Expresso puede usar tantos objetos de Base de Datos como requiera, y cada uno deberá ser listado en el objeto “Schema” de la aplicación, con el fin de que los utilitarios “DBTool” y “DBCreate” de Expresso puedan crear de manera automática las tablas necesarias cuando la aplicación es definida por primera vez para una determinada Base de Datos. Los objetos de Base de Datos están diseñados para ser el mecanismo de persistencia dentro de Expresso, y deberá tomar un beneficio completo de las facilidades que ofrece Expresso para estos, tales como “caching”, valores validos, chequeo de la integridad referencial declarada y la seguridad implícita en estos. Vea la página en la “Guía del Desarrollador de Aplicaciones Web” con Expresso para entender como usar estos objetos para sacarle todos sus beneficios.

- Crear cualquier objeto “Controlador” requerido

Los objetos “Controlador” deben ser usados para encapsular cualquier interacción con el usuario requerida por la aplicación, nuevamente la idea es tomar todos los beneficios de las cosas que son ofrecidas por el Marco de Trabajo y no reinventar nada que no sea necesario. Usted debe diseñar su Interfaz de Usuario de modo que sea independiente de la presentación, pensando en términos de Entradas y Salidas abstractas, para obtener los beneficios de los objetos Controladores. Esto le permite tomar todos los beneficios de las capacidades de Interfaces independientes de los usuarios en los proyectos Expresso y Expreso XML.

La página dedicada a los Controladores explica los detalles de los objetos “Controlador” que puede ayudarlo a crear y probar sus Controladores fácilmente.

- Crear cualquier objeto Proceso (“Job”) requerido

Los objetos Proceso pueden ser creados y usados para cualquier tarea que puede correr independiente del usuario, por ejemplo tareas que demoran mucho y que no requieren la intervención del usuario, o tareas que requieren un uso intensivo de las Base de Datos u otras fuentes de datos y que pudiera tomar algún tiempo.

Típicamente un objeto Proceso está diseñado para enviar algún tipo de notificación de su terminación al usuario que lo generó, los métodos que contiene el objeto Proceso permite realizar esto de una manera muy fácil.

La página que da detalles de los objetos Proceso (“Job”) le puede ayudar a crear e instalar sus Procesos de una forma efectiva.

CAPÍTULO 4. OBJETOS DE BASE DE DATOS

¿PORQUE USAR OBJETOS DE BASE DE DATOS?

Los objetos de Base de Datos proveen un número significativo de ventajas:

- Independencia de la Base de Datos
Los objetos de Base de Datos están contruidos desde su base para ser independientes de la Base de Datos, ya que ellos no descansan su funcionalidad en características específicas de una Base de Datos determinada. Esto permite construir una aplicación usando objetos de Base de Datos que puede ser llevada a otra plataforma de Base de Datos literalmente en segundos, proveyendo un potencial para una gran escalabilidad de la aplicación.
- Acceso Seguro y Dinámico
Cada interacción con un objeto de Base de Datos puede ser seguro, y los datos de la seguridad pueden ser fácilmente mantenidos a través de las capacidades que posee Expresso. Esto permite que la seguridad pueda ser actualizada desde cualquier lugar, y los cambios tengan efecto inmediatamente.
- Elimina sentencias SQL
Usando objetos de Base de Datos permite a una aplicación eliminar el uso de sentencias SQL mezclada con el código, lo que es difícil de mantener y crea dependencias con el sistema. Las aplicaciones simplemente interactúan con otros objetos Java, permitiendo que el diseño de la aplicación sea completamente orientado a objetos.
- Integridad y Consistencia de los datos
Insertar la lógica de acceso directamente en el objeto de Base de Datos, le permite que pueda tener las misma ventajas que cuando se usan procedimientos almacenados (“store procedure”) lo que de manera independiente a la Base de Datos. La integridad referencial se convierte de igual forma independiente de la Base de Datos, y las complejas relaciones entre los objetos de Base de Datos se convierten portables diferentes plataformas de Base de Datos. Por ejemplo, las reglas de negocio pueden ser

integradas en los objetos de Base de Datos, a fin de que todas las aplicaciones que acceden a los objetos estén seguras de seguir las reglas establecidas.

- Campos Virtuales

Con la adición de campos virtuales (Ej. campos calculables) a los objetos de Base de Datos, los datos los cuales están almacenados a través de varias tablas pueden ser tratados por la aplicación como un objeto único. Por ejemplo, si un campo virtual en un encabezado de una factura da el total de la factura, la aplicación no tiene la necesidad de tratar con los detalles del objeto Factura para obtener la información sobre el total de la factura, este es un detalle de la implementación que es ocultado por el objeto de Base de Datos Factura.

- Campos específicos

Para Tablas largas (Ej. una Tabla con muchos campos), es frecuentemente mas eficiente hacer las preguntas (“queries”) recuperando solo los campos necesarios: Ej. algo como “SELECT a,b,c FROM...” en lugar de “SELECT * FROM”. DBObjects soportan esta facilidad, a través del método “setFieldsToRetrieve(cadena de caracteres)”, el cual toma una lista delimitada de campos que deben ser recuperados para subsiguientes preguntas.

- Componentes empresariales del tipo entidad(Entity EJB) sin usar EJB

Los objetos de Base de Datos proveen una funcionalidad que es similar a una extensión de la especificación de los componentes Java empresariales EJB del tipo entidad, pero no requiere que el uso de los EJB, con la complejidad que posee un Servidor de Aplicaciones. Donde los componentes empresariales EJB son usados, los objetos de Base de Datos pueden ser implementados fácilmente como componentes empresariales del tipo entidad, permitiendo escalarlos hacia arriba y hacia abajo en tamaño y complejidad.

CAPACIDADES DE OBJETOS DE BASE DE DATOS

Los objetos de Base de Datos (mas específicamente, objetos “SecuredDBObject”) tienen un número de capacidades innatas las cuales pueden ser usadas por la aplicación cliente. Algunas de estas son descritas a continuación:

NÚMERO MÁXIMO DE ARTÍCULOS

El método `setMaxRecords()` puede ser usado para decirle al objeto de Base de Datos que las subsiguientes llamadas para la recuperación (tales como `searchAndRetrieve`) solo deben retornar un número determinados objetos, esto puede ser útil cuando se necesitan mostrar solamente los n primeros artículos de una pregunta(query) a la Base de Datos, u otra función para “preguntar” que deba prevenir la recuperación de un conjunto muy largo de artículos a ser procesados.

INFORMACIÓN DEL ESTADO

Los objetos de Base de Datos pueden decir al cliente su propio estado a través de una llamada al método `getStatus()`, para que el cliente determine si son necesaria actualizaciones, si el dato ha sido borrado o si este necesita ser almacenado, etc.

CAPACIDAD DE MÚLTIPLES BASE DE DATOS

Un objeto de Base de Datos puede ser definido para ser accedido desde un almacén de datos alternativo, Ej. otra Base de Datos en el mismo servidor, u otra Base de Datos en un servidor diferente. Esto es muy apreciado en escenarios de almacenes de datos (data warehousing), o donde los datos de control para Expresso están almacenados en una Base de Datos y los datos de la aplicación están almacenados en otra Base de Datos.

INFORMACIÓN SOBRE LA TABLAS

Los objetos pueden suministrar a la aplicación el nombre de la Tabla de la Base de Datos donde estos están almacenados, esto permite comandos SQL donde se necesite sin sacrificar la independencia de los objetos de Base de Datos de las Tablas y Base de Datos.

RASTREO (LOGGING) DE CAMBIOS

Los objetos de Base de Datos pueden ser definidos para que automáticamente rastreen y sigan la pista a los cambios en sus datos, proveyendo la habilidad de una auditoria automática para los datos críticos, esto sin esfuerzos adicionales de desarrollo.

BÚSQUEDAS Y CONJUNTOS DE RESULTADOS

Los objetos de Base de Datos pueden ser usados en modo agregado (“aggregate mode) donde un único objeto de Base de Datos (“DBObject”) representa una lista de artículos, u otros objetos de Base de Datos. Esto permite que las búsquedas y sus resultados puedan ser manipulados. El conjunto de resultados puede ser ordenado por cualquier campo, y las búsquedas pueden ser hecha sobre cualquier campo, incluyendo el uso de comodines y criterios de rango. El conteo de la cantidad de artículos recuperados puede ser obtenido sin acceder al conjunto de resultados.

Es posible también retornar solamente la llave(key) de los artículos recuperados, para reducir de esta forma el tamaño de los datos que van a ser manipulados.

INTEGRIDAD REFERENCIAL DECLARATIVA

Los objetos de Base de Datos pueden definir las Integridad Referencial dentro de ellos mismos con la simple llamada a métodos de estos. Estas reglas de Integridad Referencial son entonces verificadas para cualquier operación con estos objetos. Es posible de igual forma implementar actualizaciones y borrados en cascada basados en las reglas de Integridad Referencial.

ADICIONANDO ARTÍCULOS

Adicionar un nuevo artículo con un objeto de Base de Datos es tan fácil como llenar los campos con sus valores previamente validados y llamar al método `add()` para almacenar el nuevo objeto. Las reglas de las llaves(Key) son tenidas

en cuenta de manera automática. Las operaciones de Actualizar y Borrar son igualmente sencillas de implementar.

MAPEO DE TIPOS

Todos los tipos `java.sql.Types` son soportados, y puede ser mapeados a cualquier tipo de datos de cualquier Base de Datos para tener una completa independencia del Sistema de Administración de la Base de Datos. Los campos pueden ser accedidos como cualquier tipo de dato Java con conversión automática. Métodos especiales hacen la manipulación de los campos fecha y de horas muy fácil.

CAMPOS DE MÚLTIPLES VALORES

Los campos pueden ser especificados como de Múltiples Valores, en tales casos los valores pueden ser retornados con una llamada al método `getValues()`. Estos valores pueden ser almacenados en zonas temporales (Cache) para mejoras en el rendimiento (a través de la clase `ConfigManager`), y pueden ser recuperados desde otros objeto de Base de Datos, o contados según se necesite. Existen métodos especiales que hacen el chequeo de validación para ambos tipos de campos, los normales y de Múltiples Valores muy fácil. La validación es entonces aplicada para todo los accesos a los objetos de Base de Datos, asegurando así la validez de todos los datos.

DESCRIPCIÓN DE CAMPOS

Los objetos de Base de Datos pueden retornar una información adicional de sus campos, incluyendo una descripción larga (otra a parte del nombre de la Base de Datos) y otras informaciones.

TABLAS GENERADAS AUTOMÁTICAMENTE

Un objeto de Base de Datos puede crear automáticamente las tablas que este requiere en la Base de Datos que las contiene, eliminando la necesidad de ejecutar manualmente guiones SQL para la creación de la estructura de la Base de

Datos. Adicionalmente se suministran herramientas que pueden ser usadas para llevar a cabo ingeniería inversa y generar el código fuente Java para objetos de Base de Datos que mapea las tablas en las Bases de Datos.

O

USANDO LOS OBJETOS DE BASE DE DATOS

Para ganar en mayor flexibilidad en sus aplicaciones con Expresso, es importante usar los objetos de Base de Datos para todos sus accesos al mecanismo de persistencia, y eliminar el uso de comandos SQL directos siempre y cuando sea posible. Este documento lo ayudará a seleccionar los mejores patrones de uso para su propósito particular y explica las ventajas de cada selección.

El uso más directo de los objetos de Base de Datos es escribir una clase que extienda la clase `com.jcorporate.dboj.DBObject`. Su clase entonces necesita implementar unos cuantos métodos simples para describir su relación con la Tabla en la Base de Datos. Una vía directa aquí es usar la herramienta DBTool para generar el código para usted a partir de la Base de Datos por métodos de ingeniería inversa. ver la documentación para más detalles sobre DBTool.

Una vez que usted ha escrito sus objetos de Base de Datos (DBObject) usted puede inmediatamente comenzar a usarlos en su programa para acceder a la Tabla destino.

Un DBObject deberá implementar al menos el constructor `DBObject()` y el método `setupFields()`. En el método `setupFields()`, un número de llamadas a métodos son usadas para establecer las relaciones de estos DBObject con la Base de Datos y con la Tabla o Tablas especificadas. Como mínimo el objeto DBObject debe llamar a los métodos `setTargetTable()` para especificar una tabla en la Base de Datos, y uno o más llamadas al método `addField()` para especificar los campos en la Tabla. Por ejemplo, digamos que estamos definiendo un objeto con información sobre los clientes; la Tabla deberá tener un identificador de cliente único, un nombre y un tipo de cliente. Usted podrá nombrar el objeto DBObject como “Cliente” y especificar un método `setupFields()` como se muestra a continuación:

```
1. public void setupFields() throws DBException {
2.     setTargetTable ("CLIENTE");
3.     addField("ClienteID", "int", 0, false,
"Identificación del Cliente");
```

```

4.    addField("ClienteNombre", "varchar", 80, false,
"Nombre del Cliente");
5.    addField("ClienteTipo", "char", 2, false, "Tipo de
Cliente");
6.    addKey("ClienteID");
7. }

```

Línea 1: declara el método el cual debe estar bien escrito para que su DBObject compile correctamente.

Línea 2: especifica la Tabla destino o la Tabla en la Base de Datos estará asociada con este objeto de Base de Datos. Note que no hay ningún nombre de Base de Datos anexado con el nombre de la Tabla, esto permite que el mismo DBObject sea usado en cualquier Base de Datos apropiada. El contenido de la aplicación de Espresso especificará la Base de Datos a la cual conectarse en tiempo de corrida.

Línea 3: Especifica el primero de los campos, o columnas de este objeto de Base de Datos y de la correspondiente Tabla.

Aunque es posible para la Tabla de la Base de Datos tener columnas que no estén especificadas en el objeto DBObject, esto no es recomendable.

En la llamada al método `addField()` nosotros podemos especificar los siguientes parámetros:

- *Nombre de la columna (Column Name)*: Este es el nombre por el cual esta columna es referida dentro del objeto de Base de Datos y la Base de Datos en si.

Este nombre debe corresponderse con el nombre de la columna dentro de la Base de Datos, pero tenga en mente que el objeto DBObject puede ser usado para crear la tabla automáticamente la cual exploraremos luego. Nosotros recomendamos la convención de nombres mostrada aquí, con la primera letra en mayúscula y una mayúscula al comienzo de cada nueva palabra, sin espacios ni guiones entre estas palabras. Nuevamente, no se han usado prefijos.

Sea cuidadoso cuando nombra una columna de no usar palabras reservadas de

cualquier Sistema de Base de Datos, aún cuando una palabra en particular no esta reservada en el Sistema de Base de Datos que usted seleccionó, es muy prudente que no restrinja la portabilidad de su aplicación. El nombre usado debe ser único dentro del objeto DBObject usado pero no tiene que ser único a través de todas las Tablas en la Base de Datos. En otras palabras, usted puede tener solamente una columna “ClienteID” en este DBObject pero otros DBObject pueden usar ese mismo nombre de columna.

- Tipo de Dato (Data Type): Las cadenas de caracteres que usted especifique en este parámetro deben especificar el tipo de dato de Expresso usado por este campo.

Este puede ser o no el mismo tipo de datos usado por su Sistema de Base de Datos, los tipos de datos de Expresso son mapeados en tiempo de corrida con el tipo apropiado del motor de Base de Datos. Usted puede definir nuevos tipos para Expresso y mapear estos para tomar ventajas de las capacidades de un Sistema de Base de Datos en particular; aunque esto puede limitar la portabilidad y por esta razón no lo recomendamos.

- Tamaño del Campo (Field Size): Algunos tipos de datos requieren un tamaño de campo o longitud máxima para ser definidos. Otros, tales como “int” generalmente no. Si un tamaño de campo es necesario debe ser especificado en este parámetro de lo contrario debe ponerse 0. Existe un método adicional para especificar los tamaños de los campos para campos que necesitan especificar dos tamaños; Ej. los campos “float”, donde el primer tamaño es la longitud del campo y el segundo es el número de lugares decimales.
- Permitir Nulos (Allow Nulls): Este parámetro booleano indica si el campo puede guardar valores nulos(null): true indica que permite nulos y false indica que no acepta valores nulos.
- Descripción del campo (Field Description): Este es un término corto(entendible) o frase que describa el campo, frecuentemente usado como encabezado de columna, listado o reporte. Para un mejor soporte de idiomas, esta descripción deberá ser especificada como una llamada al método `getString()` del objeto “Schema” de la aplicación el cual puede seleccionar

el nombre apropiado en el idioma correcto en tiempo de corrida. ver para más detalles la documentación sobre internacionalización.

Líneas 4 y 5: en el listado se especifican campos adicionales en nuestra Tabla, aunque el orden en que los campos son adicionados es irrelevante, esto no afecta la secuencia en la cual son listada cuando una Tabla es creada desde el objeto DBObject o cuando se usan los componentes de Expresso para listar los artículos de la Tabla, así que algún orden lógico debe ser seguido.

Línea 6: especifica el campo que fungirá como llave primaria. Usted debe especificar al menos un campo como llave primaria y usted puede usar mas de uno haciendo repetidas llamadas al método `addKey()` con el nombre de los campos que se usarán como llave primaria. Los campos que sean usados como llave primaria no pueden ser nulos.

Ahora que usted tiene creado el objeto DBObject, usted puede comenzar a usar este en sus programas. Nosotros examinaremos cada una de las operaciones mas comunes con los DBObjects.

RECUPERANDO ARTÍCULOS

Para recuperar un determinado DBObject correspondiente a una fila en particular de una Base de Datos, usted deberá inicializar el objeto en su programa:

```
import com.sucompania.dboj.Cliente;  
.  
.  
.  
Cliente unCliente = nuevoCliente();
```

Esto inicializa una instancia de Cliente, llamada unCliente. Ahora para recuperar un Cliente en especifico, nosotros debemos especificar un valor para el campo llave (o los campos si hay más de un campo llave).

```
unCliente.setField("ClienteID", "1");
```

Esta especifica el valor "1" para los campos ClienteID. Note que debemos especificar el campo como una cadena de caracteres, hay también métodos `setField()` para otros tipos de datos, pero podemos usar siempre cadenas ya que serán convertidas al tipo de dato apropiado.

Todo lo que hemos hecho hasta ahora es en memoria, no habrá acceso a la Base de Datos hasta que hagamos:

```
unCliente.retrieve();
```

Este accederá a la Base de Datos (o en su lugar a la "cache") y recuperará los artículos que cumplen con la llave especificada. Ahora nosotros podemos acceder a otros campos en el artículo con el método `getField(Nombre del Campo)`;

```
System.out.println(getField("Nombre"));
```

Este código imprimirá el Nombre del primer Cliente.

¿Qué pasa si no conocemos el ID del Cliente pero queremos buscar clientes basados en otros criterios? El método `retrieve()` requiere que los campos que forman la llave primaria debe cada uno tener un valor especificado, de lo contrario será lanzada una excepción. La mayoría de los métodos de los objetos `DBObject`s pueden lanzar excepciones del tipo `DBException`, usted deberá encerrar el código de arriba en un bloque `try/catch` para manipular la excepción o su método deberá lanzar una excepción del tipo `DBException`).

Otros métodos para recuperar cuando no se puede aplicar la búsqueda por la llave primaria es el siguiente:

```
/* Borra cualquier valor en los campos */
unCliente.clear();
unCliente.setField("Nombre", "Juan");
if (unCliente.find()) {
    System.out.println("¡Se encontró a Juan!");
}
```

```
}
```

El método `find()`, si tiene éxito la búsqueda retorna verdadero (`true`) y llena el objeto `DBObject` con los valores de los campos para el primer artículo recuperado. Si el método `find()` no encuentra ningún artículo este retorna falso (`false`) y los campos no son llenados.

RECUPERANDO MULTIPLES ARTÍCULOS

`DBObject`s también pueden ser usados para recuperar conjuntos completos de artículos en lugar de uno cada vez. Por ejemplo, digamos que queremos realizar algún tipo de procesamiento sobre todos los Clientes del tipo “AB” en nuestra Base de Datos. El código usado pudiera ser:

```
1. Cliente listaCliente = new Cliente();
2. Cliente unCliente = null;
3. listaCliente.setField("Tipo", "AB");
4. for (Enumeration e =
listaCliente.searchAndRetrieve().element(); e.hasMoreElements();
) {
5.     unCliente = (Cliente) e.nextElement();
6.     /* haga aquí lo que desee con unCliente */
7. }
```

Ahora examinaremos una a una las líneas de este código:

Línea 1: Inicializamos un nuevo `DBObject` Cliente, `listaCliente`. En lugar de tratar con un solo artículo Cliente, este `DBObject` es usado para recuperar una lista completa de objetos Cliente.

Línea 2: Declaramos una segunda instancia de Cliente para almacenar los Clientes individuales que recuperemos. No necesitamos inicializar esta instancia, por lo que la definimos como “null” por ahora.

Línea 3: Aquí suministramos el criterio de búsqueda para el objeto `listaCliente`, especificando que estamos buscando artículos donde el Tipo de Cliente sea igual a “AB”.

Línea 4: Esto es una línea compleja. Estamos comenzando un ciclo “for” para crear una enumeración e, inicializar esta con el resultado obtenido del método `searchAndRetrieve()` del objeto `listaCliente`. Esta línea pudiera reescribirse de la siguiente forma:

```
Vector v = listaCliente.searchAndRetrieve();
Enumeration e = v.elements();
while (e.hasMoreElements()) {
    unCliente = (Cliente) e.nextElement();
}
```

Línea 5: Aquí obtenemos cada objeto `Cliente` recuperado de manera individual, rehusando el objeto `unCliente` para guardar cada artículo. La primera vez que se ejecuta el ciclo esta guardará el primer artículo de `Cientes` que cumple la condición. La segunda vez el segundo y así sucesivamente. Usted podrá usar la información que se encuentra en `unCliente` para realizar cualquier procesamiento en la línea 6.

De esta forma usted puede procesar una larga lista de artículos , mientras el acceso a la Base de Datos se hace con una búsqueda eficiente. Esto minimiza el acceso a la Base de Datos logrando un gran eficiencia.

MANIPULANDO DATOS ORDENADOS

Si es necesario procesar los artículos en una secuencia específica, otra versión de `searchAndRetrieve` puede ser llamada con un parámetro para especificar campos ordenados. Por ejemplo:

```
listaCliente.searchAndRetrieve("Nombre");
```

Recuperar los artículos especificados en orden ascendente por el Nombre del `Cliente`. Para especificar por orden descendente, usted debe especificar “Desc” al final de la cadena de caracteres de la siguiente forma:

```
listaCliente.searchAndRetrieve("Nombre Desc");
```

Usted también puede especificar orden por múltiples campos especificando uno mas de estos separados por el símbolo “|” de la siguiente forma:

```
listaCliente.searchAndRetrieve("Nombre|Tipo");
```

Esta línea especifica que los artículos serán recuperados por el orden de los Nombres pero además los clientes con el mismo Nombre deben estar ordenados por su ID.

Si no se especifica un criterio para la búsqueda *no se puede asegurar* que los artículos serán recuperados en un orden determinado. Frecuentemente los artículos serán recuperados en el orden en que estos son insertados, o por el orden de la llave, pero usted no puede contar con que este sea el caso. Si usted necesita recuperar los artículos en un orden específico, pregunte por estos especificando el parámetro correspondiente al método `searchAndRetrieve()`.

MANIPULANDO GRANDES CONJUNTOS DE DATOS

Como practica general usted deberá especificar su criterio de búsqueda tan exacto como sea posible, para recuperar así solamente los artículos que son necesarios para su tarea. Usted puede definir criterios en cuantos campos usted necesite y todos los criterios serán combinados para obtener el conjunto de artículos resultantes. Por ejemplo:

```
listaCliente.setFields("Nombre", "Juan");  
listaCliente.setFields("Tipo", "AB");
```

Esto especifica que usted desea recuperar los artículos cuyos clientes tengan por Nombre “Juan” y sean del tipo “AB”.

USANDO RANGOS Y COMODINES

Usted puede especificar mucho más que criterios estrictos para su búsqueda. Los comodines y los rangos pueden ser usados también, la sintaxis exacta depende de su motor de Base de Datos en uso. ver el fichero de Propiedades (Properties file) para una documentación detallada sobre la definición de los caracteres de comodines para su Base de Datos de manera específica.

Por ejemplo:

```
listaCliente.setField("Nombre", "A%");
```

Especifica una búsqueda para todos los clientes que el Nombre comience con la letra "A".

```
listaCliente.setField("Nombre", "[A-M]%");
```

Especifica una búsqueda para todos los clientes cuyo Nombre comience con las letras de la "A" a la "M".

```
listaCliente.setField("ClienteID", "BETWEEN 1 AND 20");
```

Especifica una búsqueda para clientes con un identificador "ClienteID" entre los números 1 y 20.

Si usted especifica criterios de búsqueda cuidadosamente el número de artículos a procesar podría reducirse y agilizar su aplicación.

Usted también puede especificar que su búsqueda retorne un número máximo de artículos. Esto puede ser útil cuando usted necesita por ejemplo obtener solamente los 100 primeros artículos para el criterio de búsqueda dado.

```
listaCliente.setMaxRecords(100);
```

Esta línea dice que el método `searchAndRetrieve()` recuperará un máximo de 100 artículos, aún cuando otros también cumplan la condición.

Usted puede usar el método `count()` para ver cuantos artículos cazaron con la búsqueda sin recuperar realmente todos los artículos.

```
listaCliente.setField("Nombre", "A%");
int ct = listaCliente.count();
System.out.println("Hay " + ct + "clientes que su nombre
comienza con A");
```

Si es necesario procesar un conjunto muy largo de artículos, usted puede usar una técnica de banderas (flagging technique). Como se muestra en el código de abajo donde asumimos que todos los artículos tienen un campo “Procesado” que está definido inicialmente a “N”:

```
Cliente listaCliente = new Cliente();
Cliente unCliente = null;
listaCliente.setField("Procesados", "N");
listaCliente.setMaxRecords(100);

boolean masArticulos = true;

while (masArticulos) {
    for (Enumeration e =
listaCliente.searchAndRetrieve().elements(); e.hasMoreElements();)
    {
        unCliente = (Cliente) e.nextElement();
        /* Cliente procesado */
        unCliente.setField("Procesado", "Y");
        unCliente.update();
    }
    if (listaCliente.count() == 0) {
        masArticulos = false;
    }
}
```

El código de arriba procesará todos los clientes recuperando solo 100 cada vez.

ACTUALIZANDO ARTÍCULOS

Como usted puede ver en el código anterior, actualizar artículos en la Base de Datos es muy fácil. Haciendo una llamada al método `update()` en el objeto usted efectúa los cambios y son escritos hacia la Base de Datos. Solamente un único artículo es actualizado a la vez y siempre es más seguro recuperarlo primero. ver mas abajo la sección Control de Transacciones para información sobre operaciones de Hacer(`commit`) y Deshacer(`rollback`) Actualizaciones en Bloques o simplemente Actualizaciones.

ADICIONANDO NUEVOS ARTÍCULOS

Adicionar nuevos artículos es tan fácil como actualizarlos: simplemente llene los campos del artículo (especialmente la llave primaria) y haga una llamada al método `call()`. Si un artículo existe con la misma llave primaria, una excepción será lanzada.

BORRANDO ARTÍCULOS

Como en la adición, el borrado requiere que el objeto `DBObject` tenga valores en sus campos al menos para la llave primaria. La llamada:

```
unCliente.delete();
```

elimina el artículo especificado por el objeto `unCliente` (ver la sección sobre Integridad Referencial).

La operación de borrado, tal como la de actualizar y adicionar, afecta a un solo artículo a la misma vez.

CLÁUSULAS “WHERE” PERSONALIZADAS

Si usted necesita especificar relaciones del tipo “or”, u otras condiciones especiales, usted puede usar las habilidades de los `DBObject`s para definir cláusulas “where” personalizadas para la pregunta(query) SQL a ser ejecutada. Por ejemplo:

```
listaCliente.setCustomWhereClause("Tipo= \"AA\" \ OR  
Tipo=\"AB\"");
```

Las cláusulas “where” personalizadas solo afectan la próxima pregunta(query) que se corra y serán limpiadas una vez que se ejecuten por razones de seguridad.

ESPECIFICANDO BASE DE DATOS

Hasta ahora, todos los ejemplos que hemos visto asumen que los objetos DBObject están accediendo a la Base de Datos/Contexto definido por defecto cuando Expresso es instalado. Los objetos DBObjects pueden acceder a cualquier Base de Datos/Contexto haciendo uso del método `setDBName(Cadena de Caracteres)`. Este método recibe como parámetro el código de una Base de Datos/Contexto, el cual es el igual al nombre para este contexto dentro del fichero `.property` en Expresso. Por ejemplo:

```
listaCliente.setDBname("oracle");
```

dice que el objeto DBObject `listaCliente` deberá usar el contexto “oracle”, el cual es de suponer que ha sido definido para conectarse con un Sistema de Base de Datos Oracle.

La mayoría de los otros objetos de Expresso permiten obtener la Base de Datos/Contexto en uso a través de una llamada al método `getDBName()`, los objetos Controladores por ejemplo hacen esto. Así que un objeto DBObject siendo usado dentro de un objeto Controlador simplemente hace:

```
listaCliente.setDBname(getDBName());
```

para especificar que el va a usar la Base de Datos/Contexto del Controlador, esto hace que el objeto Controlador completo sea portable entre Base de Datos.

Es una buena practica siempre definir la Base de Datos/Contexto del objeto DBObject justo en el momento de inicializarlo.

SEGURIDAD Y OBJETOS SECUREDDBOBJECT

Un objeto DBObject puede ser accedido por cualquier programa y leer y escribir para cualquier Base de Datos que el usuario especifique en el fichero de propiedades (como el usuario de la Base de Datos) que este tienen permiso para acceder a esta, frecuentemente a la Base de Datos completa. Para definir una forma estándar e independiente de la Base de Datos de especificar los permisos para la Base de Datos, una extensión del objeto DBObject esta disponible y se denomina SecuredDBObject.

Extendiendo SecuredDBObject en lugar de DBObject, sus objetos DBObject pueden tener la facilidad de hacer uso de las capacidades de seguridad.

Usando las funciones de administración que posee Expresso, usuarios autorizados (tales como administradores del sistema) organizados en grupos, se les puede otorgar 4 de los posibles permisos en cada SecuredDBObject: Adicionar, Actualizar, Buscar, Borrar. Usted puede especificar, con el método `setUser(cadena de caracteres)`, el usuario que esta corriendo su programa y el objeto SecuredDBObject verificará cada solicitud de operación contra la información de seguridad. Por ejemplo:

```
Cliente unCliente = new Cliente();
unCliente.setUser("Gaspar");
unCliente.setField("ClienteID", "1");
unCliente.retrieve();
```

el código de arriba tendrá éxito solo si el usuario "Gaspar" tiene permisos de búsqueda ("Search") en el objeto Cliente del tipo SecuredDBObject. Si no lo tiene la llamada al método `retrieve()` lanzará una excepción de seguridad (Security Exception).

Usted también puede chequear los permisos antes de la llamada a un método, por ejemplo:

```
listaCliente.isAllowed("S");
```

retornará verdadero(true) si el usuario actual le esta permitida la búsqueda “S” (Search), y falso(false) si no lo tiene. A, U, y D pueden usarse para chequear los permisos de Adicionar “A” (Add), Actualizar “U” (Updated) y Borrar “D” (Delete).

USANDO DBMAINT

Una vez que usted a escrito el código para el objeto SecuredDBObject, usted puede inmediatamente tomar las capacidades de administración automática de la Base de Datos de Expresso a través del Controlador DBMaint. Este Controlador le permite a usted otorgarle a sus usuarios las capacidades de Adicionar (Add), Actualizar (Update), Borrar (Delete) y Buscar (Search) para cualquier SecuredDBObject, sin escribir ningún código.

El “Servlet” trabaja con cualquier SecuredDBObject especificado como parámetro, y puede ejecutar un número de comandos diferentes. Por ejemplo:

```
/DBMaint.do?dbobj=com.yourcompany.dboj.Customer&state=List
```

el cual listará todos los artículos (sujeto al tamaño de página especificado por Expresso y que se discutirá más adelante) en la Tabla correspondiente para el objeto DBOBJECT “Cliente”. En este caso se asume que el usuario que esta solicitando la búsqueda tiene los permisos correspondientes sobre el objeto Cliente.

La lista de opciones para “state”, el cual es extensible mediante la adición de nuevos objetos al paquete

com.jcoporate.expresso.services.controller.dbmaint, incluye:

- *Search*: Presenta una forma para preguntas(query) al usuario para que entre el criterio para la búsqueda de artículos.
- *Add*: Presenta una forma en blanco al usuario para que entre un nuevo artículo.

- *Update*: Presenta un artículo existente para que se actualizado por el usuario. Requiere parámetro de la llave del artículo como parámetro.
- *List*: Lista los artículos (o el resultado de la pregunta(query) actual) y permite al usuario seleccionar un artículo para su actualización.

Todas las formas presentan iconos al usuario para moverse de un modo a otro, haciendo que el DBMaint sea una completa aplicación de administración de la Base de Datos para sus objetos DBObject.

Las propias páginas de administración de Expresso son un buen ejemplo del uso de DBMaint.

CONTROL DE TRANSACCIONES

Cuando usted escribe aplicaciones muy sofisticadas usted necesitará en ocasiones un control de transacciones, lo que consiste en la habilidad de realizar varias operaciones sobre la Base de Datos a la vez y que todas deben ser exitosas o de lo contrario no debe ser realizada ninguna.

Hasta ahora nuestros ejemplos con objetos de Base de Datos confiaban en su habilidad de administrar sus propias conexiones a Base de Datos. A parte de especificar correctamente la Base de Datos/Contexto, hemos permitido a nuestros objetos de Base de Datos solicitar conexiones de la piscinas de conexiones y liberar esta de manera automática.

Para controlar una Transacción se requiere especificar una conexión a Base de Datos en particular, lo cual nos permitirá el acceso al método `commit()` del objeto relacionado con la conexión.

```

1. DBConnectionPool miPiscina = null;
2. DBConnection miConexion = null;
3. try {
4.     miPiscina =
DBConnectionPool.getInstance(getDBName());
5.     miConexion = miPiscina.getConnection();
6.     miConexion.setAutoCommit(false);

```

```

7.     Cliente unCliente = new Cliente(miConexion);
8.     Factura unaFactura = new Factura(miConexion);
9.     /* llena los campos de la Factura */
10.    unCliente.setField("Balance", nuevoBalance);
11.    unaFactura.add();
12.    unCliente.update();
13.    miConexion.committ();
14. } catch (DBException de) {
15.     if (miConexion != null) miConexion.rollback();
16.         throw newDBConnection(de);
17. } finally {
18.     if (miPiscina != null) {
19.         miPiscina.release(miConexion);
20.     }
21. }

```

Línea 1 y 2 declaran la piscina de conexión y los objetos de conexión que serán usados, los cuales son declarados fuera del bloque “try/catch” para que estén disponibles en los bloques “catch” y “finally”.

Línea 3 comienza el bloque “try”. Todas las operaciones en el bloque deben ser exitosas o no deben ser realizadas como un todo. Para los propósitos de nuestro ejemplo, hemos asumido que estamos creando una nueva instancia y grabando el nuevo balance de Cliente cuando la factura es adicionada. Si la factura no puede ser adicionada de manera satisfactoria, el Cliente no debe ser actualizado, y viceversa, sino el balance del Cliente no concordará con el total de Facturas para este Cliente.

Línea 4 Aquí estamos solicitando una solicitud al objeto de Piscina de Conexión correspondiente desde la clase Piscina de Conexión (DBConectionPool) para lo que pasamos como parámetro el nombre de la Base de Datos/Contexto.

Línea 5 Solicitamos una conexión de la piscina de conexiones y ...

Línea 6 decimos que esta conexión no deberá realizar las actualizaciones de manera automática, y en lugar de eso debe esperar a que el método `commit()` sea llamado.

Línea 7 y 8 Aquí instanciamos los objetos de Base de Datos Cliente y Factura, pasando el objeto de conexión que deseamos usar. El objeto usará la conexión pasada a este en lugar de solicitar su propia conexión.

Línea 9 y 10 Hemos asumidos las líneas que deberá ser puestas por usted, que puede ser una llamada a un método, para llenar los campos de los objetos de Factura y calcular el nuevo balance del Cliente. Asumimos también que el nuevo balance está almacenado en la variable “nuevoBalance”. Este balance es definido en el objeto Cliente en la línea 10 (asumimos que un campo “Balance” ha sido definido en nuestro objeto Cliente).

Línea 11 y 12 son las líneas de código que tienen que ver con la lógica de la transacción. El Cliente es actualizado y la nueva Factura es almacenada en la Base de Datos. Si una de estas operaciones fallan, ellas lanzarán una excepción del tipo `DBException` y la ejecución continuará en la línea 15.

Línea 13 llama al método `commit()` en la conexión la cual confirma que la operación debe ser ejecutada sobre la Base de Datos.

Línea 15 y 16 manejan la situación donde una de las actualizaciones o alguna otra operación sobre la Base de Datos ha fallado. El método `rollback()` asegura que no se hagan operaciones a medias sobre la Base de Datos y la cláusula “`throw`” hace que se relance la excepción que ha ocurrido, que permite que el manipulador de errores de Expresso la trate de la manera mas apropiada.

Línea 18 y 19 Son ejecutadas en cualquier caso, tanto si la transacción tuvo o no éxito, y son extremadamente importantes ya que la conexión es liberada hacia la piscina de conexiones para ser usadas por otros objetos. Sin estas líneas, la conexión se mantiene activa siempre y la piscina de conexiones rápidamente se quedará sin conexiones disponibles.

OBJETOS MULTIDBOBJECTS

Frecuentemente es necesario tratar con relaciones de unión(Join) entre las tablas en las Base de Datos relacional, el objeto `MultiDBObject` existe para manipular estas. Si un objeto `DBObject` es análogo a una Tabla, un objeto `MultiDBObject` es análogo a una Vista (tablas unidas).

Muchas de las mismas operaciones disponibles para los objetos `DBObject` están disponibles para los objetos `MultiDBObject`s incluyendo `searchAndRetrieve()`, `clear()`, `setField`(con argumentos diferentes), etc.

Cuando definimos sin embargo un objeto `MultiDBObject`, usted no puede usar el método `addField()`, en su lugar usted adiciona objetos `DBObject` completos a través de una llamada al método `addDBObject()`. También a diferencia de los objetos `DBObject`, el objeto `MultiDBObject` no es una clase abstracta, usted puede instanciar directamente objetos del tipo `MultiDBObject`s, en lugar de tener que hacer una clase que herede de esta. Por ejemplo:

```
1. MultiDBObject miMultiple = new MultiDBObject();
2. miMultiple.setDBName(getDBName());
3.
miMultiple.addDBObject("com.jcorporate.expresso.services.dbobj
.UserDBObject", "usuario");
4.
miMultiple.addDBObject("com.jcorporate.expresso.services.dbobj
.UserGroup", "grupos");
5.
miMultiple.addDBObject("com.jcorporate.expresso.services.dbobj
.GroupMembers", "miembros");
6. setForeignKey("miembros", "NombreUsuario", "usuario",
"NombreUsuario");
7. setForeignKey("miembros", "NombreGrupo", "grupo",
"NombreGrupo");
8. MultiDBObject unMultiple = null;
9. miMultiple.setField("usuario", "NombreUsuario", "Juan");
10. System.out.println("Usuario Juan pertenece a los
siguientes grupos:");
11. for (Enumeration e =
miMultiple.searchAndRetrieve().elements();
e.hasMoreElements()); {
12. unMultiple = (MultiDBObject)e.nextElement();
```

```

13. System.out.println(unMultiple.getField("grupo",
"descripcion");
14. }

```

Las líneas largas han sido desplegadas en múltiples línea para ganar en claridad, pero no es necesario en su aplicación.

Línea 1 Se instancian los objetos MultiDBObject que serán preguntados(query).

Línea 2 Define el Base de Datos/Contexto de este objeto MultiDBObject para el Base de Datos/Contexto de cualquier objeto que nosotros estemos usando desde dentro del objeto MultiDBObject, por ejemplo si estamos usando un objeto Controlador (“Controller”), el método `getDBName()` tendrá acceso al nombre de la Base de Datos/Contexto actual, asegurando que el objeto MultiDBObject esta operando dentro del mismo contexto.

Línea 3 hasta la 5 especifica los objetos DBObject que este objeto contiene y especifica un nombre corto para los objetos por ejemplo, `com.jcorporate.expresso.services.dbobj.UserDBObj` se referencia por su nombre corto “usuario”.

Línea 6 y 7 establece la relación entre los 3 objetos especificando un objeto de llave foránea y una llave primaria relacionada en otro objeto. Estas dos relaciones existen en nuestro ejemplo, El campo “NombreUsuario” del objeto “miembros” deberá casar con el campo “NombreUsuario” del objeto “usuario”, y el campo “NombreGrupo” del objeto “miembros” deberá casar con el campo “NombreGrupo” en el objeto “grupo”.

Línea 8 declara un objeto MultiDBObject nuevo para guardar el resultado de cada pregunta(query) tal como los objetos DBObjects.

Línea 9 Define el criterio de búsqueda, usted notará que el nombre corto para el objeto, el nombre del campo y el valor deben ser especificados. Esto permite que el objeto MultiDBObject conozca cual campo en cual objeto debe asignársele el valor dado.

Línea 11 hasta la 13 recupera el resultado de la pregunta(query), como los objetos DBObject, excepto que usted notará que la llamada al método `getField()` también tomará el nombre corto del objeto para recuperar el valor del campo

desde este. Los 3 objetos DBObjects tienen los valores de sus campos llenos para cada elemento resultante retornado.

Usted también puede usar los objetos MultiDBObject extendiendo este en su propio objeto MultiDBObject, implementando el método `setupFields()` como en los objetos DBObjects pero en lugar de llamar al método `addField()` usted especifica llamadas al método `addDBObject()` y llamadas al método `setForeignKey()`.

Los objetos MultiDBObjects son actualmente de solo lectura. No permite operaciones de actualización (Update) pero en breve tiempo serán adicionadas.

CAMPOS VIRTUALES

Los objetos DBObject necesitan en ocasiones tener campos que no son almacenados en la Tabla destino. Ellos pueden ser calculados (tales como Total de la Factura) o recuperados desde otras tablas (tales como Tablas para búsqueda de códigos).

Por ejemplo, usted podría definir un objeto DBObject Factura para tener un campo para NombreCliente, mostrando el Nombre del Cliente al que pertenece la Factura, pero seria un mal diseño relacional almacenar los Nombres de lo Clientes en la Tabla de Factura (esta tabla no estaría normalizada correctamente).

En lugar de esto usted puede definir un “campo virtual” para los objetos DBObject, y proveer la lógica para buscar el valor. Podemos adicionar una llamada al método `addVirtualField()` dentro del método `setupFields()` de la siguiente forma:

```
addVirtualField("NombreCliente", "varchar", 30, "Nombre
Cliente");
```

En el ejemplo anterior de Cliente se especifico que el objeto Cliente tiene un campo NombreCliente. Queremos ahora que nuestro campo virtual busque el nombre de cliente apropiado de manera automática, haciendo que parezca que el nombre esta almacenado con el valor de la factura.

Para hacer esto extendemos el método `getField(Cadena de Caracteres)`:

```
1. public String getField(String nombreCampo) throws
   DBException {
2.     if (nombreCampo.equals("NombreCliente")) {
3.         Cliente nuestroCliente = new Cliente();
4.         nuestroCliente.setDBName(getDBName());
5.         nuestroCliente.setField("ClienteID",
   getField("ClienteID"));
6.         nuestroCliente.retrieve();
7.         return nuestroCliente.getField("NombreCliente");
8.     }
9.     return super.getField(nombreCampo);
10. }
```

Como usted puede ver es muy importante retornar el valor del método de la super clase para el resto de los campos, como se muestra en la línea 9.

CAMPOS DE SOLO LECTURA

Algunos campos en un objeto `DBObject` pueden tener valores que solamente pueden ser definidos cuando el artículo es almacenado por vez primera, tales como fecha de creación o números de serie secuenciales. Para especificar esto, el método `setReadOnly()` puede ser usado. Por ejemplo:

```
setReadOnly("FechaDeCreacion");
```

Si usted necesita un número de serie secuencial, tales como un número de factura, usted puede usar el objeto `DBObject NEXTNUM` para que lo ayude con esto. Por ejemplo:

```
1. public void add() throws DBException {
2.     NextNumber miProximo = new NextNumber();
3.     miProximo.setDBName(getDBName());
```

```

4.     long miNumeroDeProceso =
miProximo.getNextTentative("PROCESO");
5.     setField("NumeroDeProceso", new
String().valueOf(miNumeroDeProceso));
6.     super.add();
7.     miProximo.getNextConfirm("PROCESO");
8. } /* add()

```

CAMPOS DE MÚLTIPLE VALORES

Algunos campos en un objeto DBObject pueden tener solamente un conjunto específico de valores permitidos. Un ejemplo simple puede ser campos del tipo Si o No. Los únicos valores válidos son Si o No (Yes or No). Para hacer esta restricción, podemos llamar el método `setMultiValued()` dentro del método `setupFields()`:

```

setMultiValued("CuentaAbierta");

```

Una vez que usted ha especificado que un campo es multi-valor, usted puede enumerar los posibles valores por uno de dos métodos.

En el primer caso usted puede extender el método `getValidValues()` para suministrar los valores disponibles para este campo. Esto es más apropiado para valores estáticos, tales como los de nuestro ejemplo de arriba:

```

1. public Vector getValidValues(String nombreDeCampo)
throws DBException
2.     if (nombreDeCampo.equals("Cuenta Activa")) {
3.         Vector misValores = nuevoVector();
4.         misValores.addElement(new ValidValue("S", "Si");
5.         misValores.addElement(new ValidValue("N", "No");
6.         return misValores;
7.     }
8. return super.getValidValues(nombreDeCampo);
9. }

```

Otro método para proveer valores válidos es especificando lo que se denomina “objeto de búsqueda” para el campo. Este es apropiado usarlo si el valor para el campo proviene de otro objeto DBObject, tales como una Tabla de códigos de búsqueda. Por ejemplo:

```
1. setMultiValued("TipoCliente");
2. setLookupObject("TipoCliente",
"com.sucompannia.dboj.TipoCliente");
```

Este código asume que el objeto TipoCliente existe e implementa el método `getValues()`, el cual retorna un vector con los valores válidos.

Existe una forma directa de implementar un método `getValues()` donde el valor a ser retornado son simplemente campos llave y una descripción. Este es el método `getValuesDefault(cadena de caracteres)`.

Los campos multi-valor son manejados por el programa DBMaint de una manera específica: Cuando los artículos son listados, la descripción del valor es mostrada en lugar del valor en si mismo (ejemplo: “Si” es mostrado en lugar de “S”) y durante la entrada del campo una lista desplegable (drop-down list) será presentada para que el usuario seleccione de esta.

CAMPOS SECRETOS

Un campo en un DBObject puede ser definido como “secreto” por lo que estos valores no pueden ser vistos por los usuarios con solamente usar la habilidad “Buscar” (Search), el campo es también mostrado con asteriscos durante la entrada de datos. Los campos de Palabra de Paso, por ejemplo, podrían usar esta característica.

VALIDACIÓN DE CAMPOS

Los objetos DBObjects pueden ser definidos para que solamente valores apropiados sean aceptados por los campos. Varias características facilitan esto:

- *Nulo / no Nulo (Null / not Null):*

El parámetro booleano para el método `addField()` es una forma

simple de validación, un valor no nulo debe ser especificado para todos los campos donde este valor sea falso (false).

- *Chequeo de Tipo de Dato (Data-type checking):*

El tipo de dato es también una forma básica de validación, solamente los valores del tipo apropiado serán aceptados, aún cuando el campo halla sido definido usando el método `setField(cadena de caracteres, cadena de caracteres)`.

- *Valores válidos (Valid Values):*

Los campos del tipo multi-valor son también validados, solamente un valor de los que se encuentran en la lista es permitido.

- *Extendiendo el método `setField()`:*

Una forma mas especifica de validar campos es extendiendo el método `setField()`, como se muestra a continuación:

```
1. public void setField(String nombreDelCampo,
String valorDelCampo) throws DBException
2. if (nombreDelCampo.equals("Prioridad")) {
3. if (!(valorDelCampo.equals("A") ||
valorDelCampo.equals("B") ||
valorDelCampo.equals("C"))) {
4.     throw new DBException ("La Prioridad debe
ser A, B o C");
5. }
6. super.setField(nombreDelCampo, valorDelCampo);
7. }
8. }
```

LOS OBJETOS AUTODBObject

Los objetos `AutoDBObject` es la forma mas fácil de obtener acceso a sus Tablas dentro de las Base de Datos para construir prototipos de su aplicación. Los objetos `AutoDBObject` pueden llenar sus campos de manera automática con la información del “Schema” de su Tabla destino. Esto permite a un `AutoDBObject` ser instanciado y usado para acceder a una Tabla sin crear ningún código para

esto. El Servlet DBMaint tiene un parámetro especial que permite que un objeto AutoDBObject sea usado.

```
DBMaint?dbobj=com.jcorporate.expresso.core.dbobj.AutoDBObj  
ct&table=SCHEMALIST&cmd=List
```

Este comando listará (y hace posible la edición) en la Tabla SCHEMALIST en la Base de Datos actual. No se necesita ninguna línea de código pero el usuario deberá tener acceso al objeto AutoDBObject (recuerde que el objeto AutoDBObject es un objeto del tipo SecuredDBObject).

ADVERTENCIA: Si usted le da acceso completo a un usuario al objeto AutoDBObject le permitirá leer/escribir desde/hacia cualquier Tabla en la Base de Datos actual (o al menos en cualquier Tabla que el usuario de la Base de Datos halla especificado en el fichero de propiedades como que tiene acceso a ella. Esto deberá ser usado con gran cuidado particularmente en ambientes de producción.

PROCESO DE “CACHING”

Para mejorar el rendimiento, los objetos DBOBJECT y valores válidos pueden ser “cacheado” o guardados en memoria. Como el acceso a la memoria es muchas veces mas rápido que el acceso a disco (o el acceso a Base de Datos) esto puede resultar en mejoras significativas sobre el rendimiento.

El “Caching” de objetos DBOBJECTS puede ser habilitado adicionando una entrada a la Tabla DBOBJPAGELIMIT, especificando un valor diferente de 0 para el campo “Cache Limit”. Este campo define el número de objetos DBOBJECT que serán “cacheados”, como máximo. El administrador de “cache” llenará la “cache” con los términos usados mas frecuentemente, así que el rendimiento será mejorado en la medida que pase el tiempo y los elementos de la “cache” sean llenados. Los mejores valores para los limites de la “cache” para cada objeto DBOBJECT dependen de la naturaleza exacta de su aplicación y la memoria disponible para su Maquina Virtual Java (Java Virtual Machine - JVM). La línea

de comandos de la JVM tiene opciones para ajustar la cantidad de memoria disponible para el “caching” por ejemplo la opción –Xmx.

Adicionalmente, los valores válidos en la “cache” son afectados por las operaciones de actualización, borrado y adición por lo que los valores de la “cache” nunca estarán desactualizados. Esta es otra razón importante para usar solamente accesos a su Base de Datos a través de los objetos DBObject en sus aplicaciones, si usted lo hace, los valores de la “cache” para los artículos siempre estarán actualizados. Por otro parte, si usted usa comandos SQL para hacer actualizaciones, entonces los valores en la “cache” podrían desactualizarse comparado con los últimos valores en sus Base de Datos.

Para mas detalles sobre “caching”, ver la documentación sobre “Caching”.

OBJETOS DE BASE DE DATOS EN MÚLTIPLES BASE DE DATOS

Expresso tiene la capacidad para definir y mantener una piscina de conexiones para múltiples Base de Datos, y para enlazar objetos de Base de Datos con una fuente de datos (data source) en particular.

Expresso deberá tener al menos una conexión de Base de Datos definida para poder operar. Esta Base de Datos primaria es conocida como Contexto de Base de Datos por defecto. El Contexto de Base de Datos por defecto contiene información sobre instalación y configuración que Expresso necesita para operar. La Base de Datos por defecto es siempre la que Expresso asumirá que esta siendo usada cuando un objeto DBObject es manipulado, a menos que se le diga a Expresso que busque en otros sitio en una de las 2 formas siguientes

1. Contexto Logged-In
2. Mapeando objetos de Base de Datos.

CONTEXTO LOGGED-IN

Virtualmente todos los programas estándares que vienen con Expresso utilizarán el Contexto Logged-In actual para sus objetos de Base de Datos. Cuandoquiera que un usuario Entra al Sitio, este establece, explícitamente o por defecto un Base de Datos/Contexto actual. Usualmente esto es hecho

seleccionando un contexto desde una lista desplegable cuando se Entra al Sitio a través del Controlador “Login”, pero esto también puede ser implícito por medio de la opción “db” para un Controlador URL en particular.

DBMaint, por ejemplo, siempre definirá cualquier objeto DBObject para que un usuario en particular use el Contexto Logged-In actual. También, si el usuario ha Entrado al Sitio en la Base de Datos/Contexto “demo”, todos los objetos que utiliza DBMaint harán una llamada a `setDBName("demo")` antes de ser usados.

Esto provee la habilidad para tener contexto enteramente separados con Bases de Datos distintas corriendo en el mismo servidor de Servlet/JSP/Aplicación, con solamente una única máquina virtual de java (Java Virtual Machine -JVM).

Los contextos para múltiples Base de Datos son definidos teniendo mas de un elemento “context” en el fichero de configuración `expresso-config.xml`, cada elemento “context” puede definir una sección JDBC para especificar información sobre la conexión a la Base de Datos para este contexto.

MAPEANDO OBJETOS DE BASE DE DATOS

Muchas aplicaciones empresariales requieren acceso a datos almacenados en múltiples Base de Datos. La funcionalidad de Mapeo de objetos de Base de Datos en Expresso permite a un desarrollador definir nuevos contextos para describir Base de Datos particulares, y entonces asociar los objetos DBObject con esos contexto de Base de Datos a fin de que cuando el objeto sea manipulado este siempre use el contexto de Base de Datos correcto.

Las aplicaciones Expresso pueden ahora correr con un “Contexto dividido”, con un contexto manejando las Tablas de control (tales como USERLOGIN, etc.), y una Base de Datos almacenando los datos de la aplicación (como un repositorio de datos compartidos por diferentes aplicaciones Expresso). Esto permite crear diferentes “interfaces“ y ”vistas” para los mismos datos, con Entradas la Sitio y grupos de seguridad independientes, etc. En el fondo usted puede ahora especificar muchos contextos de Base de Datos diferentes, y definir

objetos DBObject como que pertenecen uno de estos contextos a nivel del “schema”. Después de este paso, el sistema siempre buscará en la Base de Datos correcta cuando usted usa un objeto DBObject.

Veremos ahora como se hace esto:

- Crear un contexto de Base de Datos nuevo, otro aparte del contexto “default”. Por ejemplo, digamos que estamos creando un contexto de Base de Datos llamado “rh” que mapea a nuestra Base de Datos de recursos humanos.

Para hacer esto, nosotros adicionamos un elemento “context” al fichero de configuración expresso-config.xml para este nuevo contexto con la información de conexión a la Base de Datos en un elemento JDBC. ver la documentación sobre el fichero expresso-config.xml para detalles sobre el formato del mismo.

Usted puede desear que este nuevo contexto no contenga ninguna de las Tablas de Expresso. Usted puede indicar esto a través del uso de la bandera(flag) “notExpresso”, como se describe en la documentación y el DTD sobre el fichero expresso-config.xml. Esto especifica que el contexto marcado con la dicha bandera no puede ser usado para almacenar las Tablas propias de Expresso, tales como las Tablas de seguridad y grupos, tablas de definiciones y así sucesivamente. Esto usualmente es preferible si el contexto será usado estrictamente para datos de la aplicación, con la información de las definiciones de Expresso en un contexto diferente.

- Creamos unos cuantos objetos que mapeen las Tablas en la Base de Datos de recursos humanos “rh”. Por ejemplo creamos un objeto DBObject llamado “Empleados” que se mapee a la Tabla de empleados en la Base de Datos “rh”, y creamos un objeto DBObject llamado “Certificación” que se mapee a la Tabla “Certificación” en la Base de Datos “rh”.

- En nuestro “schema”, hemos definido el objeto DBObject como que pertenece a nuestro contexto “rh”. Hacemos esto adicionando el objeto DBObject al “schema” de la siguiente forma:

```
add(Empleado(), "rh");
add(Certificacion(), "rh");
```

- Ahora corremos la herramienta DBTool (o DBCreate). DBTool verá estas directivas y automáticamente crea una entrada DBOtherMap para decirle a Expresso que siempre use estos dos objetos DBObject contra el contexto de Base de Datos “rh”. Después de correr DBTool, las siguientes dos entradas aparecerán en la Tabla de Expresso “DBOTHERMAP” en el contexto por defecto

```
com.minombredepaquete.Empleado | rh | Tabla de
Empleados
com.minombredepaquete.Certificacion | rh | Tabla de
Certificacion
```

- Como usted puede ver, la tabla DBOTHERMAP es usada por Expresso para mapear los nombres de clases de los objetos DBObject a un contexto de Base de Datos en particular. Esta Tabla puede ser directamente manipulada para cambiar los mapeos de objetos a contextos de Base de Datos, pero corriendo la herramienta DBTool con las entradas del “schema” es generalmente mas seguro y fácil.
- Las entradas en la Tabla “DBOTHERMAP” son leídas en memoria cuando el Expresso comienza. Además si Expresso esta corriendo y usted hace cambio a las entradas DBOTHERMAP, usted necesitará restaurar Expresso para que los mapeos tengan efecto.
- Cuando Expresso se inicia, usted verá un mensaje que dice: “Leyendo los mapeos otherdb...”, “encontrados 2 mapeos otherdb” (“Reading otherdb mapings...”, “2 otherdb mappings found”).
- Estos objetos ahora siempre serán buscados y salvados para el contexto “rh”, en lugar del contexto por defecto o contexto “logged-in” actual.

NOTA IMPORTANTE: Los objetos `DBObject` mapeados a otros lugares trabajarán en todos los casos donde se le permita a los objetos crear sus propias conexiones. Sin embargo si usted especifica un objeto de conexión que apunte a alguna Base de Datos, y entonces crea el objeto `DBObject` con esta conexión de manera explícita, el objeto `DBObject` trabajará contra esta conexión de Base de Datos independientemente de lo que se especifique en la Tabla “`DBOTHERMAP`”. Esto frecuentemente es el caso con una transacción de múltiples partes contra una Base de Datos, donde se declara explícitamente una conexión a través del objeto `DBConnection` con el objetivo de poder usar sus métodos `commit()` y `rollback()`. Cuando usted crea un objeto `DBObject` con una conexión de manera explícita, usted tiene la responsabilidad de asegurarse que la conexión se hace contra la Base de Datos correcta. Una forma de hacer esto es inicializando normalmente el objeto (ejemplo sin la conexión de manera explícita) y usar el método `getDBName()` para determinar que conexión esta asignada a este, entonces usar este contexto para subsiguientes conexiones.

CAPÍTULO 5. USANDO LOS OBJETOS CONTROLADORES

VISTAZO GENERAL

Los objetos Controladores (Controller Object) proveen una forma de encapsular las interacciones con el usuario haciéndolos disponibles para cualquier tipo de interfaz de usuario (Ej. Servlet, JSP, Applet, Aplicaciones y otros). Un Controlador es una maquina de estados finitos(finite-state machine), donde el flujo de un estado a otro es controlado por el Controlador en si.

Los Controladores pueden asegurarse usando los mecanismo de seguridad de Expresso, a un grupo de usuarios puede o no permitírsele acceder completamente al Controlador, o se le pueden dar permisos para acceder solamente a algunos estados dentro del Controlador. Esto permite que un único Controlador ser usado por una amplia cantidad de usuarios, donde todos los estados podrían estar solamente disponibles para ciertos usuarios (Ej. administradores del sistema).

¿PORQUÉ USAR CONTROLADORES ?

Los Controladores poseen un número de ventajas:

- Encapsulan la lógica de negocio separándola de la lógica de la interfaz del usuario:

Los Controladores no tienen que ver con la interfaz del usuario que presenta sus Entradas(Inputs), Salidas(Outputs), y Transiciones(Transitions) al cliente, esto separa la lógica de negocio de la interfaz del usuario de una manera muy clara, y permite que cada una sea mantenida de manera independiente tanto como sea posible, promoviendo buenas prácticas de diseño.

- Acceso seguro dinámico:

Cada estado de un Controlador puede ser seguro, y la seguridad de los datos es fácilmente administrada a través de las capacidades de Expresso. Esto permite que la seguridad pueda ser actualizada desde cualquier lugar y los cambios tengan lugar inmediatamente. Esto hace

los Controladores ideales para situaciones donde los permisos a los usuarios podrían ser actualizado dinámicamente, por ejemplo, cuando un Cliente finaliza un Controlador permitiendo a el acceder a alguna información en línea, la finalización de un Controlador le permite a el tener acceso a estados adicionales en otros Controladores.

- **Arquitectura Modelo-Vista-Controlador (MVC):**
Los objetos Controladores proveen la parte del Controlador de la arquitectura MVC de forma que sea portable entre todos los tipo de ambientes Java. Ellos pueden escalar de todas las formas desde un sitio basado en páginas de servidor Java(JSP) hasta sitios complejos con grupos de servidores usando componentes empresariales java (EJB) y Servidores de aplicación.
- **Interfaz de usuario por defecto**
Existe una implementación de un manipulador de vistas (ViewHandler) que es usado cuando no se definen vistas personalizadas, lo cual puede ser usado para correr Controladores son la necesidad de programar o diseñar con JSP una Interfaz de Usuario personalizada, permitiendo que un Controlador básico sea desplegado muy rápidamente y luego mejorado con una Interfaz de Usuario personalizada.
- **Administración de sesiones**
Los Controladores en si no preservan ninguna información sobre los estados de una llamada a otra de estos, requiriendo de sus elementos de entrada y parámetros para proveer a estos de la información necesaria para procesar el próximo estado. Ellos pueden, sin embargo, usar un objeto del tipo PersistentSession para preservar la información del estado entre llamadas y hacer estos disponibles a otros objetos Controladores.
- **Pruebas**
Existen pruebas muy útiles para comprobar los objetos Controladores,

estas están disponible tanto desde la líneas de comandos como desde páginas JSP.

CONTROLADORES BÁSICOS

Los Controladores son básicamente una colección de estados, donde cada estado representa un paso en particular o una unidad de procesamiento. De esta forma los Controladores son en efecto una máquina de estados finitos, y la función de transición de un estado a otro es como ellos llevan a cabo su trabajo.

Un Controlador genera 3 tipos de objetos cuando este hace una transición de un estado a otro. Estos objetos son:

ENTRADAS (INPUTS)

Un objeto Input es una solicitud desde el Controlador para el cliente para suministrar algún tipo de información. Algunos atributos adicionales de los objetos Input pueden proveer algunas sugerencias sobre formateo, las cuales pueden o no ser usadas por el cliente, pero la presentación actual hacia el cliente se le deja la parte de la aplicación que tiene que ver con la Interfaz de Usuario. Los objetos Input pueden proveer también valores válidos, los cuales permiten a la Interfaz de Usuario presentar una lista de valores al usuario para que este seleccione un valor de esta. Nuevamente, la forma de esta lista se le deja a la parte de su aplicación que tiene que ver con la Interfaz de Usuario.

Normalmente, si un estado en particular de un objeto Controlador solicita una entrada, los estado subsiguientes requerirán este como parámetro.

SALIDAS (OUTPUTS)

Un objeto Output representa una respuesta desde el Controlador a ser presentada a los usuarios. Esta puede ser tan simple como una cadena de caracteres, o tan compleja como un árbol anidado de elementos, que la Interfaz de Usuario podría decidir presentarla como una tabla, por poner un ejemplo. Cada Output puede tener cero o más objetos del tipo Outputs anidados dentro de este, con el objetivo de representar estructura en los elementos retornados. Por

ejemplo, un objeto Output llamado “Factura” podría tener una serie de objetos Output anidados con cada línea a representar.

Los objetos Output también tienen atributos, los cuales son grupos arbitrarios de parejas de nombre/valor que ayudan además a describir el objeto Output a la aplicación Cliente, por ejemplo, el objeto Output “Factura” podría tener un atributo llamado “cantidad”, el cual podría dar el número de elementos asociados con esta Factura.

TRANSICIONES (TRANSITIONS)

Un objeto Transition representa una selección del cliente para transitar a un nuevo estado, solamente los estados que son apropiados dado el estado actual y que son permitidos por los permisos otorgados al usuario tienen acciones generadas para estos.

BLOQUES (BLOCKS)

Para un fácil uso en las páginas JSP (y en otros ambientes) donde hay muchas Entradas(Input), Salidas(Output) y acciones, los Controladores disponen del concepto de Bloques.

Un objeto Block es como cualquier otro objeto ControllerElement. Su propósito es actuar como contenedor de otros objetos del tipo ControllerElement tales como Transition, Input, Output y objetos Block. La idea detrás de los objetos Block es que este es una agrupación lógica de otros objetos ControllerElement para propósitos de presentación. Piense sobre una página HTML la cual tiene múltiples secciones (no necesariamente marcos), con cada sección teniendo sus propios mensajes de texto, botones, enlaces y formas. Si usted está escribiendo una página JSP, previamente usted podría tener una enumeración para cada objeto de tipo Input, Output y Transition. Esto permite al que escribe la página JSP conocer, por nombre, cual objeto Input, Output y Transition están agrupados juntos. Con los objetos Block, los Controladores pueden agrupar cada conjunto lógico de objetos Input, Output y objetos Transition dentro del objeto Block, con cada sección lógica de una página HTML teniendo sus propios objetos Block. El

que escribe la página JSP a su vez podría a su vez obtener una enumeración de todos los objetos Blocks desde el Controlador, iterar sobre cada objeto ControllerElement y crear entonces el código HTML apropiado. Dado que un objeto Block retorna cada objeto del tipo ControllerElement en el orden en que este fue puesto en un objeto Block, si el que escribe la página JSP no tiene cuidado, el objeto ControllerElement podría ser puesto en el mismo orden en que el Controlador lo puso. Por supuesto, el que escribe la página JSP es libre de acceder a cada objeto ControllerElement por su nombre y hacer entonces una presentación HTML personalizada.

Aquí le mostramos un ejemplo. Permítanos asumir que la página HTML consiste de varias líneas de mensajes, seguidas por algunos enlaces y por una forma. En algunos Controller.someState():

```
.....
//El primer bloque lógico
Block b1 = new Block("Header");      // El parámetro define
el nombre del Bloque
Output o1 = New Output("Msg1", "Este es el mensaje 1");
b1.add(o1); // Anida la acción dentro del bloque
Output o2 = New Output("Msg2", "Este es el mensaje 2");
b1.add(o2);
Output o3 = New Output("Msg3", "Este es el mensaje 3");
b1.add(o3);
addBlock(b1); //Adiciona el bloque al Controlador.

// El segundo bloque lógico
Block b2 = new Block("Links");
Transition a1 = new Transition("Tomar acción 1",
"/servlets/ControllerServlet/?trx=.....");
a1.setName("Transition");
b2.add(a1);
Transition a2 = new Transition("Tomar acción 2",
"/servlets/ControllerServlet/?trx=.....");
a2.setName("Transition");
b2.add(a2);
```

```

Output o4 = new Output("Msg4", "Por favor seleccione uno de
los enlaces de arriba");
b2.add(o4);
addBlock(b2);

// El tercer bloque lógico
Block b3 = new Block("Form:Form1");
Output o5 = new Output("Msg5", "Por favor llene la
información de abajo");
b3.add(o5);
Input i1 = new Input("Name");
i1.setLabel("Nombre");
i1.setType("text");
b3.add(i1);
Transition a3 = new
Transition("Submit", "/servlets/ControllerServlet?controller
=.....");
a3.setName("Submit");
b3.add(a3);
addBlock(b3);
.....

```

En el correspondiente generador HTML (tanto si es otra clase utilitaria, un Javabean para ser accedido desde la página JSP, una clase adaptadora para WebMacro/Freemarker, o una página JSP en si misma), uno podría escribir algo similar a lo siguiente:

```

.....
String blockName = "....."; // Nombre como en el
Controlador
for (Enumeration e =
controller.getBlock(blockName).getContents().elements();
e.hasMoreElements(); ) {
    ControllerElement oneElement =
(ControllerElement)e.nextElement();

```

```

    if (oneElement instanceof Transition) {
        handleTransition((Transition)oneElement);
    } else if (oneElement instanceof Input) {
        handleInput((Input)oneElement);
    } else if (oneElement instanceof Output) {
        handleOutput((Output)oneElement);
    } else if (oneElement instanceof Block) {
        handleBlock((Block)oneElement);
    } else {
        throw new Exception("No se puede manejar objetos de
tipo:" + oneElement.getClass().getName());
    }
}
}
.....

```

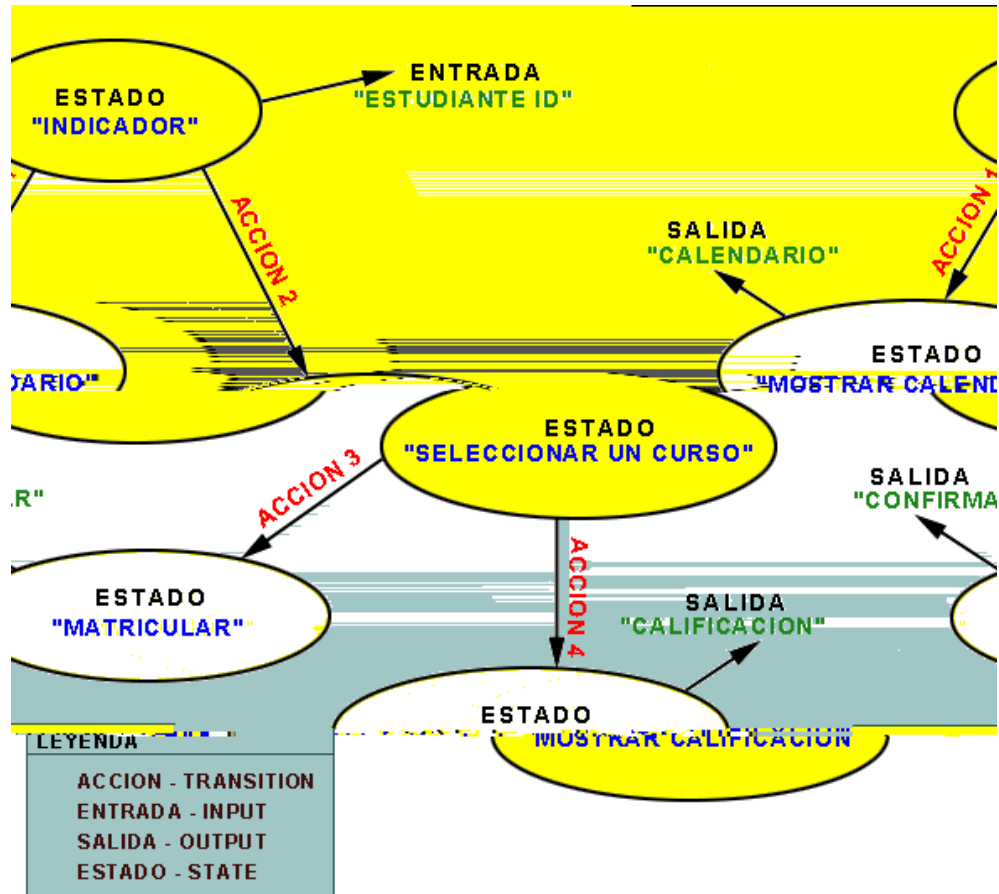
Usted puede usar también una llamada al método `controller.getBlocks()` para obtener una enumeración de todos los objetos `Block` para iterar sobre estos. *Note que los mecanismos exactos para la generación del código HTML no son mostrados aquí para mantener el ejemplo simple y sin rodeos.*

TIPOS DE CONTROLADORES

Existen varios tipos de objetos Controladores disponibles, todos extendiendo la clase base `com.jcorporate.common.controller.Controller`. Cada uno de ellos tiene una uso específico:

- clase `Controller` (Controlador)
Esta es la clase base para todos los objetos `Controller`, y puede ser directamente extendida para objetos `Controller` que manipulen sus propias conexiones o que no se conecten a ninguna Base de Datos.
- clase `DBController` (Controlador de Base de Datos)
La clase `DBController` extiende el objeto `Controller` que es la clase base y provee un fácil acceso a conexiones a Base de Datos desde la piscina de conexiones.

EXPLICACIÓN DE LA ACTIVIDAD DEL OBJETO CONTROLLER



En el diagrama anterior usted puede ver una representación simple de un objeto Controller con 5 estados(states). A continuación seguiremos la ejecución de este Controlador para ver como trabajan las interacciones:

- El objeto Controller o Controlador comienza (asumamos que este es llamado desde una página JSP para los propósitos de nuestra discusión) en el estado "INDICADOR" que se encuentra en la parte superior del diagrama. Entrando este estado causa que el Controlador genere un elemento Entrada(Input), llamado "ESTUDIANTE ID", y dos elementos de transición (Transition). Los elementos de acción indican que el usuario puede seleccionar "SELECCIONAR UN CURSO"(Transición 2) o "MOSTRAR CALENDARIO" (Transición 1) desde este punto (estamos asumiendo que cualquier usuario que halla

Entrado al Sitio tiene acceso a todos estos estados). El control es entonces retornado a la Interfaz de Usuario.

- Digamos que el usuario entró un “ESTUDIANTE ID” apropiado y seleccionó la Transición 2, “SELECCIONAR UN CURSO”. El Controlador transita desde el estado “INDICADOR” hacia el estado “SELECCIONAR UN CURSO”. El estado “SELECCIONAR UN CURSO” ha requerido de un parámetro, el “ESTUDIANTE ID” que fue pedido en el estado anterior. Si este parámetro no está presente, la transición hacia el nuevo estado no será exitosa (y el usuario deberá ser notificado de cualquier manera que el diseñador de la Interfaz de Usuario seleccione). En este caso, el “ESTUDIANTE ID” estaba presente, así que el Controlador entró en el estado “SELECCIONAR UN CURSO”.
- Este estado produce otro elemento de Entrada(Input), esta vez pidiéndole al usuario que seleccione de una lista de cursos disponibles. El elemento de Entrada(Input) podría ser suministrada con una lista de valores válidos (por ejemplo los cursos disponibles para este estudiante), y podría ser representado por la interfaz de usuario como una lista desplegable. El estado “SELECCIONAR UN CURSO” también genera dos elementos de acción, la Transición 3 y 4. Si el usuario no tiene permiso para acceder al estado “MATRICULAR”, por ejemplo, entonces la Transición 3 podría no ser generada.
- Digamos que el usuario selecciona un curso de la lista y selecciona la Transición 4 para transitar hacia el estado “MOSTRAR CALIFICACION”. En este caso, el estado “MOSTRAR CALIFICACION” podría requerir dos parámetros: el “ESTUDIANTE ID” del primer estado y el “CURSO” seleccionado en el segundo estado. Como ambos están presentes, la transición ocurre, y el estado “MOSTRAR CALIFICACION” genera un elemento de Salida (Output) (probablemente un grupo de elementos de salida, usando el anidamiento) que presenta la información solicitada al usuario.

Este ejemplo es por supuesto muy simple, y un Controlador real podría tener muchas otras conexiones entre estados, y probablemente procederes mas complejos dentro de cada estado, pero el ejemplo tiene como objetivo mostrarle el modelo que usan los objetos Controladores. Es importante tener en cuenta que este Controlador podría ser usados de diferentes formas, por ejemplo, desde una página JSP que muestre una lista de estudiantes que están pasando un determinado curso, un usuario autorizado podría seleccionar un enlace que diga “VER CALIFICACIONES”. Este enlace podría invocar el Controlador de nuestro ejemplo, pero no en el estado inicial de este: Este iría directamente al estado “MOSTRAR CALIFICACION”, asumiendo por supuesto que los permisos están correctos y que el curso y el “ESTUDIANTE ID” han sido suministrados (como parámetros requeridos para entrar en este estado), nuestro Controlador del ejemplo podría responder con la misma Salida(Output) que este hizo cuando fue invocado en la forma de nuestro ejemplo. Este tipo de interacción con otros Controladores incrementa la utilidad del modelo Controlador de manera considerable, permitiendo rehusarlos sin sacrificar la portabilidad o la integridad de los datos.

Por supuesto, un objeto Controlador típicamente interactúa con uno o mas objetos de Base de Datos (u otros Controladores) para hacer su trabajo, pero estas interacciones son completamente ocultadas dentro de los estados mismos, y el cliente no necesita tener conocimiento de estos.

ESTADOS (STATES)

Esto se refiere a los estados dentro de un Controlador que realmente contienen la lógica para que el Controlador lleve a cabo sus funciones. Estos estado pueden escribirse simplemente como métodos dentro del objeto Controlador, o pueden ser objetos independientes, heredando de la superclase “State”. Para estados simples, es frecuentemente mas fácil escribir los estados como métodos, entonces escribiendo el método `newState()` dentro del Controlador para llamar simplemente al método apropiado para cada nuevo

estado. Para Controladores mas complejos, objetos “State” separados son frecuentemente superiores ya que ofrecen una clara separación entre cada estado.

Cada estado debe ser completamente independiente de cualquier otro, Ej. ellos no deben contar con elementos compartidos que sean variables locales al Controlador. En lugar de eso, cada estado solamente trabajan en sus parámetros y produce sus propias Salidas(Outputs), Entradas(Inputs) y Transiciones(Transitions).

ESTADO INTERNOS

Si los estados son escritos internamente a la clase “Controller”, entonces los objetos “States” son creados internamente en el constructor de el método, como se muestra a continuación:

```
1. /**
2.  * Nuestro constructor declara los estados que este
   Controlador soporta
3.  */
4. public DBSecurityMatrix() {
5.     State indicador = new State("indicador", "Solicita un
   Schema y un grupo de usuarios");
6.     addState(indicador);
7.     setInitialState("indicador ");
8.
9.     State dbobjmatrix = new
   State("dbobjmatrix", "Entrar/Ver permisos de los objetos de
   Base de Datos");
11.    dbobjmatrix.addParameter("SchemaClass");
12.    dbobjmatrix.addParameter("GroupName");
13.    addState(dbobjmatrix);
14. } /* DBSecurityMatrix() */
```

En el código mostrado arriba, el objeto “State” es creado, y entonces se usa el método `addState()` para registrar que este estado pertenece al

Controlador. La lógica para el estado, sin embargo, es escrita dentro del método que se encuentra dentro del propio Controlador.

Cuando escribiendo un Controlador con estados muy simples, el método `newState()` es extendido como se muestra a continuación:

```
1. public void newState(String newState) throws
   ControllerException {
2.     super(newState(newState));
3.
4.     if (newState.equals("indicador")) {
5.         /* solicita al usuario seleccionar un grupo de
   usuarios */
6.         estadoIndicador();
7.     } else if (newState.equals("dbobjmatrix")) {
8.         dbObjMatrixState();
9.     }
10.
11.    if (!newState.equals("indicador")) {
12.        Transition deNuevo = new Transition("Comenzar
   de Nuevo", getClass().getName());
13.        again.setName("deNuevo");
14.        again.addParam("state", "indicador");
15.        addTransition(deNuevo);
16.    }
17. } /* newState(String) */
```

En el ejemplo de arriba el cual es muy simple, solamente se muestran dos estados: “indicador” y “dbobjmatrix”. El método `newState()` primero llama a la superclase “newState” (esto es de mucha importancia), y entonces chequea para ver cual de los dos posibles estados fue solicitado. Para cada estado, el control es pasado al método que implementa la lógica del estado escrita en el objeto “Controller” como un método privado.

Comenzando por la línea 11 hemos demostrado una técnica muy útil para los Controladores: la acción “indicador” es suministrada para comenzar de nuevo en todos los estados, excepto por supuesto en el estado “indicador” en si. Además

hemos chequeado que el estado solicitado no fue “indicador” y adicionamos el elemento “Transition”. En este ejemplo solamente hay 3 estados por lo que en este hecho es redundante, pero si hubiera un número mayor de estados, digamos 10, este pudiera librar de tener que adicionar el elemento “Transition” en cada uno de los 10 estados de forma separada.

ESTADOS EXTERNOS

En un Controlador mas complejo, o uno con mas estados disponibles, es frecuentemente mejor crear objetos “States” de manera individual, en lugar de tener estados internos como se describió arriba. En este caso, el método `newState()` no es de ninguna manera implementado dentro del Controlador. En lugar de eso, la superclase manipula la transición al estado correcto de forma dinámica invocando el método `run()` del objeto “State” cuando se entra en cada estado.

Cuando usamos objetos “State” externos, la declaración del Controlador es un poco diferente:

```
public Upload() {
    PromptBrowser browser = new PromptBrowser("browser",
    "Prompt for Upload from Browser");
    browser.addParameter("resource", false);
    addState(browser);

    DoBrowser dobrowser = new DoBrowser("dobrowser",
    "Process Upload from Browser");
    dobrowser.addParameter("action");
    addState(dobrowser);
} /* Upload() */
```

En el listado de arriba, “PromptBrowser” y “DoBrowser” son objetos definidos externos a la clase “Controller” que extiende los objetos “State”. Por ejemplo, “PromptBrowser” podría quedar así:

```

(package and import statements omitted)
1. public class PromptBrowser extends State {
2.
3.     public PromptBrowser(String stateName, String
descrip) {
4.         super(stateName, descrip);
5.     }
6.
7.     public void run() throws ControllerException {
8.         Controller myController = getController();
9.         String currentNumber =
StringUtil.notNull(myController.getParameter("number"));
... aqui el objeto State hará cualquier logica que necesite
10.    } /* run() */
11. } /* PromptBrowser */

```

En el listado de arriba, la clase “PromptBrowser” extiende el clase “State”, definiendo este como uno de los posibles estados para el Controlador e implementa el método `run()`. El método `run()` es llamado por la clase del objeto “Controller” cuando el Controlador transite al estado especificado, aquí es donde todo el trabajo del estado será hecho, donde las Entradas(Input), Salidas(Output) y Transiciones(Transitions) son generadas, etc.

Para que un objeto “State” tenga acceso a los parámetros del Controlador, las sesiones y otras informaciones, este tiene disponible el método `getController()`. Este método retorna un objeto “Controller”, específicamente el objeto “Controller” que contiene este estado. Esto permite al estado(“State”) usar el método `getParameter(nombre)` y otros métodos para interactuar con el Controlador en si, y usar el método `add()` para retornar sus Entradas(Input), Salidas(Output) y Transiciones(Transitions) como se requieran. Al final del método `run()` el control retorna a la superclase “Controller”, la cual entonces procesa las nuevas Entradas(Input), Salidas(Output) y Transiciones(Transitions) de manera apropiada.

Como parte de su procesamiento, un objeto “State” frecuentemente usa uno o mas objetos “DBObjects”, y por supuesto este también Procesos(jobs),

acceso a otros Controladores y generalmente realiza cualquier procesamiento que se requiera.

CAPÍTULO 6. LOS COMPONENTES PROCESO(JOB)

Expresso contiene facilidades para manejar tareas que toman tiempo en llevarse a cabo y que el usuario no desea esperar. Esta facilidad provee al cliente una forma de almacenar una entrada en una cola de Procesos, y un proceso en el servidor seleccionará las entradas de la cola para llevar a cabo el procesamiento necesario.

ESCRIBIENDO OBJETOS “JOB”

Un objeto “Job” extiende la clase “com.jcorporate.core.job.Job” y puede en su constructor hacer una llamada al método `addParameter()` declarar que parámetros requiere el Proceso para realizar su tarea. El manipulador de Procesos puede entonces chequear la existencia de estos parámetros en la cola de manera automática y manipular los errores si estos no son especificados. Por ejemplo, el objeto Proceso `com.jcorporate.expresso.ext.job.SendNotice` declara:

```
addParameter("Subject", "Message Subject");
addParameter("FileNumber", "File Number");
addParameter("NoticeText", "Notice Text");
```

El objeto “SendNotice” especifica que se requieren los parámetros Subject, FileNumber y NoticeText cuando este se pone en la cola. Usando el método `addParameter()` también significa que un Controlador genérico puede ser usado para encolar estos Procesos sin tener que escribir un Clase especial para este trabajo.

El objeto “Job” también debe implementar el método `getTitle()` para que las funciones estándares de seguridad puedan listar su nombre correctamente. Esto es tan simple como:

```
public String getTitle() {
    return new String("Send Notices");
}
```

El último método que un objeto “Job” deberá implementar es el método `run()`, igual que un hilo de ejecución(`thread`). El método `run()` es llamado por el manipulador de Procesos una vez que el Proceso es instanciado y definido, y llevar a cabo el procesamiento del Proceso. Un proceso puede interactuar con una Base de Datos, enviar correos electrónicos, y hacer cualquier procesamiento que requiera. El método `run()` usualmente usa el método `getJobQueueEntry()` de la superclase para obtener acceso a sus parámetros, a fin de conocer que procesar, y típicamente envía alguna notificación por correo electrónico cuando se completa. El Proceso “SendNotice” es nuevamente un buen ejemplo de un Proceso simple que puede ser copiado y modificado.

LOS OBJETOS “JOB”. SU SEGURIDAD

Los objetos “Job” tienen datos de seguridad para estos en la Base de Datos a través del objeto “JobSecurity”. Estos datos pueden usarse para especificar si le esta permitido a un usuario solicitar un Proceso(Job) específico, o para especificar que “funciones” de un Proceso un usuario tiene permiso para usar. El significado exacto de “funciones” es específico a un Proceso, por ejemplo, si un Proceso calcula la hipoteca (función “hipoteca”) o un préstamo personal (función préstamo), un determinado usuario puede dársele permisos para usar solamente la función “hipoteca” esto le permite al Proceso tomar las acciones apropiadas si se solicita la función incorrecta. De esta forma, la seguridad de los Procesos es menos sofisticada que la seguridad que poseen los Controladores, aunque por supuesto no existen razones para que un Proceso no pueda llamar a un objeto de tipo “Controller” para llevar a cabo su procesamiento.

ENCOLANDO LOS PROCESOS (OBJETOS “JOB”)

Un proceso es encolado con la creación de un objeto de Base de Datos “JobQueue” y sus parámetros a través del objeto “JobQueueParam”, el cual provee parámetros al Proceso de la misma forma que un método se los da a un método.

Un objeto del lado del servidor que herede de la clase “Job”, es invocado cuando el proceso “JobHandler” comienza a manipular un Proceso. Esta clase puede leer los parámetros dados y procesar su Controlador como se requiera, usualmente reportando al usuario que la tarea iniciada será confirmada por correo electrónico cuando esta se complete.

Aquí pueden haber por supuesto múltiples objetos “JobHandler” al mismo tiempo procesando todo los procesos encolados, y ellos se sincronizarán de forma tal que solamente un proceso del servidor solo procese un Proceso determinado una sola vez. Esto permite una Tolerancia a fallos y Escalabilidad, y el rendimiento puede mejorarse adicionando simplemente nuevos objetos del tipo “JobHandler” para manipular las solicitudes mas rápido.

CAPÍTULO 7. DESARROLLANDO PÁGINAS JSP

El desarrollo de páginas JSP usando Expresso se apoya enormemente en la arquitectura Modelo-Vista-Controlador, donde los componentes son como sigue:

CAPÍTULO 8. DESARROLLANDO SERVLETS

Esta sección esta dirigida a los desarrolladores que desean construir sus propios Servlets usando los componentes estándares de Expresso Framework. Es muy fácil construir Servlets que hereden funcionalidad avanzada sin escribir mucho código tal y como usted verá.

Existen fundamentalmente dos tipos de Servlets que pueden ser contruidos extendiendo los bloques pre-construidos que posee Expresso:

- Servlets de Base de Datos
Estos se conectan a través del objeto DBConnectionPool a una Base de Datos relacional en el servidor, y puede acceder la los objetos de Base de Datos (DBObjects). Estos Servlet heredan de la clase DBServlet. Un ejemplo de tales Servlet es el RunSQL, el cual permite pasar preguntas SQL a la Base de Datos.
- Servlet que no se conectan a Base de Datos
Esto Servlets son aquellos que no se conectan a una Base de Datos durante su ciclo de vida, tales como los Servlet “Search” que se provee con Expresso. Estos aún pueden beneficiarse de un número incontable de valiosos servicios heredando del objeto StdServlet.

SERVLET QUE NO SE CONECTAN A BASE DE DATOS

Estos Servlet pueden beneficiarse de los métodos que posee el objeto “StdServlet” heredando de este.

El objeto “StdServlet” provee la siguiente funcionalidad que puede ser usada por sus predecesores:

- Extracción de parámetros
El objeto StdServlet automáticamente lee sus argumentos desde la invocación del Servlet en la página HTML en un vector el cual puede ser dispuesto por sus predecesores.
- Auto referencia
El método `getServletPrefix()` puede ser usado para construir

enlaces para llamar al mismo Servlet con parámetros diferentes o otro Servlet en el mismo sistema.

- Manipulación de errores

Quizás la funcionalidad mas significativa que proveen los objetos `StdServlet` es la facilidad para manipular las condiciones de error y excepciones a través de la función “`showError`”. Es posible simplemente encerrar completamente los métodos `doGet()` y/o `doPost()` en bloques `try/catch`, para ver un ejemplo vea el Servlet “`Search`”

SERVLETS DE BASE DE DATOS

Los Servlets de Base de Datos heredan del objeto `DBServlet` y están provistos con un número de servicios adicionales:

- Conexiones a Base de Datos

Los Servlets de Base de Datos solicitan una conexión a una Base de Datos a través de la piscina de conexiones. El Servlet de Base de Datos es responsable de liberar la conexión tan pronto como sea posible.

- Seguridad de la aplicación

El método `checkAppSecurity()` permite a un Servlet determinar si el usuario actual que Entro al Sitio tiene permiso para correr este Servlet.

- Verificación de Entrada al Sitio

A través del método `requireLogin()`, los predecesores del Servlet `DBServlet` pueden asegurar que el usuario que Entro al Sitio es un usuario válido para acceder a sus funciones.

- Valores de Configuración e Instalación

Por medio de la conexión a la piscina de conexiones, los predecesores de `DBServlet` tienen acceso a los valores de Configuración e Instalación almacenados en la Base de Datos. Estos valores pueden ser usados para configurar la aplicación para preferencias y caminos específicos del Sitio, por poner un ejemplo.

SERVLET DE MANTENIMIENTO DE LA BASE DE DATOS

Una instancia especial de un Servlet de Base de Datos que es también fácil de construir usando Expresso es un Servlet de mantenimiento de la Base de Datos. Este está definido como un Servlet el cual permite operaciones básicas de mantenimiento de la Base de Datos en algunos objetos de Base de Datos (frecuentemente una Tabla) en una Base de Datos relacional. Por ejemplo, una forma HTML de mantenimiento de Clientes podría ser un ejemplo de un Servlet de mantenimiento de la Base de Datos.

Existen muchos ejemplos de Servlet de mantenimiento de Base de Datos, este es con mucho el tipo de Servlet más común en el Marco de Trabajo Expresso. La mayor parte de los Servlets simplemente usan el Servlet por defecto, DBMaint para dar sus servicios. DBMaint usa la información almacenada en un objeto DBObject para proveer las funcionalidades de adicionar, listar, buscar, actualizar y borrar de una manera estándar, sin tener que escribir cada Servlet de manera individual. Con el uso del objeto que hereda de DBObject llamado SecuredDBObject, estas funciones también pueden ser seguras, permitiendo a cada usuario solamente los permisos que tiene otorgado para cada Tabla en la Base de Datos.

El Servlet DBMaint está diseñado para permitir que se pueda extender de manera muy fácil. Esto se logró haciendo cada comando a los que responde DBMaint un objeto extensible en sí mismo. Estos objetos están almacenados en el paquete `com.jcorporate.common.commands`, y puede ser extendido para obtener funcionalidades personalizadas.

USANDO DBMAINT

Para usar DBMaint y que este provea inmediatamente capacidades de mantenimiento de la Base de Datos a través de un Servlet, usted primero debe definir un objeto de Base de Datos seguro (SecuredDBObject). Esto en la práctica es muy simple:

- Decida en qué paquete su nuevo objeto va a residir. Este no debe estar dentro de los directorios de Expresso. Nosotros recomendamos un

nombre de paquete de la forma “com.jcorporate.ext.aplicacion.dbobj”, donde “aplicacion” es el nombre de la aplicación que usted esta creando. Por ejemplo, si es una aplicación de Recursos Humanos, usted podría seleccionar “rh”, por lo que el nombre de su paquete podría ser “com.jcorporate.ext.rh.dbobj”. El nombre del objeto deberá reflejar el propósito deseado: digamos que usted este creando un objeto para mantener una Base de Datos de Empleados, usted podría nombrar esta “Empleado”.

- Copie un definición de un objeto SecuredDBObject de Expresso para usar esta como plantilla y ahorrarse escribir un poco. Le recomendamos que use el objeto “com.jcorporate.common.dbobj.User” como un buen ejemplo.
- Modifique el objeto copiado, reemplazando todas las instancias del nombre “User” con el nombre “Empleado”, modificando la definición del paquete, y por supuesto la Tabla y la definición de los campos para que sean los de la Tabla que usted desea tener en su Base de Datos. vea la documentación JavaDoc para más detalles sobre el objeto SecuredDBObject.
- En este punto usted puede decidir crear manualmente la Tabla para almacenar los objetos “Empleado”, existe una forma de que Expresso lo haga por usted, lo cual se discute en la sección -Los objetos “Schema”-.
- Compile su nuevo objeto para que el fichero .class este disponible.

Ahora usted va a crear un enlace a su nuevo objeto de Base de Datos usando el Servlet DBMaint:

- Seleccione la página Web desde la que usted va a llamar la función de mantenimiento: Permítanos asumir que usted la llamará “rh.html”.
- En esta página, cree un enlace para las funciones de Listar, Adicionar y Buscar para su nuevo objeto, usted puede querer usar las imágenes y formatos que son usadas en Expresso, estas pueden copiarse de una

página como “server.html” en el directorio /components/expreso del HTML bajado de nuestro sitio.

- Cada enlace podrá tener la siguiente forma:

```
"/servlet/DBMaint?next=rh.html&dbobj=com.jcorporate.ext.rh.dbobj.Employee&cmd="Add"
```

- El último parámetro (cmd) puede tener uno de los tres valores siguientes:

1. Add (Adicionar): Muestra una forma a pantalla completa para aceptar nuevas entradas para este objeto de Base de Datos.
2. List (Listar): Lista todos los artículos existente para este objeto de Base de Datos.
3. Search (Buscar): Muestra una forma para aceptar criterios de búsqueda para localizar uno o más artículos para este objeto.

Por supuesto, la opción “List” debe ser usada con cuidado si el número de artículos que se obtendrá es muy probable que sea largo (existe una facilidad para limitar el número de artículos ha mostrar y proveer paginación en su lugar). También existe un argumento “Update”, vea la documentación JavaDoc para más detalles sobre el objeto DBMaint.

- Crear un entrada de seguridad para su nuevo objeto de Base de Datos: ver el fichero de ayuda “Database Object Security” para más información en como hacer esto.

¡Eso es! Usted tiene ahora un Servlet para el mantenimiento seguro de su Base de Datos completamente funcional.

DBMAINT: UN EJEMPLO PASO POR PASO

A continuación presentamos un ejemplo de construcción de una simple aplicación para permitir el mantenimiento de un cliente, completamente seguro. Este ejemplo sigue el proceso que describimos arriba, pero con el código actual incluido como referencia, y con más detalles.

Asumiremos que nuestra forma de mantenimiento del cliente es el comienzo para una aplicación más sofisticada para el rastreo de ventas.

1. *Definir el objeto de Base de Datos*

Permítanos asumir para este ejemplo que usted ha instalado Expresso en el directorio `usr/java/lib/com/jcorporate`. Su variable `CLASSPATH` deberá entonces incluir el directorio `usr/java/lib`. Por supuesto la variable `CLASSPATH` para su motor de Servlet también deberá incluir este directorio, vea la sección de instalación para más detalles.

1. Primero crearemos el directorio `/usr/java/lib/com/jcorporate/ext`, donde iremos para definir extensiones al Marco de Trabajo Expresso. El nombre de paquete puede estar fuera del directorio de `Jcorporate` si usted lo desea.
2. Dentro del directorio “ext” crearemos el directorio para nuestro paquete, llamado “ventas” (Ej. `/usr/java/lib/com/jcorporate/ext/ventas`). Dentro de este crearemos otro directorio, llamado “dbobj” para guardar nuestros objetos de Base de Datos.

Esta jerarquía de directorios es bastante típica en las aplicaciones Java y ayuda a mantener los paquetes bien organizados, nuevamente nosotros usaremos estos nombres en particular en este ejemplo, si usted desea una organización diferente, eso está OK.

3. Dentro del directorio `ext/dbobj`, crearemos nuestros objetos de Base de Datos, llamado “Cliente.java”. El siguiente código define nuestro objeto:

Los comentarios dentro del código fuente describen el propósito de

cada sección.

```
1.  /*
2.   * Cliente.java
3.   */

4.  package com.jcorporate.ext.ventas.dbobj;

5.  import com.jcorporate.expresso.core.dbobj.*;
6.  import com.jcorporate.expresso.core.db.DBConnection;
7.  import com.jcorporate.expresso.core.db.DBException;

8.  /**
9.   * Un objeto Cliente almacena información sobre los
   clientes
10.  * para nuestra Aplicación Ventas de ejemplo.
11.  */
12. public class Cliente extends SecuredDBObject {
13.     private String thisClass = new
String(this.getClass().getName() + ".");
14.     /**
15.      * Existen 3 constructores posibles para los
SecuredDBObjects que
16.      * generalmente no es necesario extender estos
para los objetos que nosotros implementamos
17.      */
18.     public User() throws DBException {
19.         super();
20.     } /* User() */
21.     /**
22.      * Este constructor es usado cuando se suministra
23.      * un objeto conexión por el objeto que los llama
24.      */
25.     public User(DBConnection theConnection) throws
DBException {
26.         super(theConnection);
27.     } /* User(DBConnection) */
28.     /**
```

```

29.      * Este constructor es llamado por el Servlet
DBMaint
30.      * y suministra el nombre del usuario que esta
tratando
31.      * de conectarse al objeto de Base de Datos.
32.      */
33.      public User(DBConnection theConnection, String
theUser) throws DBException {
34.          super(theConnection, theUser);
35.      } /* User(DBConnection, String) */
36.      /**
37.      * El método setupFields hace el trabajo de
establecer la
38.      * definición del DBObject.
39.      */
40.      public void setupFields() throws DBException {
41.          /* Establece la Tabla primaria para este
DBObject */
42.          /* Note que puede haber mas de una, pero esta
es la por defecto */
43.          /* Tabla */
44.          setTargetTable("CLIENTE");
45.          /* Define una descripción para este objeto.
Esta debe ser un texto legible */
46.          /* que aparece arriba de la forma cuando se
usa con DBMaint */
47.          setDescription("Usuarios");
48.          /* Define cada uno de los campos en la Tabla
*/
49.          /* Note que los tipo de campos usados son
"internos", y pueden ser */
50.          /* mapeados a otros tipos para la Base de
Datos relacional usada */
51.          addField("ClienteNumero","int", 0, false,
"Número ID del Cliente");
52.          addField("ClienteNombre","varchar", 80,
false, "Nombre del Cliente");

```

```

53.         addField("Telefono", "varchar", 15, true,
"Número telefónico");
54.         addField("EMail", "varchar", 80, false,
"Correo electrónico del Cliente");

55.         /* ... usted podría por supuesto usar mas
campos para un Objeto */
56.         /* de producción */
57.         /* addKey es llamado para cada campo que
forme la llave primaria */
58.         addKey("ClienteNumero");
59.     } /* setupFields() */
60.     /**
61.     * Un método utilitario usado cuando se generan
lista de objetos
62.     */
63.     public DBObject getThisDBObj() throws
DBException {
64.         return (DBObject) new
Customer(this.getConnection());
65.     } /* getThisDBObj() */
66. } /* Cliente */

```

4. Ahora compile el objeto Cliente.java y corrija los errores.

2. Crear un objeto “Schema” para su nueva aplicación

Los objetos “Schema” identifican la aplicación completamente y sus objetos DBObjects requeridos para el Marco de Trabajo Expresso. Este le permite usar las facilidades de Expresso para crear Tablas y valores de configuración, ahorrándole mucho trabajo cuando usted esta definiendo su aplicación.

- En el mismo directorio “dbobj”, cree un fichero que se llame “Ventas.java” como se muestra abajo:

```

1.  /**
2.  * Ventas.java
3.  */
4.  package com.javacorporate.ventas;

```

```

5.  /**
6.   * objeto Schema para la aplicación demostrativa Ventas
7.   */
8.  public class Sales extends Schema {
9.   /**
10.   * Constructor por defecto
11.   */
12.   public Sales() throws DBException {
13.       super();
14.       /* adicionar una llamada para dada objeto para que
15.       sea miembro de este "Schema" */
16.       addDBObject("com.jcorporate.sales.Cliente");
17.   } /* constructor */
18.   /* Otros metodos son opcionales en el objeto "Schema"
19.   - ver the Javadoc */
20.   /* para mas detalles */
21. } /* Ventas */

```

Este simple “Schema” podría extenderse cuando usted adicione nuevos objetos a su aplicación.

- Compile el objeto “Ventas” y corrija cualquier error.

3. *Crear los enlaces en la página HTML*

Ahora usted creará un enlace a su nueva Base de Datos usando el Servlet DBMaint:

- Crear la página Web desde la que usted va a llamar a las funciones de mantenimiento: permítanos asumir que se llama “ventas.html”.

El diseño de la página no es importante para el funcionamiento del Servlet en sí. Nos concentraremos en los enlaces para llamar nuestro nuevo objeto.

- En esta página cree los siguientes enlaces:

Adicionar Clientes

```

<a href="/servlet/DBMaint?next=ventas.html
&dbobj=com.jcorporate.ext.ventas.dbobj.Cliente

```

```
&cmd=Add">Adicionar Cliente</a>
```

Listar Clientes

```
<a href="/servlet/DBMaint?next=ventas.html  
&dbobj=com.jcorporate.ext.ventas.dbobj.Cliente  
&cmd=List">Listar Cliente</a>
```

Buscar Clientes

```
<a href="/servlet/DBMaint?next=ventas.html  
&dbobj=com.jcorporate.ext.ventas.dbobj.Cliente&cmd=Search  
h">Buscar Clientes</a>
```

4. *Registrar su nuevo “Shema”*

Ahora usted le debe decir a Expresso sobre la existencia de su nuevo “Schema”, a fin de que la función de Inicialización pueda acceder correctamente a su objeto “Schema”:

- En su Navegador Web, vaya a las páginas de Expresso que usted bajo del sitio de JCorporate.
- Haga “clic” en el enlace “Setup” en la columna que esta en la izquierda.
- En la línea titulada “Application Schema Objects” haga clic en el signo – mas- para adicionar una nueva entrada.
- Para el campo “Schema Class File” entre, “com.jcorporate.ext.dbobj.Ventas”, el nombre del objeto para el “Schema” será creado.
- Para el campo “Schema Description” entre “Aplicación de Ventas”.

5. *Correr la función de Inicialización*

La función de inicialización creará la Tabla para su nuevo objeto y también insertará entradas de seguridad para el grupo de administradores “Admin” y el usuario administrador tendrá acceso al nuevo objeto creado.

- Haga “clic” en el enlace “Setup” de la columna izquierda de la página de Expresso.
- Seleccione el Servlet de inicialización “Initialize”. Deje todos los botones de opciones y haga “clic” en el botón “Run”.

- En un momento usted verá el mensaje de confirmación de que las Tablas han sido creadas e inicializadas y las entradas de seguridad han sido creadas.

6. *Entrar al Sitio como administrador “Admin”*

Vaya a la página de Expresso de Entrar / Salir del Sitio “Log In / Out” para entrar como administrador “Admin”.

Como su nuevo objeto es solamente accesible como administrador en este punto que es quien tiene el propósito de probarlo. Mas tarde por supuesto usted puede adicionar seguridad para otros grupos y usuarios.

Ahora usted puede probar sus nuevas funciones de mantenimiento de la Base de Datos, vaya a su página `ventas.html` y trate de usar las funciones de Adicionar, Listar y Buscar.

Este es por supuesto un ejemplo muy simple, pero estas pueden ser las bases para un aplicación mas completa.

CAPÍTULO 9. DESPLIEGUE DE COMPONENTES EXPRESSO PARA APLICACIONES

CAPÍTULO 10. MONITOREANDO Y VERIFICANDO LAS OPERACIONES DE LAS APLICACIONES EXPRESSO

Ahora que usted tiene su aplicación desarrollada y desplegada, ¿que herramientas están disponibles para que usted pueda monitorear y mantener su aplicación?. Usted seguro desea que los componentes individuales de Expresso y sus componentes personalizados estén operando correctamente, para asegurarse de que su aplicación se mantiene operacional y disponible a los usuarios y asegurase también que el rendimiento es aceptable y no decae con el tiempo.

Existen herramientas dentro de Expresso para verificar su funcionamiento y el de cualquier otra aplicación web. Estas herramientas son de dos tipo: Pruebas de Unidades y Pruebas de Rendimiento.

PRUEBAS DE UNIDADES

Expresso contiene el Marco de Trabajo JUnit, un Marco de Trabajo para Prueba de Unidades que permite que los componentes individuales de Expresso sean probados y verificados que funcionan correctamente. Las aplicaciones de Expresso pueden (y deben) también incluir un número de Pruebas de Unidades. Las Pruebas de Unidades difieren de las pruebas de Rendimiento (cubierto mas abajo) en que ellas tratan con componentes individuales, haciendo mas fácil localizar y corregir un problema antes de que un componente sea ensamblado en una aplicación final.

El objeto “Schema” para cada aplicación contiene una lista de todas las Pruebas de Unidades registradas para esta aplicación, y un objeto “Controller” esta disponible para correr estas Pruebas de Unidades:
`com.jcorporate.expresso.ext.controller.TestController`. Un enlace para correr este Controlador puede ser la página “Applications” dentro de Expresso, aunque este puede por supuesto también ser ejecutada desde la línea de comando.

PRUEBAS DE RENDIMIENTO

Las condiciones que afectan el rendimiento de una aplicación pueden cambiar con el tiempo. Un servidor puede estar muy concurrido, una red puede llegar a sobrecargarse, una falta de memoria puede manifestarse y así sucesivamente. Es importante que el rendimiento de su aplicación no deje de ser aceptable, y que cuando las condiciones cambien, siga siendo aceptable. Expresso tiene facilidades que lo pueden ayudar en este sentido.

En la página “Operation” de Expresso hay un enlace llamado

[Performance and Runtime Monitoring](#) que le permite monitorear el rendimiento de su aplicación y recibir alertas cuando el rendimiento de su aplicación se vuelve inaceptable.

de error). De esta forma, la Prueba de Rendimiento esta disponible para verificar que esta salida esperada es recibida, y reportar un problema si no es así.

Cuando cada Prueba de Rendimiento es ejecutada, esta es cronometrada: El cronometraje comienza cuando la URL es solicitada y termina cuando el flujo de entrada termina. Estos tiempos son comparados con tiempos almacenados para determinar si la respuesta fue recibida en un tiempo esperado, el tiempo “normal” para esta prueba. En adición a este tiempo “normal” (normal time), un tiempo de “advertencia” (warning time) y un tiempo “máximo” (max time) puede ser especificado. Si el tiempo de respuesta es mayor que el tiempo “máximo” esto es considerado un error, y la prueba falla. Si se toma mas tiempo del normal se reporta como “cuidado” (caution), si es mayor que el tiempo de “advertencia” se hace una advertencia, pero la prueba puede aun tener éxito.

USANDO PRUEBAS DE RENDIMIENTO. CONJUNTO DE PRUEBAS DE RENDIMIENTO.

Las Pruebas de Rendimiento pueden ser organizadas en “conjuntos”. Estos conjuntos permite una secuencia de pruebas a ser ejecutadas una después de otra, ideal para situaciones donde un resultado depende de uno anterior, Ej. donde sesiones son usadas.

CHEQUEO DE VITALIDAD (HEALTHCHECK)

El Chequeo de Vitalidad provee un Controlador para ejecutar una o mas conjuntos pruebas que son específicamente indicadas como parte de un Chequeo de Vitalidad periódico del sistema. La prueba define una bandera que indica si el conjunto será usado por el Chequeo de Vitalidad.

Una vez que usted ha definido una serie de Pruebas de Rendimiento, la utilidad Chequeo de Vitalidad es usado para ejecutar este periódicamente. Este puede, la mayor parte se realiza fácilmente definiendo un interprete de guiones o fichero de lotes que es ejecutado por el organizador de tareas de su sistema (cron, organizador del sistema, etc) en algún organizador especificado. Por ejemplo, el guión para un sistema Linux puede quedar así:

```
#!/bin/sh
java com.jcorporate.expresso.core.utility.ControllerRun
configDir=/home/expresso/expresso/expresso-web/WEB-
INF/config webAppDir=/home/expresso/expresso/expresso-web
db=site
controller=com.jcorporate.expresso.ext.controller.HealthChe
ck state=health
```

La línea que comienza con “java” es toda una línea dentro del intérprete de guiones, escrito aquí en múltiples líneas para mayor claridad.

Este guión puede entonces ser ejecutado, por ejemplo, cada hora a través del utilitario “cron” especificando una entrada “crontab” como se muestra a continuación”

```
SHELL=/bin/sh
MAILTO=root
0 * * * * healthcheck.sh
```

Cuando el Chequeo de Vitalidad se ejecuta, este lee la lista del conjunto de Rendimiento que ha sido definido para su inclusión en el Chequeo de Vitalidad y ejecuta estos, acumulando estadísticas de Rendimiento en la medida que este se ejecuta. Si cualquier URL no retorna la cadena esperada, o toma demasiado tiempo para ejecutarse, este construye un correo electrónico para enviar una notificación de “Evento”. El código del evento “HEALTH” es usado para determinar quien recibirá la notificación, la cual quedaría de la siguiente manera:

```
From: support@jcorporate.com
Sent: Tuesday, March 27, 2001 12:01 PM
To: mnash@jcorporate.com
Subject: ERROR: System Health Check
HealthCheck
HealthCheck at 2001-03-27 08:00:11
For database/context 'default'
There were no failures, no warnings, and 1 caution,
CAUTION: Test 6 (Center for Espresso) for URL
'http://www.jcorporate.com/components/
internal/Center.jsp?category=65'
```

```
responded slower than it's set normal time of 160
milliseconds.
It ran in 237 milliseconds.
>From Server:www.jcorporate.com,
Database/context:default (Default Database)
```

El ejemplo de arriba muestra los mensajes recibidos cuando una URL en particular se salió de su tiempo normal de respuesta, pero no sobre su tiempo de advertencia, y aun así se ejecuto exitosamente (Ej. retorno la cadena especificada).

Parte de la utilidad del proceso HealthCheck es que si este genera un evento de correo electrónico si no hay advertencia o cuidado, este correo será un evento exitoso, por lo que usted puede optar por solo ver los mensajes cuando hay un problema, aunque la llegada de correos exitosos son un buen indicador de que todo en el Sitio esta bien. Una condición de error que el Chequeo de Vitalidad por supuesto no puede capturar es cuando el servidor se cae, lo que provoca que el Chequeo de Vitalidad no se pueda ejecutar o cuando el servidor de correo esta indisponible para enviar correos. Si usted sabe, sin embargo, que usted debe obtener un correo electrónico cada una hora, o cualquier tiempo que se halla definido en el organizador, entonces usted debe emprender alguna acción si usted nota que el correo no ha llegado.

El Chequeo de Vitalidad genera un evento de sistema "HEALTH", el cual es enviado por correo a todos los usuarios suscritos de manera automática. El código de evento exitoso lo es si el Chequeo de Vitalidad no encuentra "cuidados", "advertencias" o "errores" y no lo es en caso contrario.

CONTROLADOR "RUNTEST"

El Controlador "RunTest" se usa para ejecutar una o mas pruebas. Este tiene la habilidad para correr una secuencia de conjuntos de pruebas, y correr esa secuencia múltiples veces, y opcionalmente correr múltiples hilos de ejecución de pruebas a la vez. Esto permite cargar el sistema para ser simulado, obteniendo

errores de Rendimiento y problemas que solo ocurren cuando el sistema este cargado.

OPTIMIZANDO EL RENDIMIENTO

Es importante concentrar sus esfuerzos en las áreas donde se puede obtener el mejor provecho, para hacer esto usted debe entender donde están los puntos cruciales de su aplicación. Expresso puede ayudar a esto, y entonces le ayudará a realizar un análisis mas detallado de estos puntos cruciales para asegurarse que estos se ejecuten tan rápido como sea posible.

FACTORES QUE INFLUYEN EN EL RENDIMIENTO

Existen una número de cosas que pueden disminuir el Rendimiento de sus aplicaciones Expresso. Chequeando esta lista cuidadosamente antes de comenzar cualquier ajuste, usted puede verificar que no este “manejando con el freno puesto” y que frecuentemente puede resolver los problemas de Rendimiento inmediatamente.

Alguno de los elementos a ver son:

- Sistema de Rastreo (Logging)

Log4j, el cual esta integrado a los mecanismos de rastreo construidos en Expresso, es un paquete muy apto para esta función, y su impacto en el rendimiento es muy pequeño comparado con otros métodos de Rastreo. Aunque la forma mas adecuada para agilizar el Rastreo es no rastreando nada, un cuidadoso ajuste del fichero `expressoLogging.xml`, y el fichero correspondiente para su aplicación puede resultar en grandes beneficios para el Rendimiento.

La primera cosa del Rastreo a mirar es por supuesto seleccionar el Rastreo apropiado.

AJUSTE DE LA CACHE

A menos que usted específicamente desactive la opciones que hacen esto, (ver la documentación del fichero de propiedades) Expresso mientras se ejecuta

acumulará información sobre la eficacia del Sistema de Cache de los DBObject. Esta información incluye cuantas operaciones de lectura (retrieve(), searchAndRetrieve(), find() y así sucesivamente) se han realizado para un objeto DBObject en particular (en un Base de Datos/Contexto determinado), y cuantas de estas lectura usaron la cache para encontrar el artículo que estaban buscando. La es por supuesto que generalmente hablando los artículos mas frecuentes pueden ser leídos de la cache, el lugar de recuperarlos de la Base de Datos, lo que agilizará enormemente la ejecución de su aplicación.

El Servlet “Status” es usado para recuperar esta información, el cual calcula el porciento de “hits” al objeto de Base de Datos que fueron suministrado por la Cache, lo que usted busca en ese listado es un número grande de “hits” con un porciento bajo de recuperación desde la cache. Esto indica un DBObject que esta sobrecargado por operaciones de lecturas dentro de su aplicación, el cual no esta siendo muy “cacheado”. El listado del “Status” también muestra el tamaño actual de la cache para este DBObject, si es cero, entonces puede que usted no tenga una entrada para este DBObject en la Tabla “Database Object Page Limits”, que se accede desde la página “Setup” en Expresso.

Existen un número de otros factores que pudieran considerarse cuando se ajusta el sistema de “Cache” para los objetos de Base de Datos:

- Memoria Disponible

El sistema de “Caching” puede usar cantidades sustanciales de memoria, por lo que usted debe chequear la cantidad de memoria disponible para su aplicación antes de incrementar demasiado el tamaño de la “cache” para los objetos. Usted puede ver una “instantánea” del uso actual de la memoria desde el Servlet “Status”, pero usted puede desear leer el fichero “log” para ver como ha sido el uso de la memoria con el tiempo. Usted puede ajustar el nivel de detalles del “Rastreo” (logging) para el objeto “core.cache.CacheManager” para ver reportes periódicos sobre la memoria. Si usted ve un número de instancias donde el administrador de la “cache” limpia la “cache” para obtener mas memoria (Ej. para traerlo al mínimo especificado – vea nuevamente la

documentación sobre el fichero de propiedades para mas detalles), entonces usted probablemente no puede incrementar la “cache” sin disminuir el rendimiento de su aplicación.

Asegúrese que esta lanzando su Máquina Virtual Java (o su servidor de aplicaciones) con la opciones apropiadas para tener suficiente memoria disponible para sus aplicaciones., porque usted puede tener suficiente memoria físicamente pero esto no significa que la Máquina Virtual Java use esta. vea la opción `-X` para su “Java runtime” para detalles de como incrementar este limite.

- Elementos “Cacheados”

Algunos objetos de Base de Datos pueden mostrar una alta actividad, y bajos “hits” en la cache, pero eso aún no es un problema. Esto es particularmente cierto para elementos que siempre son cacheados por Expresso, tales como, valores de Instalación (“Setup”), entradas de seguridad para objetos DBObjects y controladores, y otros cuantos.

- La efectividad del sistema de “caching” esta también influenciada por los patrones en los cuales su aplicación se basa para escribir hacia la Base de Datos, si un objeto DBObject en particular es cambiado, su entrada en la cache será eliminada (para que datos echados a perder no sean leídos) y este debe leer de la Base de Datos la próxima vez que este sea accedido. Si usted tiene un objeto que tiene asignada bastante cache, pero aun así no se “cachea” muy exitosamente, esto puede ser que este objeto esta siendo escrito muy frecuentemente.

Esto tiene que ver mas con el diseño de la aplicación que algo que este relacionado con el ajuste de la “cache”.

Con una cuidadosa manipulación de los artículos en la Tabla “Database Object Page Limit”, usted puede hacer un mejor uso de la memoria disponible a través del “caching” de los objetos de Base de Datos. Esta es frecuentemente el único y mas importante paso hacia la optimización que usted puede llevar a cabo.

OPTIMIZACIÓN DE LA BASE DE DATOS

Si todas las optimizaciones de arriba y técnicas ajustes de Rendimiento indican que hay un problema con el acceso a la Base de Datos que esta bajando el rendimiento del sistema (a pesar de ajustar la cache de la Base de Datos), entonces esto algunas veces requiere analizar las preguntas(queries) a la Base de Datos en si mismas para determinar cual pregunta esta causando el problema.

El objeto “DBConnection”, el cual es responsable de la ejecución de las preguntas SQL hechas por los objetos DBObjects, puede ayudar a realizar este análisis: active la prioridad de Rastreo (logging priority) para este objeto para “debugearlo” usando el fichero expressoLogging.xml, y monitorear los resultados. Cada pregunta SQL tendrá un tiempo transcurrido guardado, y usted puede usar esta información para seleccionar los peores y modificar su aplicación para lograr un mejor Rendimiento, o usar índices en las Base de Datos para que las operaciones sean mas rápida.