Dinámica
El framework RADical
…J2EE sin complicaciones

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Biblioteca Técnica
Guía del Programador

Última actualización: 2010-02-03
Requiere framework v2.1.4 en adelante

Contenido

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Sumario

Este documento explica en detalle el funcionamiento interno del framework Dinámica, su arquitectura y facilidades. Es una guía fundamental para el programador que desea tener un conocimiento cabal del framework, y también es un elemento básico en la documentación de aplicaciones web basadas en Dinámica, en tanto que explica la arquitectura que subyace a estas a aplicaciones. En este sentido se recomienda utilizar este documento como un entregable principal en sus proyectos basados en Dinámica.

Introducción

El framework Dinámica es un conjunto de componentes, herramientas y técnicas para construir aplicaciones web 2.0 (ajax) basadas en Java/J2EE usando únicamente el API de Servlets. Es liviano, eficiente y muy fácil de usar. Se obtienen resultados sofisticados en minutos gracias al estilo único de construcción de software y a una amplia colección de plantillas de módulos prefabricados. Es un framework orientado específicamente a la construcción de aplicaciones de negocios, con énfasis en la productividad y la calidad. Hace gala de un enfoque muy pragmático de construcción de software, fuertemente integrado con el entorno de programación Eclipse.

Es rico en contenido, con una amplia base de documentación original (más de 30 PDFs) que explican cada aspecto del framework, además de todo el código y herramientas que incorpora. Es software libre –licencia LGPL- escrito 100% en Java, por lo tanto portable.

Dinámica es una propuesta de ingeniería de software completa, no solo abarca el proceso de programación, sino que también incorpora facilidades para el control de las aplicaciones antes y después del pase a producción, tales como la generación automática de documentación y scripts de configuración, manejo centralizado de errores con auto-notificaciones por email, trazas de rendimiento transparentes al programador, servicios de seguridad, servicios de diagnóstico del proceso, trazas de auditoria declarativas, plugins para Eclipse, etc. La profundidad de Dinámica va mucho más allá de ser solo un enfoque de programación rápida.

Producido por:

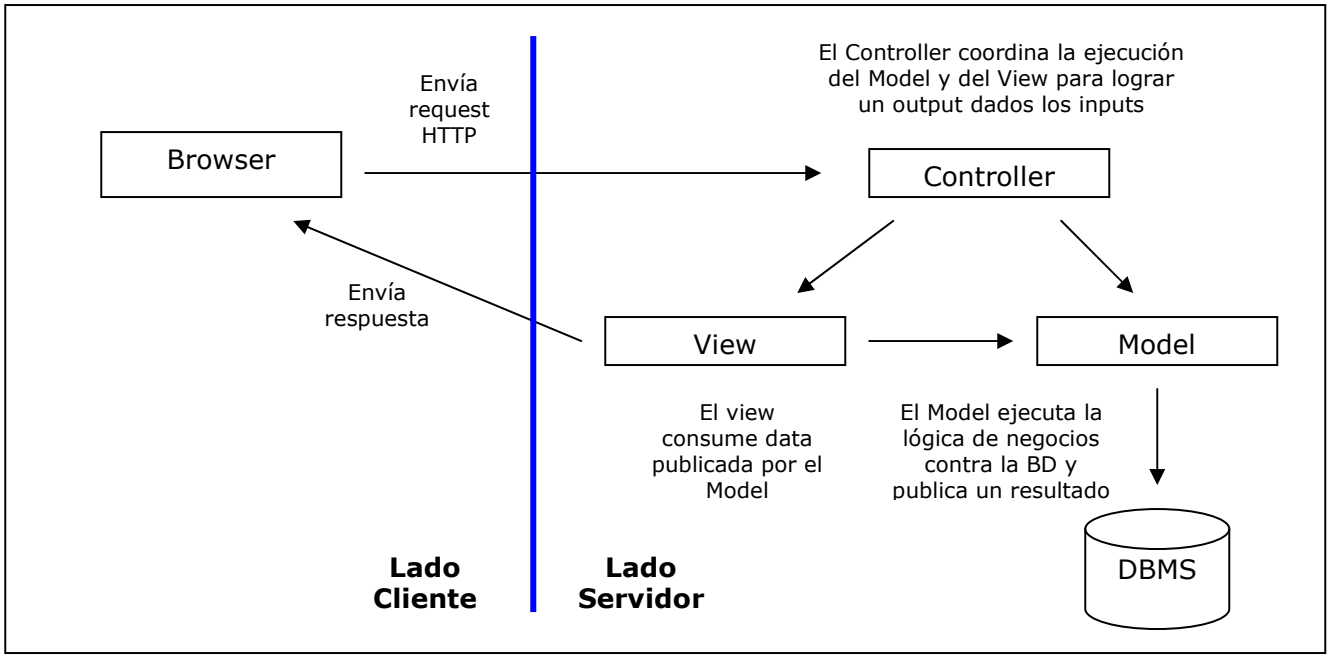
Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Arquitectura de software

El diseño de Dinámica se basa en una implementación flexible y novedosa de la arquitectura Model-View-Controller (MVC), que además de mantener el ordenamiento modular con una estricta separación de tareas también promueve la productividad mediante un mecanismo de programación declarativa. En Dinámica a menudo no es necesario programar, sino declarar mediante un archivo de configuración lo que se quiere hacer. En aquellos casos donde la programación es requerida, a menudo solo es necesario escribir una pequeña clase que representara el modelo y que por norma extiende a alguna de las clases base del framework, heredando cuantiosa funcionalidad y simplificando su trabajo.

La arquitectura MVC es muy adecuada para aplicaciones web escritas con lenguajes orientados a objetos, como Java. Dinámica le añade un toque especial a esta arquitectura que ayuda a minimizar el esfuerzo de programación y permite incorporar cambios de manera rápida y flexible, facilitando el mantenimiento, que es una de las principales fuentes de costos en el proceso de producción de software.

Modelo general de la arquitectura MVC



En la arquitectura MVC los requests o peticiones de un cliente son atendidos por un módulo central coordinador de toda la actividad, el *Controller*, este módulo se encarga de invocar al módulo *Model* que se encarga de ejecutar la lógica de negocios mediante accesos a la base de

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

datos, publica la data resultante, y luego el *Controller* invoca al módulo *View* para que se encargue de consumir esta data y genere una salida, sea una página HTML, un documento XML o una imagen. La lógica de presentación está estrictamente separada de la lógica de acceso a la base de datos, y de esto se desprende otra consecuencia interesante, el mismo *Model* puede ser utilizado para generar distintos outputs, usando diferentes Views. El *Model* no tiene dependencia en los *Views*, no sabe quién lo invoca, se limita a publicar la data. La misma data de un *Model* puede ser utilizada para generar una tabla HMTL o un gráfico de barras como una imagen JPEG. El desacoplamiento modular es uno de los beneficios principales de la arquitectura MVC, ya que promueve la reutilización, al menos a nivel de la lógica de negocios.

El estilo MVC de Dinámica

El diseño del framework Dinámica añade valor a los beneficios de la arquitectura MVC, haciendo cada una de las piezas mucho más genéricas, configurables mediante parámetros, de esta manera no solo hay desacoplamiento entre Views y Models, sino que incluso los Views serán independientes de los Models, y habrá un solo Controller para atender todas las solicitudes.

Usando Dinámica se puede construir una aplicación que no utilice sino los módulos genéricos que ofrece el framework, configurando los distintos Views y Models que provee Dinámica, con un estilo de programación declarativa. Una gran parte de las tareas comunes de una aplicación de negocios se pueden completar usando las clases genéricas de la arquitectura MVC de Dinámica, proveyendo parámetros adecuados para cada caso, sin necesidad de escribir una línea de código Java.

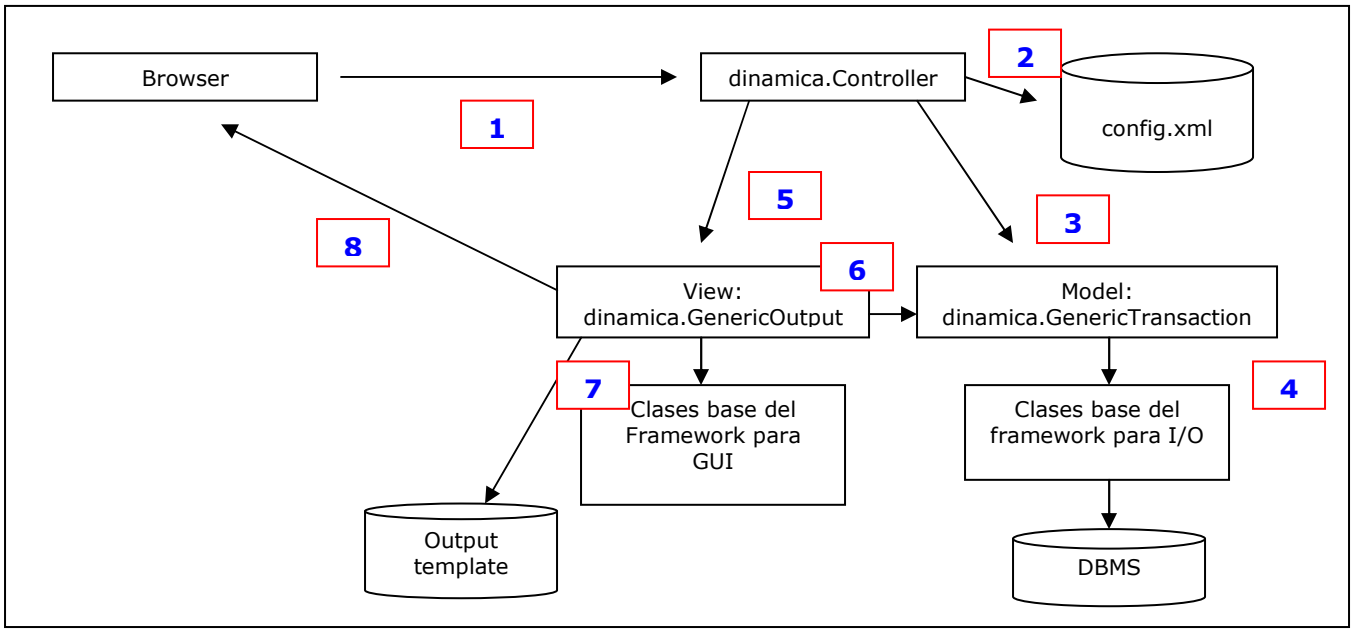
El Controller de Dinámica es un Servlet, el único que utiliza una aplicación basada en el framework. El Model y el View pueden ser clases provistas por el framework o por el programador que escribió las suyas extendiendo las clases del framework para ahorrar esfuerzo. Las clases que aporta una aplicación pueden ser suficientemente genéricas (configurables mediante parámetros) como para que sean de utilidad en otras aplicaciones, Dinámica promueve este estilo de programación, para que cada proyecto vaya enriqueciendo al framework. Los mecanismos básicos de programación para extender al framework son la herencia y la implementación de interfaces, y en el caso de la herencia es mucho el trabajo que se ahorra gracias a la fuerte factorización de código de utilidad común (independiente del dominio) que existe en la estructura de clases de Dinámica.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

El mecanismo de procesamiento de solicitudes HTTP

Cada vez que un cliente HTTP enviá una solicitud a una aplicación basada en Dinámica, dependiendo de la ruta del URL se determinará si corresponde a Dinámica procesarla, de ser así esta solicitud será pasada al Controller para que la atienda. Las solicitudes que atiende Dinámica se denominan ***Actions***. Simplificando un poco, una aplicación web basada en Dinámica no es más que un conjunto de ***Actions***, recursos estáticos (imágenes, html, css, etc) y clases, si es que se requiere escribir alguna.



El ***Action*** es la unidad básica de ejecución en Dinámica, es el equivalente a una página JSP o PHP, es el proceso del lado servidor que atiende a una solicitud. Detrás de un Action se mueve todo el mecanismo MVC que se muestra en el diagrama de arriba. A continuación la explicación del mecanismo de acuerdo a los números del diagrama:

1. El browser envía el request, si el URL contiene un texto como "/action/xxxx" el request será pasado al Servlet que funge como Controller para que lo atienda.

2. El Controller computará la ruta de una carpeta dentro de la aplicación (una aplicación web se organiza en un conjunto de carpetas) a partir del URL. Por ejemplo si la ruta dice "/action/xxxx" entonces el Controller buscará la carpeta /WEB-INF/action/xxxx dentro de la aplicación, y dentro de esa carpeta leerá el archivo de configuración del Action (config.xml). Un Action se representa con una carpeta que contenga este archivo, el mismo declara la configuración de parámetros para ejecutar el Action, tales como la clase que

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

representa el Model, el View, que recursos utilizar (plantillas html, sql, etc) y otras características.

3. Luego de que el Controller lee la configuración del Action, procede a ejecutar el mecanismo MVC. Instancia a la clase que representa al Model, y ejecuta el método con la lógica de negocios.

4. Este Model lee lo que le corresponde de la configuración del Action (que SQLs debe ejecutar, si usa o no transacciones de base de datos, etc) y procede a ejecutar la lógica de negocios apoyándose en APIs de alto nivel para acceso a bases de datos SQL, al terminar puede publicar la data resultante, usando una abstracción fundamental del framework: Recordsets. El Recordset es una clase que representa una estructura de datos tabulada, columnas y registros. Es un recordset desconectado, una herencia útil de los tiempos de Visual Basic. La data contenida en el Recordset puede provenir de un query SQL o de cualquier otra fuente, es desconectado porque no mantiene conexiones abiertas a la base de datos. Es un medio de intercambio de data tabulada entre las clases del framework, particularmente entre el View y el Model, pero se usa en otros casos también.

5. El Controller detecta que el Model terminó su tarea, y procede a instanciar la clase del View y ejecutar el método generará el output o respuesta del Action.

6. El View procede a consumir los Recordsets que haya publicado el Model, usando interfaces bien definidas entre estos tipos de módulos. El Model no sabe que tipo de View consumirá su data, y el View no sabe que tipo de Model publica la data, le basta que le publiquen lo que la configuración dice que debe consumir. El desacoplamiento es aun más extremo que en un modelo MVC tradicional, y por ende hay mayor grado de reutilización.

7. El View utiliza componentes utilitarios del framework para cargar la plantilla de página a partir de la cual se generará la respuesta, y procede a "inyectar" la data de los Recordset del Model dentro de la plantilla. Esto se hace mediante un proceso llamado "data binding", que permite declarar en la plantilla donde van los campos, y el framework se encarga de sustituir estas marcas por la data actual, aplicando formatos de máscaras y codificaciones de caracteres especiales si así fue indicado.

8. Una vez que la respuesta ha sido generada en memoria con técnicas optimizadas a partir de los recordsets y la plantilla, el View la envía al browser, completando la respuesta a la solicitud y cerrando el circuito del mecanismo MVC.

Tenga siempre presente estos conceptos:

- Request = Action
- Al recibir un request se ejecuta un Action, lo que dispara mecanismo MVC de Dinámica para procesar los inputs y generar un output. Se puede coordinar un Action mediante configuración únicamente, sin necesidad de programar nada.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Orientación del diseño

El enfoque de Dinámica hacia la arquitectura MVC es sustancialmente distinto al de otros frameworks más ortodoxos, donde cada entidad de la base de datos resulta modelada como una "clase del dominio". Son técnicas intensivas en código y en mano de obra. En contraste, Dinámica utiliza la tecnología de orientación a objetos para crear abstracciones que resuelven problemas técnicos comunes a todas las aplicaciones de negocios: cómo procesar y validar un formulario sin programar, cómo mostrar una gráfica, cómo mostrar un PDF, etc.

Al resolver estos problemas de manera limpia y eficiente se generan soluciones a problemas de negocios, con un proceso de producción que es intensivo en tecnología en vez de serlo en mano de obra. Y al haber menos clases particulares gracias a la programación declarativa que hace uso de clases genéricas, hay menos errores, y el mantenimiento se hace más sencillo. Por otro lado cuando hay que aportar una clase particular, a menudo resulta ser que con un poco de diseño inteligente se la puede hacer más genérica y de utilidad amplia, no solo para el proyecto donde fue creada.

La selección de abstracciones utilitarias de amplio espectro resulta directamente en mayores niveles de reutilización a través de distintos dominios de aplicaciones.

Principios generales del diseño

Diseño abstracto: Usar abstracciones, separando interfaces de implementación, esto permite extender el framework con diversas implementaciones de la misma interfaz, respetando la semántica y reglas de la interfaz, reduciendo los mecanismos de intercomunicación entre los módulos, lo que ayuda a la simplificación del sistema y a su extensión para adaptarse a nuevos requerimientos.

Resiliencia/Diseño Flexible: Crear abstracciones que se adaptan bien a los cambios de requerimientos, permiten el crecimiento y la modificación ordenada del sistema sin alterar su arquitectura modular.

Separación de tareas: Mantener relacionados los módulos que tratan problemas comunes, no mezclarlos con módulos que no tienen que ver con el problema.

Simplicidad: Enfocarse en mecanismos simples, fáciles de entender en el futuro.

Producido por:

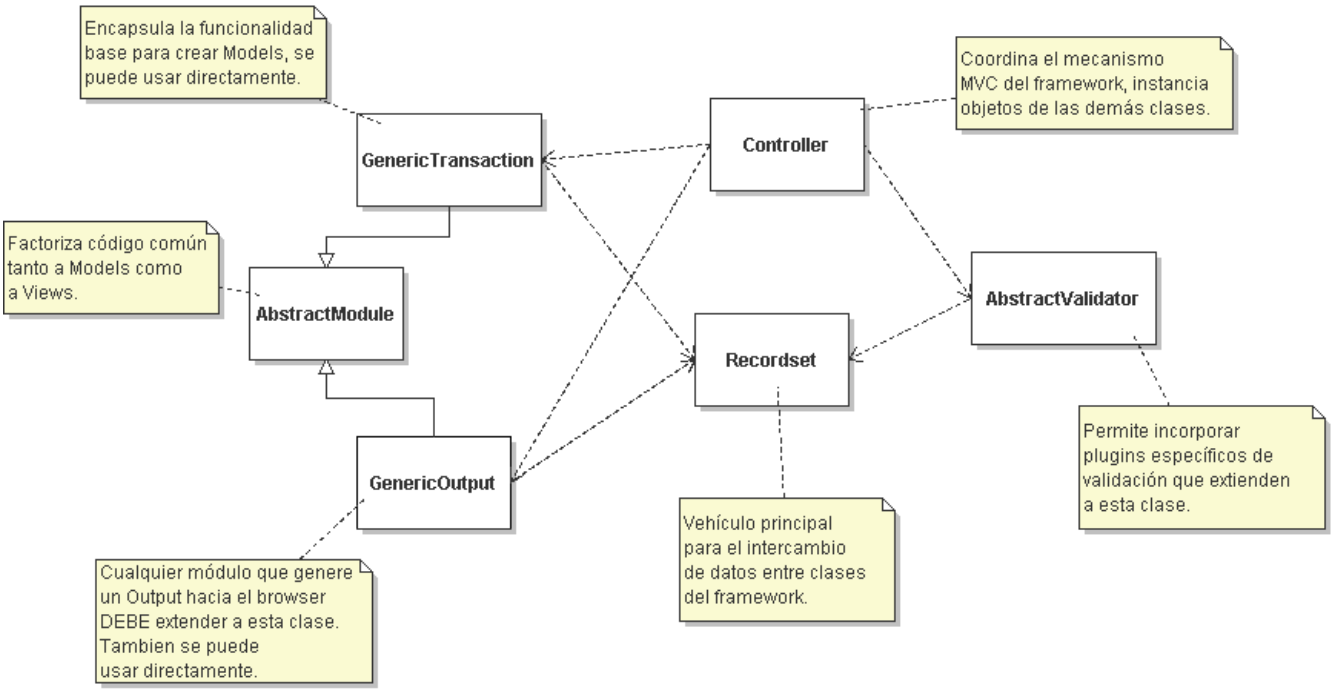
Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Estructura de clases

Esta sección describe por partes la estructura de clases del framework, ya que sería muy engorroso hacerlo en un solo diagrama. Por favor tome en cuenta que el framework crece en cantidad de clases cada cierto tiempo y estos diagramas podrían no estar actualizados. Sin embargo la estructura jerárquica fundamental no cambiará y esta bien plasmada en estos diagramas. Del website de Dinámica puede descargar la documentación completa del código fuente de todas las clases, en formato JavaDocs.

Clases básicas

Son las clases principales del mecanismo MVC del framework, y aunque conforman la base más abstracta, algunas de ellas contienen funcionalidad suficiente como para ser utilizadas directa y frecuentemente, otras son estrictamente abstractas, definen solamente la interfaz para las clases que la implementadotas y consumidoras.

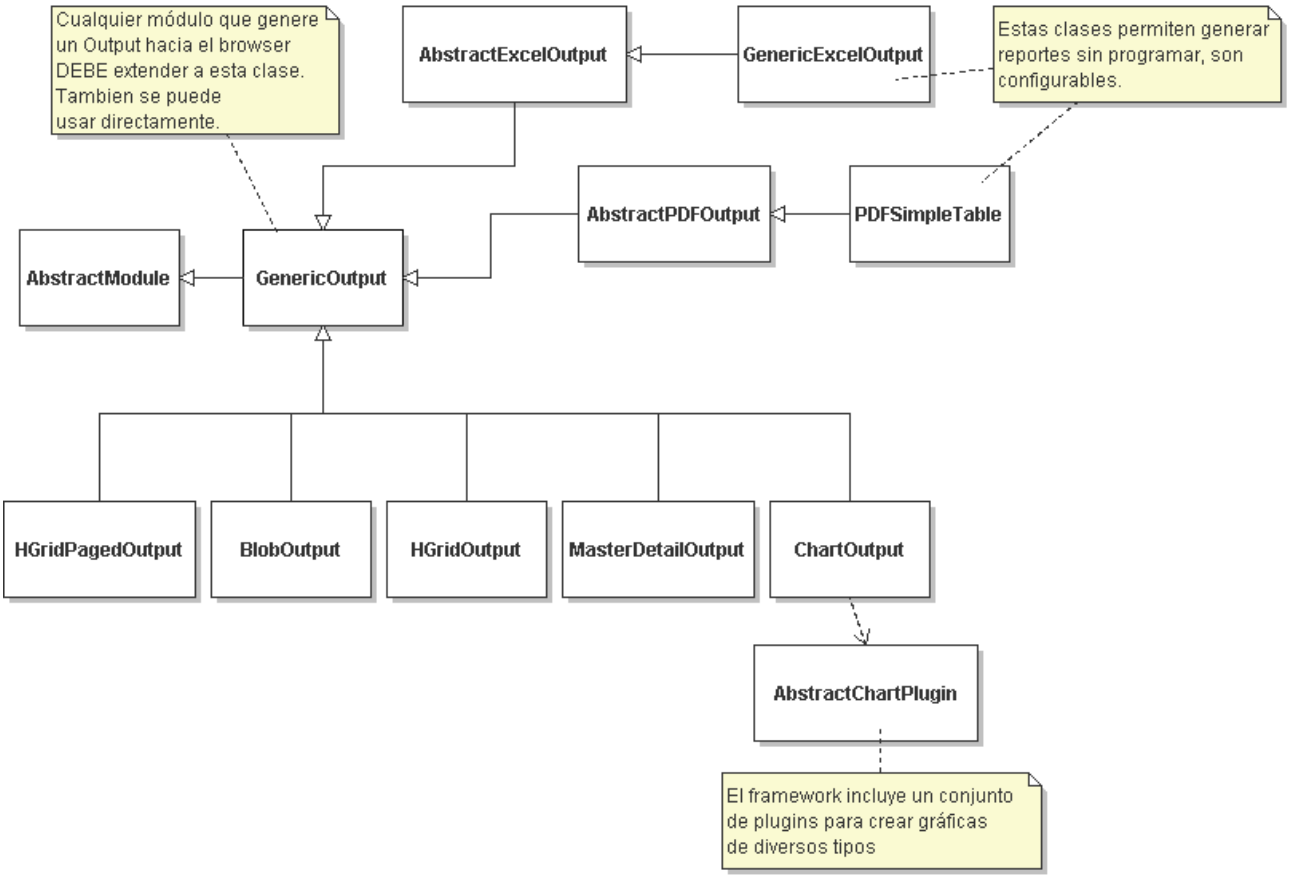


Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Clases de tipo Output (View)

A partir de la clase View básica GenericOutput que sirve para generar salidas de tipo texto (html/xml/javascript/text), se derivan un conjunto de clases muy especializadas, que abarcan generación de PDFs, Excel, reportes anidados, galerias tipo thumbnail, gráficas de negocios, etc. Algunas de estas clases se pueden usar mediante configuración sin necesidad de programar, otras son más abstractas y sirven de base para otros módulos más complejos. Algunos de estos módulos encapsulan el uso de componentes open source de terceros, como IText para PDFs o JFreeChart para las gráficas.

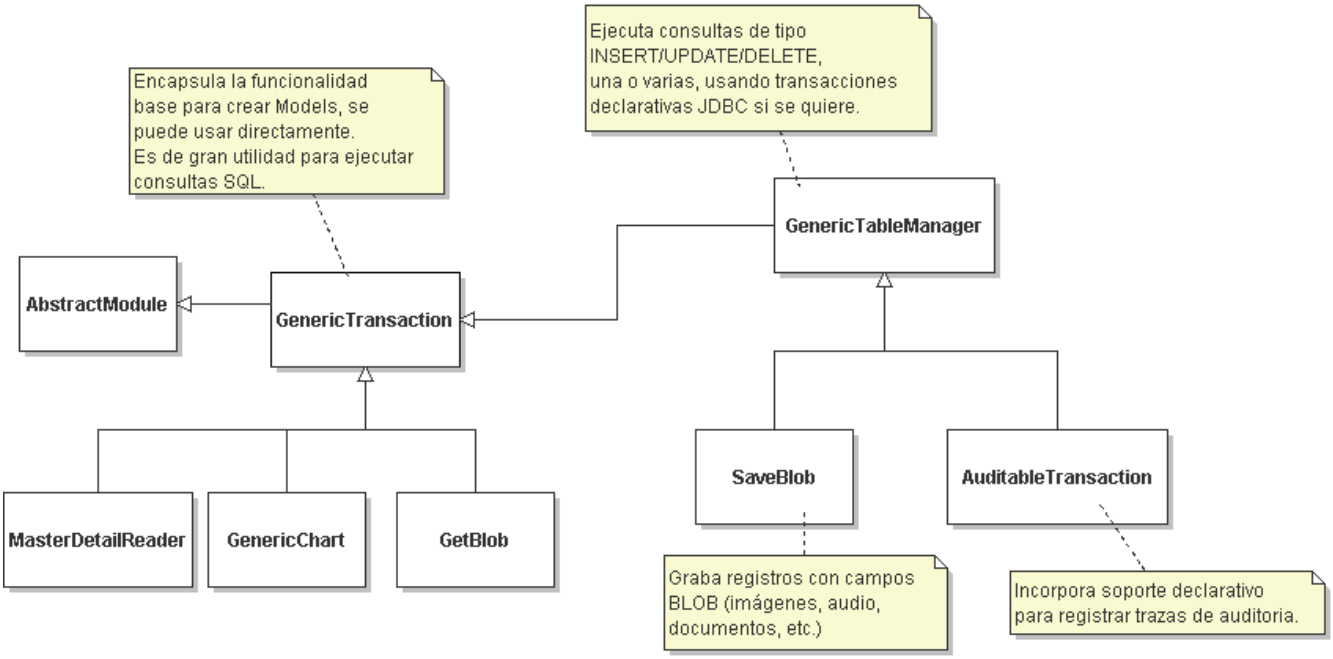


Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Clases de tipo Transaction (Model)

La clase GenericTransaction es la clase base para todos los Models que incluye el framework. Esta clase factoriza una buena cantidad de código, aliviando considerablemente el trabajo de sus subclases. La otra clase de uso muy frecuente es GenericTablerManager, que sirve para consultas de modificación y creación de registros. Ambas clases pueden ejecutar cualquier cantidad de queries, y el control de las transacciones de base de datos es declarativo, a nivel del Action. Además soportan la composición de objetos, esto es que una clase puede llamar a otras, y todas forman parte de la misma transacción. Si va a escribir su propia clase de tipo Transaction, lo mejor y lo común es extender a GenericTransaction o a GenericTableManager.



Clases utilitarias

Las clases que constituyen el mecanismo MVC del framework se apoyan en otras clases que proveen un API de servicios fundamentales, y conforman el "core" o API de "bajo nivel" por decirlo así, a continuación se mencionan las mas relevantes:

- **dinamica.Db**: Encapsula el protocolo JDBC, se utiliza sobre todo desde clases de tipo Model o Transaction. Reduce considerablemente el trabajo de escribir lógica de negocios

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

contra bases de datos SQL. Además enriquece los mensajes de error con información que resulta muy útil a la hora de diagnosticar problemas en consultas SQL.

- **dinamica.Recordset**: Implementa una abstracción fundamental del framework, la de un recordset desconectado, una potente estructura de datos tabulada que sirve de vehículo de transferencia de data entre los distintos módulos del framework, ayudando a reducir la complejidad de las interfaces intermodulares. Es serializable y puede ser retornado como parámetro de Web Services con JAX-WS (si el cliente es Java y usa el API de Dinámica).

- **dinamica.TemplateEngine**: Manipula todo tipo de plantillas basadas en texto, sean HTML, XML, JavaScript o SQL. Participa activamente en casi todo lo que se hace con el framework, y es sin duda una de las clases más importantes y de mayor complejidad.

Puntos de extensión frecuentes

- Cuando surge la necesidad de escribir clases propias de una aplicación, lo mas común es que se trate de Models especializados, que hacen algo que los Models que incluye el framework no pueden hacer, y normalmente son muy simples porque solo deben extender a uno de los Models existentes, añadiendo muy poco código (en general).

- Otra opción frecuente es la creación de nuevos "Custom Validators", que son plugins que extienden las reglas de validación de parámetros de un request o solicitud HTTP. Para esto puede crear el plugin de cero, extendiendo a dinamica.AbstractValidator, o extender alguno de los ya existentes, o en última instancia tomar muestras del código de alguno de estos validators para crear su versión particular, pero recomendamos que siempre evalúe con detenimiento si en realidad es necesario añadir un módulo, muchas veces los que ya existen en el framework cubren su necesidad.

- También puede añadir nuevos módulos de tipo View o plugins de Charts, pero esto no es tan común, y en el caso de los Charts lo más frecuente es crear una versión especializada de uno ya existente, en vez de extender a dinamica.AbstractChartPlugin. Hay ejemplos interesantes de esto en las demos de Dinámica.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Funcionamiento del framework

Dinámica resuelve a grandes rasgos dos problemas fundamentales de la programación de aplicaciones web:

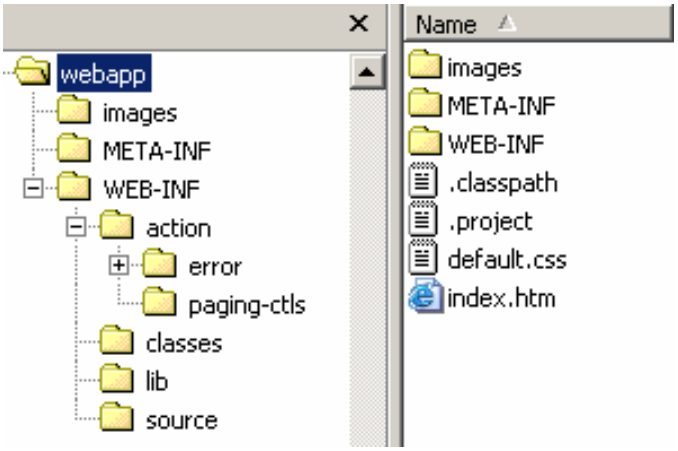
- Validación de los parámetros de un request o solicitud HTTP.
- Generación de la respuesta a un request.

Estos dos tópicos estan detrás del procesamiento de formularios, vinculación entre páginas, reportes, PDF, Excel, generación de gráficas, vistas tabuladas paginadas, etc. Todo lo que se pueda imaginar en una aplicación web, del lado servidor se reduce a procesar inputs y generar un output.

Al diseñar Dinámica, nos enfocamos en las tareas más tediosas, que generan mas costo y errores en el proceso de producción, y se crearon mecanismos para facilitarlas.

A continuación se explica en detalle como funcionan los diversos mecanismos del framework que entran en funcionamiento para procesar un Action.

El Controller



Una aplicación web/J2EE –o webapp- es una carpeta con una organización particular dentro de ella, y tiene una subcarpeta especial llamada WEB-INF que no visible para los browsers y es especial porque contiene el código de la aplicación, el archivo de configuración central de la webapp llamado web.xml, además de otras cosas que el programador decida colocar.

Como ya se explicó, cada request en Dinámica es atendido por un Action, que no es más que una carpeta ubicada dentro de /WEB-INF/action y que DEBE contener un archivo config.xml y otros recursos, como las plantillas de los comandos SQL a ejecutar y la plantilla HTML para generar la salida. Este archivo config.xml le dice al Controller como coordinar toda la operación para atender al request, instanciando las clases del Model y el View, además de los plugins de validación.

El Controller es una clase de tipo Servlet, y se define y configura en el archivo web.xml, siguiendo el estándar J2EE para Servlets. Los Actions no tienen nada que ver con el estándar J2EE, es una abstracción añadida por el framework, que se monta sobre el estándar de Servlets. Gracias a la

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

configuración del Controller en web.xml, los requests HTTP que van dirigidos a ejecutar un Action son interceptados por el Controller y entonces este procede a disparar el mecanismo MVC de Dinámica:

```
<!--Controller Servlet to intercept requests-->
<servlet>
      <servlet-name>Controller</servlet-name>
      <description>Request Controller for the MVC mechanism</description>
      <servlet-class>dinamica.Controller</servlet-class>
      <init-param>
            <param-name>base-dir</param-name>
            <param-value>/action</param-value>
            <description>
            Base string to build path starting
            from /WEB-INF/ to search for the action configuration file
            </description>
      </init-param>
      <load-on-startup>1</load-on-startup>
</servlet>

<!--URLs intercepted by the Controller-->
<servlet-mapping>
      <servlet-name>Controller</servlet-name>
      <url-pattern>/action/*</url-pattern>
</servlet-mapping>
```

Esto define el Servlet del Controller

Esto hace que el Controller intercepte el request

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Programación declarativa con config.xml

En el archivo config.xml de un Action se declara lo que desea que el Action ejecute en cada etapa del mecanismo MVC. Con el siguiente ejemplo se irá explicando lo esencial de este archivo:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<config>

      <summary>
            Desplegar picklist de productos
      </summary>

      <log>false</log>

      <transaction>
            <classname>dinamica.GenericTransaction</classname>
            <validator>true</validator>
            <transaction>false</transaction>
            <jdbc-log>false</jdbc-log>
            <recordset id="query.sql" source="sql" scope="transaction" />
      </transaction>

      <output>
            <classname>dinamica.GenericOutput</classname>
            <template>template.htm</template>
            <set-http-headers>true</set-http-headers>
            <content-type>text/html</content-type>
            <expiration>0</expiration>
            <print mode="table" recordset="query.sql" tag="rows" alternate-
colors="true"/>
      </output>

</config>
```

Documenta el Action, esto se usa para generar automáticamente documentación de todos los Actions de una webapp.

Se usa para generar una traza de rendimiento del mecanismo MVC.

Configuración del Model

Configuración del View

El archivo tiene dos secciones importantes: <transaction> y <output>. La primera describe lo que hará el Model, la segunda lo que hará el View.

El Model

```
<classname>dinamica.GenericTransaction</classname>
```

Con esto indicamos la clase o módulo que representa al Model de este Action. Generalmente es una clase genérica del framework, pero podría ser una clase particular de la aplicación. El framework incluye varios tipos de Models de uso general en diversas aplicaciones de negocios.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

```
<validator>true</validator>
```

Con esto indicamos que usaremos los servicios de validación automática del framework, y el Controller entonces leerá el archivo validator.xml que describe las reglas de validación que aplican a este Action. Más adelante se explica este archivo, por ahora lo que es importante entender es que el framework hace la validación de los inputs o parámetros del request, y es un mecanismo extensible, podemos proveer plugins para resolver reglas complejas, aunque el framework ya incluye varios de uso general.

```
<transaction>false</transaction>
```

Si queremos transacciones de base de datos controladas vía JDBC podemos indicarlo con este elemento. En este ejemplo solo hacemos consultas, así que no usaremos transacciones. Cuando las usamos, el framework se encarga de hacer el commit o el rollback por nosotros, dependiendo del resultado de las operaciones del Model.

```
<jdbc-log>false</jdbc-log>
```

Con esto indicamos si queremos dejar trazas de la comunicación con la base de datos, son trazas detalladas y muy útiles si necesitamos diagnosticar problemas de rendimiento con algún comando SQL. Hay un documento completo sobre el manejo de trazas de rendimiento con Dinámica en el website del framework.

```
<recordset id="query.sql" source="sql" scope="transaction" />
```

Esta es la parte más importante. Acá declaramos que se debe crear un recordset desconectado, a partir de un comando SQL que se llama "query.sql" y que está almacenado en el mismo directorio del Action. El Recordset resultante solo se usará durante el procesamiento de este Action o request, eso lo indica el atributo "scope", porque existe la opción de almacenarlo en el request, o en la sesión. Solo con indicar esto, el framework se ocupa de obtener una conexión a la base de datos, ejecutar el query y leer toda la data y montarla en el Recordset, usando técnicas avanzadas de programación JDBC, y además maneja correctamente los casos de error, sin dejar conexiones huérfanas o recursos sin cerrar. Es el poder de la programación declarativa.

Se pueden crear tantos recordsets como sean necesarios.

En este ejemplo, el Model solo debe ejecutar un query, y el recordset resultante es publicado con el nombre "query.sql" para que pueda ser consumido por el View. En la imagen que aparece a la derecha se puede visualizar como están organizados los recursos de este Action en su carpeta.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

El View

```
<classname>dinamica.GenericOutput</classname>
```

Indica la clase a utilizar para representar al View, esta es una de las más comunes, el framework incluye otras para tareas más especializadas. El programador puede suplir la suya, pero no suele ser la norma.

```
<template>template.htm</template>
```

Indica que se debe cargar esta plantilla desde el mismo directorio del Action, esta plantilla contiene marcas especiales para inyectarle la data de los recordsets. Estas plantillas con simples páginas HTML con unas marcas especiales (markers), no se usa lenguajes tipo JSP en Dinámica.

```
<set-http-headers>true</set-http-headers>
<content-type>text/html</content-type>
<expiration>0</expiration>
```

Esto nos da control sobre los encabezados de la respuesta http, permite indicar el contenido de la respuesta al browser, así como las directivas de caché. La clase del View no sabe que tipo de contenido está generando, por eso es importante indicárselo de esta manera.

```
<print mode="table" recordset="query.sql" tag="rows" alternate-colors="true"/>
```

Esta es la parte más importante, el "data binding" o asociación de la plantilla HTML con los Recordset publicados por el Model. Esta orden le dice al View que debe consumir un recordset llamado "query.sql", buscar en la plantilla una sección encerrada en un tag llamado "rows", y usar los markers o marcas de campos que encontrará en esa sección para imprimir N veces (por cada registro del recordset) los campos de cada registro. Adicionalmente se indica que se deben utilizar colores alternados en las filas de la tabla HTML resultante. Básicamente asociamos un grid a un datacontrol, para hablar en términos de Visual Basic.

La sección de la plantilla que será afectada tiene esta estructura:

```
<table class="grid" width="100%">
    <rows>
    <tr ${fld:_rowStyle} onmouseover="rowOn(this)" onmouseout="rowOff(this)"
          onclick="selectItem('${fld:productid}')">
          <td style="font-size:9pt">
                <span id="${fld:productid}">${fld:productname}</span>
          </td>
    </tr>
    </rows>
</table>
```

Sección que se repetirá N veces por cada registro

Markers de campos

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Aparecen en amarillo resaltados los markers de campos que serán sustituidos por la data de los campos, N veces, una por cada registro del recordset.

Existen directivas para distintos tipos de databinding, como llenar un formulario, seleccionar un elemento en un combobox, marcar unos checkboxes, etc. Todo se hace de manera declarativa. Las posibles configuraciones del archivo config.xml están documentadas en la guía de referencia de Dinámica, disponible en el website del framework.

Validación de parámetros del request

Cuando el Controller determina que se requiere validación automática de parámetros, buscará en la carpeta del Action el archivo validator.xml. Abajo se muestra uno de ejemplo:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<validator id="viewchart.filter">

      <parameter id="fdesde" type="date" required="true" label="Fecha desde"/>
      <parameter id="fhasta" type="date" required="true" label="Fecha hasta"/>

      <custom-validator
            classname="dinamica.validators.DateRangeValidator"
            on-error-label="[Fecha desde] no puede ser mayor que [Fecha hasta]."
            date1="fdesde" date2="fhasta" />

</validator>
```

Los servicios de validación de Dinámica garantizan que si alguno de los parámetros no cumple con las reglas requeridas, entonces el Model no será ejecutado. Esto es un contrato estricto, el programador puede contar con ello.

El elemento "parameter" define el parámetro, el tipo de dato y si es requerido o no, además del nombre descriptivo, que será utilizado para los mensajes de validación. Los servicios de validación verifican que el parámetro cumpla con el tipo de dato que debe representar (date, varchar, integer o double) y verifican que no sea nulo si es requerido.

Los elementos "custom-validator" permiten añadir un plugin a la cadena de validaciones, solo son evaluados si se pasan las validaciones básicas de los "parameter". Los plugins de validación extienden a la clase dinamica.AbstractValidator, y son clases muy simples aunque pueden resolver reglas complejas de validación. El framework incluye varios plugins de fábrica para casos comunes de negocios.

Si la validación es exitosa el Controller creará un Recordset con un solo registro, conteniendo los parámetros del request, representados con tipos de datos nativos de Java, listos para ser utilizados por el Model. Este Recordset se publica automáticamente con el nombre "_request", y es pasado al Model y también a los plugins de validación.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Los plugins de validación son evaluados en el orden en que son declarados, y si alguno falla los siguientes no son evaluados, el request es abortado con el mensaje apropiado de validación.

Otro detalle importante:

```
<validator id="viewchart.filter">
```

El atributo "id" indica que el recordset resultante de la validación (se suele denominar inputParams) deberá ser almacenado en la sesión con el nombre indicado en el valor de este atributo.

Aparte de todo el trabajo nada trivial que ahorran los servicios de validación, tienen otra consecuencia importante, y es que ahora el Model podrá utilizar los APIs del framework para manipular las plantillas SQL y sustituir automáticamente los valores de los parámetros dentro del SQL, siguiendo las normas de SQL portable y aplicando protección contra ataques de inyección de SQL, todo de manera transparente al programador.

Cuando en el config.xml se mandó a crear este Recordset:

```
<recordset id="query.sql" source="sql" scope="transaction" />
```

El framework cargó esta plantilla:

```
select
      productid, productname
from
      demo.products
where
      categoryid= ${fld:categoryid}
order by
      productname
```

El validator.xml del ejemplo contenía estas reglas:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<validator>
      <parameter id="categoryid" type="integer" required="true" label="ID
category"/>
</validator>
```

Y de esa manera el framework sustituye automáticamente el parámetro validado y lo inyecta dentro de la plantilla SQL.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Generación declarativa de respuestas

El framework incluye módulos de generar diversos tipos de salidas, desde páginas HTML simples, hasta documentos Excel. En la mayoría de los casos el control es declarativo, mediante config.xml.

A continuación se muestran algunos ejemplos, refiérase a la galería de plantillas en el website del framework para documentación completa y específica de cada caso.

Generación de reportes simples en PDF

```
<output>
      <classname>dinamica.PDFSimpleTable</classname>
</output>

<pdf-table recordset="query.sql" width="100">
      <logo url="/images/logo-dinamica.png" scale="100" />
      <col name="alias" title="Alias" width="20" align="center" />
      <col name="lname" title="Apellido" width="40" align="center" />
      <col name="fname" title="Nombre" width="40" align="center" />
</pdf-table>
```

Exportación a Excel

```
<output>
      <classname>dinamica.GenericExcelOutput</classname>
</output>

<excel recordset="query.sql" sheetname="Lista de Contactos"
      filename="contactos.xls">
            <col id="alias" label="Alias"/>
            <col id="lname" label="Apellido"/>
            <col id="fname" label="Nombre"/>
</excel>
```

Exportar BLOBs de la base de datos hacia el browser

```
<transaction>
      <classname>dinamica.GetBlob</classname>
      <validator>true</validator>
      <transaction>false</transaction>
      <jdbc-log>false</jdbc-log>
```

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

```
</transaction>

<output>
      <!--for databases other than PostgreSQL use dinamica.BlobOutput-->
      <classname>dinamica.BlobOutputPGSQL</classname>
</output>

<!--custom element for THIS Action only-->
<!--
      set to true if you want to force a "Save as" dialog
      when the browser downloads the BLOB - your query-info.sql
      template MUST contain a "filename" column.
-->
<attach>false</attach>
```

Reportes Master/Detail

Estos reportes no se pueden emitir con GenericOutput por los niveles de anidamiento que requieren, subtotales por grupos, etc. Por eso se provee un módulo que automatiza la generación de estas salidas:

```
<output>
      <classname>dinamica.MasterDetailOutput</classname>
      <template>template.htm</template>
      <set-http-headers>true</set-http-headers>
      <content-type>text/plain; charset=iso-8859-1</content-type>
      <expiration>0</expiration>
      <print mode="form" recordset="query.sql" />
      <print mode="form" recordset="masterdetail.filter" />
</output>

<!--specific elements for master/detail html reports-->
<query-master>group-master.sql</query-master>
<query-detail>group-detail.sql</query-detail>
<group-template>group.htm</group-template>
```

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

El modelo de persistencia de Dinámica

Con este framework no existe el problema de "mapping" objeto-relacional, porque no se modelan las entidades del dominio del negocio como clases, las abstracciones son mas bien de tipo técnico, para facilitar el procesamiento de solicitudes HTTP y la interacción con bases de datos SQL, por ese motivo la integración de una BD relacional es algo natural en Dinámica. No hay necesidad de Hibernate ni frameworks de tipo "object-relational mapping". Es uno de los primeros shocks culturales a la hora de abordar Dinámica, sobre todo por parte de programadores que ya tienen una experiencia previa con otros frameworks Java, y se debe a la influencia de herramientas 4GL que tiene Dinámica en su diseño y conceptos fundamentales. Nos concentramos en utilizar la tecnología de programación orientada a objetos para resolver problemas básicos de la construcción de aplicaciones Web/SQL, nada más.

La mecánica de la persistencia es la siguiente:

1.- El browser hace un POST de un formulario, el request es interceptado por el Controller de Dinámica, el cual lo direcciona hacia un Action. La configuración del Action indicará si se requiere validación de parámetros. De ser así, el archivo validator.xml será leído por el Controller y aplicado para las reglas de validación de este request, de manera automática.

2.- Una vez que la validación se aplica y pasa satisfactoriamente la prueba, se crea un Recordset que contiene un solo registro, con un campo por cada parámetro, cada uno representado como un tipo nativo de Java (Date, String, Integer o Double).

3.- La clase Transaction del Action (dinamica.GenericTableManager por ejemplo) se encargará de recibir este Recordset cuando se invoca su método service(), que ejecuta la lógica de negocios. Esta clase leerá de la configuración del Action cada uno de los SQLs que deban ser ejecutados para modificar data (inserts, updates o deletes), y procederá a sustituir los parámetros usando el formato adecuado para SQL, en cada uno de estos comandos SQL, usando el Recordset provisto por el mecanismo de validación. Vía configuración se puede activar el uso de transacciones JDBC (begin, commit, rollback), sin necesidad de programar.

4.- Cada uno de los comandos SQL será ejecutado, posiblemente dentro de una transacción de base de datos si así se dispuso. Si llega a ocurrir un error, la transacción será revertida, sino será ejecutada con un "commit". El control transaccional es automático.

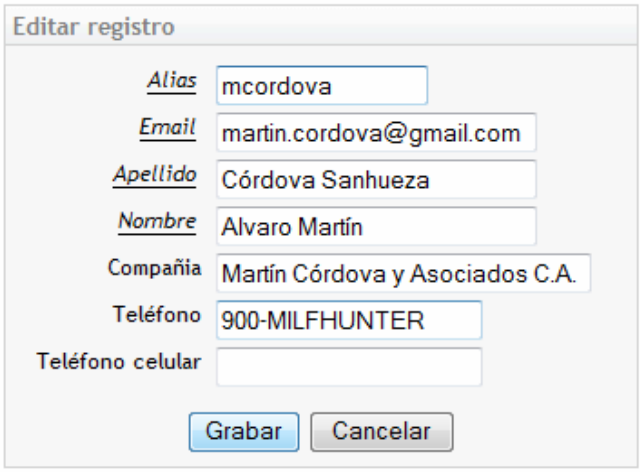
De esta manera un Action puede ejecutar uno o mas comandos SQL que modifican data, sin necesidad de programar nada, los parámetros del request se utilizarán para rellenar los valores correspondientes en los comandos SQL, usando formatos adecuados y protección contra ataques de inyección SQL.

El siguiente diagrama ilustra la secuencia de pasos y los artefactos utilizados por el framework.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Un formulario está a punto de enviar los datos mediante un POST ajax:



El framework valida el request usando un validator.xml dentro de la carpeta del Action:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<validator onerror="/action/error/validation/ajax">

      <parameter id="alias" type="varchar" required="true" label="Alias"
maxlen="20" />

      <parameter id="email" type="varchar" required="true" label="Email"
            maxlen="150" regexp="^([\w-]+\.)*?[\w-]+@[\w-]+\.([\w-]+\.)*?[\w]+$"
            regexp-error-label="Formato incorrecto, use xxxx@xxxxxx.xxx" />

      <parameter id="lname" type="varchar" required="true" label="Apellido"
maxlen="30" />
      <parameter id="fname" type="varchar" required="true" label="Nombre"
maxlen="30" />
      <parameter id="company" type="varchar" required="false" label="Compañia"
maxlen="80" />
      <parameter id="phone" type="varchar" required="false" label="Teléfono"
maxlen="20" />
      <parameter id="cellphone" type="varchar" required="false" label="Teléfono
celular" maxlen="20" />

      <custom-validator classname="dinamica.validators.DuplicatedKeyValidator"
            on-error-label="Este Alias ya se encuentra registrado, use otro Alias."
            sql="alias.sql" id="alias" />

</validator>
```

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Nótese que podemos extender las capacidades de validación básicas usando expresiones regulares y "custom validators", los cuales pueden provenir del framework (incluye varios genéricos) o ser provistos por nosotros mismos.

Luego de que el request ha sido validado, se procede a ejecutar la lógica de negocios, usando el config.xml como guía para el Controller:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<config>

	<summary>
		Insertar valores de un formulario en una tabla
	</summary>

	<log>false</log>

	<transaction>
		<classname>dinamica.GenericTableManager</classname>
		<validator>true</validator>
		<transaction>false</transaction>
		<jdbc-log>false</jdbc-log>
	</transaction>

	<query>insert.sql</query>

	<output>
		<classname>dinamica.GenericOutput</classname>
		<template>script.js</template>
		<set-http-headers>true</set-http-headers>
		<content-type>text/javascript; charset=iso-8859-1</content-type>
		<expiration>0</expiration>
	</output>

</config>
```

Los elementos "<query>" le indicar a la clase GenericTableManager que comandos SQL debe ejecutar, y procederá a reemplazar los markers que puedan tener estos comandos SQL por los parámetros pre-validados que correspondan.

El comando SQL de este ejemplo luce así:

```
insert into demo.address_book
(
	id,
	alias,
	email,
	lname,
	fname,
```

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

```
        company,
        phone,
        cellphone
)
values
(
        ${seq:nextval@demo.seq_addrbook},
        ${fld:alias},
        ${fld:email},
        ${fld:lname},
        ${fld:fname},
        ${fld:company},
        ${fld:phone},
        ${fld:cellphone}
)
```

Es importante destacar que con el plugin AmaterasERD que incluye el framework, basta pararse en el modelo de la BD, sobre una tabla, y al darle click al botón derecho podrá generar automáticamente tanto un validator.xml como sentencias INSERT o UPDATE, ahorrando mucho trabajo y evitando errores de tipeo o por desconocimiento de SQL o Dinámica.

Esta clase extiende a GenericTransaction, así que hereda todas sus capacidades, incluyendo la de crear Recordsets de manera declarativa. Es genérica, no está atada a ningún caso de negocios o dominio, solo se especializa en ejecutar de manera eficiente uno o más comandos SQL, ofreciendo facilidades para ello.

Se puede indicar que para un SQL en particular, se debe utilizar un Recordset particular, creado por este Action, y el comando será ejecutado una vez por cada registro que contenga el Recordset. Esto permite por ejemplo grabar un maestro y su detalle (de N registros) en un solo Action, sin programar y con transacciones JDBC:

```
        <transaction>
                <classname>dinamica.GenericTableManager</classname>
                <validator>true</validator>
                <transaction>true</transaction>
                <jdbc-log>false</jdbc-log>
                <recordset id="detail.sql" source="session" scope="transaction" />
        </transaction>

        <query>insert.sql</query>
        <query params="detail.sql">insert-detail.sql</query>
```

El modelo de persistencia de Dinámica está pensado para ahorrar trabajo en la medida de lo posible para los casos más comunes en aplicaciones de negocios, y hacerlo de manera robusta y eficiente.

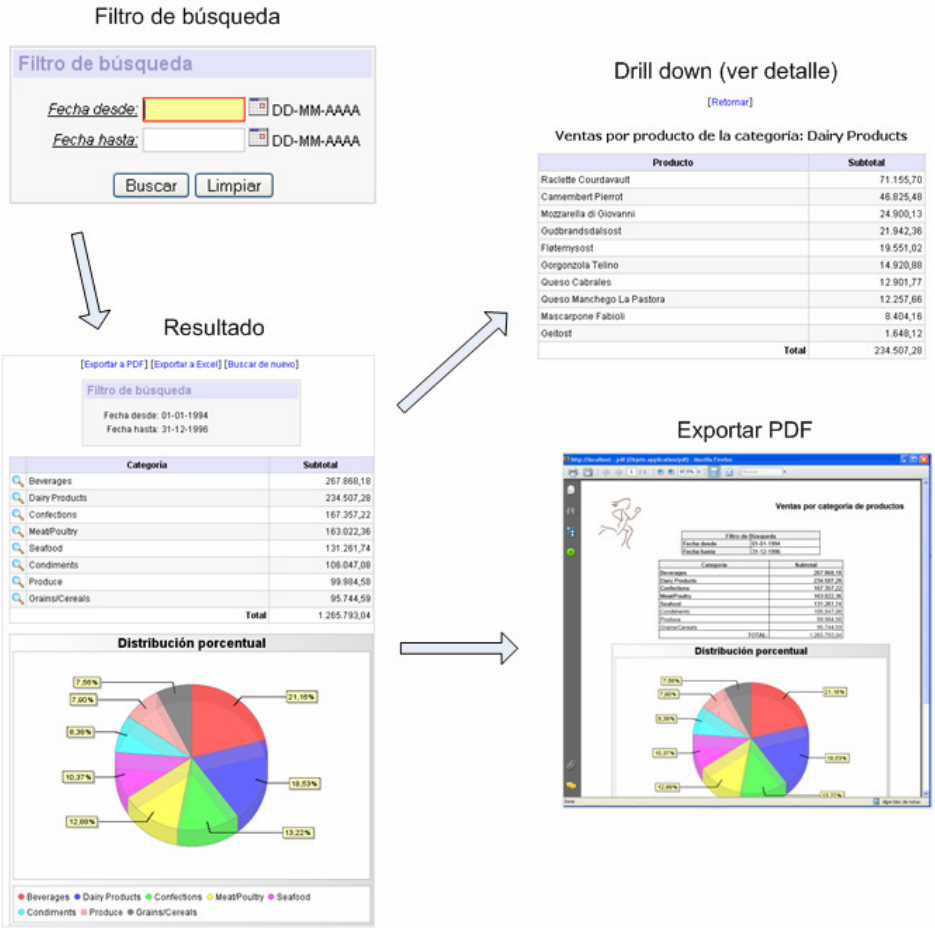
Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

El lado cliente – el mecanismo Ajax

Como se menciona al inicio del documento, Dinámica es un framework para la construcción de aplicaciones Ajax, sin embargo no hay nada específico en la infraestructura del lado servidor que se ha descrito hasta ahora que implique una orientación hacia la web 2.0, lo que realmente le añade esto al framework son las plantillas de soluciones, cuya porción cliente (páginas html, biblioteca de javascript) si hace uso intensivo de esta técnica, y la porción server del framework se acopla bien, ayudando en la generación de fragmentos html o javascript.

Normalmente un modulo web que resuelve un problema de negocios estará conformado por un conjunto de Actions, y uno de esos Actions (el Action de inicio o entrada) será el que carga la página principal. De allí en adelante son interacciones Ajax desde esta página con el resto de los Actions del modulo. Es decir, una solución contará con una sola página, el resto de los Actions solo retornarán fragmentos para ir cambiando dinámicamente, mediante Ajax, el estado de esta página de arranque.

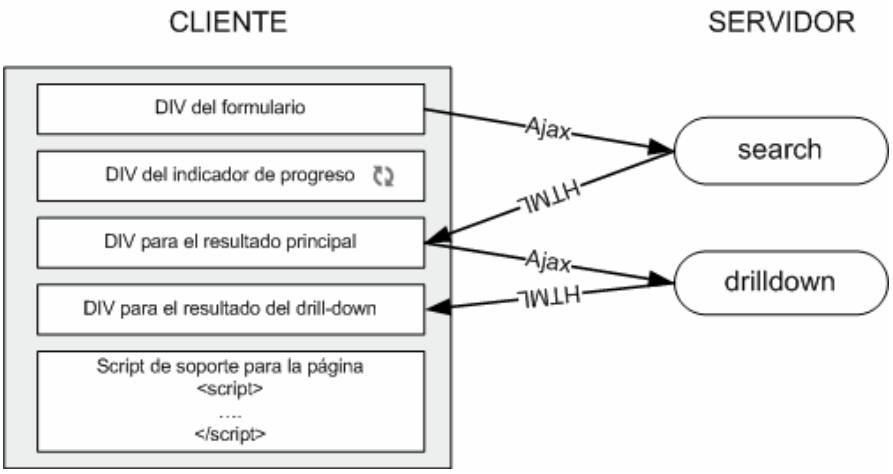


Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

La página principal carga la biblioteca de JavaScript de Dinámica, que es una capa simple y ligera de código para soportar las operaciones Ajax del lado cliente. El estilo Ajax de Dinámica se basa en retornar fragmentos de HTML o de JavaScript, no se retorna XML para evitar operaciones de parsing lentas del lado cliente.

Esta página de arranque contiene los DIVs donde irán apareciendo los resultados, así como todo el javascript específico requerido por esta plantilla.



Siempre que se invoca una acción vía Ajax, se esconde un DIV y se muestra el DIV con el indicador animado de progreso, lo que le da sensación de agilidad a la página. Cuando el resultado llega, se esconde el DIV de progreso y se muestra el DIV de resultado. Así la misma página va cambiando de estado, pero puede retornar al anterior de inmediato, simplemente escondiendo un DIV y mostrando otro, y esto tiene un impacto particularmente positivo para la validación de formularios, porque cuando se hace un POST del formulario con Ajax los valores no se pierden en caso de errores de validación porque nunca se navegó hacia otra página como seria el caso de un POST "clásico", retornar al formulario solo es cuestión mostrar el DIV con el formulario y su último contenido.

La biblioteca de JavaScript de Dinámica es servida por un Action que viene por defecto con cualquier webapp creada con el plugin de Dinámica para eclipse, el Action es: /action/script. La función más importante es "ajaxCall()" que encapsula la lógica de una llamada Ajax y simplifica el código requerido por las páginas HTML. Esta biblioteca es portable entre browsers. No tiene la pretensión de ser un framework JavaScript, de hecho puede coexistir con productos de este tipo como jQuery o ExtJS.

Producido por:

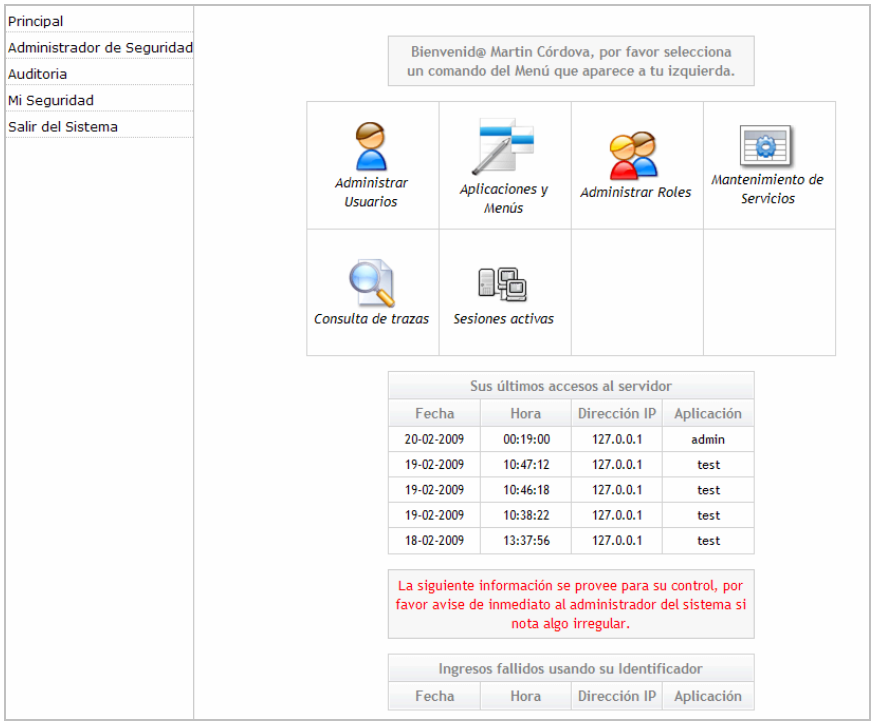
Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Facilidades del framework

Para que el programador conozca el potencial que tiene a mano con Dinámica, a continuación se enumeran algunas de las ventajas más relevantes que distinguen a este framework:

Dinámica provee todo lo necesario para construir e implantar aplicaciones de negocios, incluyendo:

- ✓ Sistema completo de seguridad integrada y configurable (de uso opcional), incluye una consola administrativa vía Web y los scripts DDL/DML para inicializar las tablas de seguridad en diversos dialectos SQL. Incluye integración LDAP y servicios de auditoria declarativa – sin programación.



- ✓ Manejo centralizado y configurable de los errores con notificaciones automáticas vía email cuando ocurre una excepción.

- ✓ Registro transparente de métricas de rendimiento a tres niveles de detalle (incluyendo JDBC), integradas dentro del framework y activadas mediante parámetros de configuración.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

- ✓ Una extensa colección de plantillas de soluciones prefabricadas, esto supone el nivel de reutilización más extremo, porque no estamos hablando de clases o componentes, sino de soluciones completas, plenamente documentadas y probadas, con calidad de producción.

	**ViewChart** Ejecutar consulta con un número fijo de parámetros, retorna un grid con totales y un gráfico, permite hacer drilldown desde el grid para ver detalle.		**PagedView** Técnica básica para mostrar un Recordset usando un grid paginado, permite hacer sort por cualquier columna, el sort se ejecuta en el servidor gracias a Recordset.
	**CrossTab** Resuelve el problema de presentar resultados de consultas SQL tipo CROSSTAB, incluye ejemplos de varias extensiones al framework.		**PickList** Muestra el uso de los picklists o listas de selección para reemplazar Combos. Contiene diversos ejemplos que atacan los casos más comunes.
	**EditForm** Muestra el manejo de los formularios con Ajax, editar, validar y actualizar una estructura de datos maestro/detalle.		**Filter** Ejecutar consultas mediante un formulario de filtro complejo que permite combinar varias condiciones, mostrar el resultado en un grid paginado.
	**CrudSimple** Módulo básico para manejo de tablas pequeñas (CRUD = Create, Read, Update, Delete). Incluye grid paginado y formulario con validación ajax.		**Crud** Combina las plantillas Filter y CrudSimple para proveer un modelo sofisticado de consulta y manejo de registros con Ajax.
	**Blob** Manejo completo de BLOBs (documentos e imágenes) con base de datos, incluye estilo Ajax para la carga (upload) de los archivos en la BD.		**MasterDetail** Muestra como generar tanto en HTML como en PDI un reporte Master/Detail sin escribir ni una sola línea de código (para la salida HTML).
	**ParentChild** Genera un reporte PDF con tres niveles de agrupamiento, subtotales y totales, a partir de un recordset jerarquico.		**Agenda** Muestra el uso de un Calendario empotrado en un IFRAME, y usando Ajax responde para traer un grid relacionado con la fecha seleccionada.
	**HGrid** Grids paginado con orientación horizontal, tipo "thumbnail" o mosaico, ideales para hacer galerías de imágenes o catálogos. ¡No requiere código!		**Calendario** Muestra como manejar y configurar el control de Calendario, para acotarlo con reglas de negocios.
	**CAPTCHA** Valida los dígitos ingresados vs los mostrados en una imagen, para mejorar un proceso de Login o afiliación y evitar robots.		**EditGrid** Grid Editable y Dinámico, permite ingresar varios registros en lote, pre-validados, usa controles avanzados en las celdas.
	**ImportExcel** Insertar en batch, registros leídos y validados correctamente de un archivo Excel. Incluye estilo Ajax para la carga (upload) de los archivos.		**LinkedGrids** Muestra dos grids paginados vinculados entre si. ¡No requiere código JAVA!
	**RegGen** Genera un reporte dinámico, donde el formulario de búsqueda permite seleccionar las columnas que aparecerán en la consulta HTML, PDF y EXCEL.		**Locator** Búsqueda de registro por criterio exacto, útil para localizar datos de un cliente o de una cuenta, con toda su información relacionada.
	**Rep8020** Permite ver dado unos rangos de montos totales de las facturas, como se distribuyen estas en esos rangos, tanto por cantidades de facturas como por montos.		**ConcatPDF** Permite crear un PDF a partir de los PDFs generados por otros Actions locales, los concatena en memoria para producir un solo documento que los combina a todos.

- ✓ Protección automática contra ataques de inyección de SQL.

- ✓ Transacciones de base de datos (JDBC) controladas de manera transparente y declarativa. Modelo automático de persistencia de datos de alto rendimiento.

Producido por:

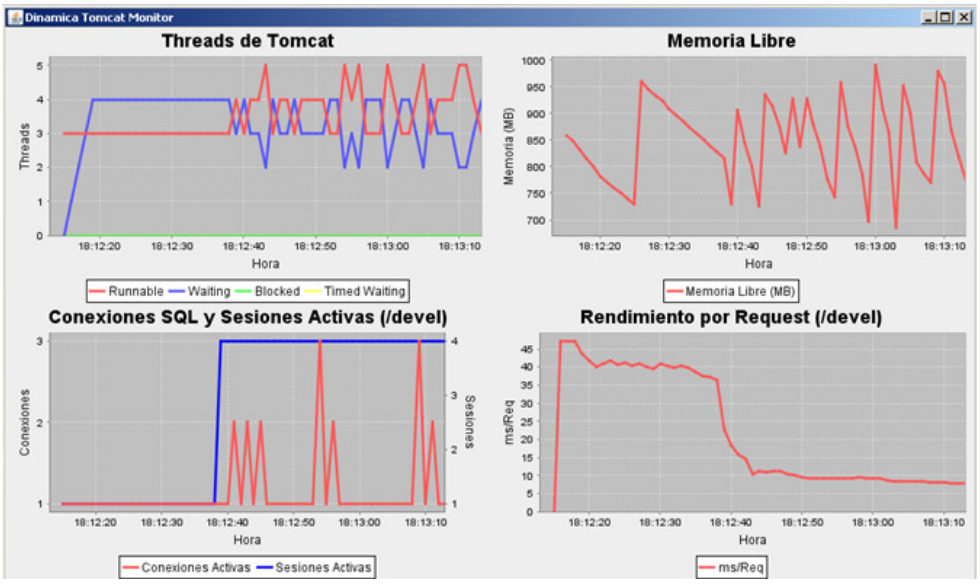
Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

- ✓ Soporte para manejo de documentos e imágenes (BLOBs) en base de datos sin necesidad de escribir código, probado con Oracle, SQLServer y PostgreSQL.

- ✓ Facilidades integradas para generar eficientemente del lado servidor gráficos de negocios y reportes PDF o Excel, utilizando componentes de código abierto de alta calidad.

- ✓ Procesamiento automático-declarativo de formularios.

- ✓ Servicio de monitoreo integral del proceso servidor (Tomcat) utilizando instrumentación JMX simplificada por Dinámica. Con un simple URL se provee valiosa información en tiempo real para diagnosticar el funcionamiento del servicio. Incluye envíos automáticos por email, generación de reportes PDF o monitoreo online con aplicación GUI portable llamada el "Monitor":



Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Implicaciones técnicas de Dinámica

Hay algunos aspectos técnicos relativos al uso del lenguaje Java, que favorecen a las aplicaciones construidas con este framework y es importante documentarlas y conocerlas:

Herencia vs. Reflexión

Dinámica no utiliza reflexión, sino herencia. Los tipos de variables son chequeados en tiempo de compilación, y las invocaciones a métodos son resueltas rápidamente, no hay chequeos dinámicos para ver si una clase implementa un método. La herencia es más rápida que la reflexión, es un mecanismo nativo al lenguaje y que favorece la compilación a código nativo y la ejecución optimizada por la JVM (máquina virtual de Java) ya que hace más predecible al código. La extensión del framework por parte de los programadores se hace vía herencia, extendiendo las clases básicas del framework. El polimorfismo basado en herencia e interfaces es un aspecto básico y estructural del diseño de Dinámica.

Creación eficiente de objetos

Una aplicación basada en Dinámica usa en la mayoría de sus Actions las mismas clases: GenericTransaction, GenericOutput, etc. Esto tiene una implicación muy importante con máquinas virtuales modernas, como el Java 6 de Sun, ya que el cache de clases se ha hecho muy eficiente a nivel de la JVM, y la creación de objetos de clases usadas frecuentemente se ve favorecida por estos mecanismos, es de hecho en extremo rápida. La consecuencia directa es que el circuito o mecanismo que se activa cuando se recibe y sirve un request o solicitud de un cliente HTTP, es resuelto rápidamente, crear la clase Transaction, crear la clase Output, generar la salida con un TemplateEngine, etc. Desde que una aplicación atiende los primeros requests, estas clases entraron en acción, ya la JVM las tiene "cacheadas". Desde hace tiempo que no se recomienda escribir pools de objetos hechos a la medida, sobre todo de objetos tipo Bean, que no tienen un tiempo de creación particularmente alto, porque es más ineficiente que dejar a la JVM hacer su trabajo. Y eso es una de las directrices de diseño de Dinámica, no tratar de sustituir el trabajo de la JVM, la dejamos hacer bien su trabajo.

Diseño que favorece la extensión y el mantenimiento

El diseño de las clases básicas de Dinámica buscó posibilitar la personalización de componentes de manera simple, a menudo todo lo que se requiere es sobre-escribir un método sencillo, como en este ejemplo, que crea una versión especial del plugin de gráficos de barras, pero con las etiquetas rotadas en 90 grados:

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

```
/**
 * Variante del plugin estandar de grafico de barras 3D verticales,
 * esta version permite etiquetas rotadas verticalmente.<br>
 * <br><br>
 * Actualizado: 2007-05-31<br>
 * Framework Dinamica - Distribuido bajo licencia LGPL<br>
 * @author mcordova (martin.cordova@gmail.com)
 */
public class VerticalLabelBarChart extends dinamica.charts.VerticalBarChart3D
{

    /**
     * Implementa este metodo para cambiar el comportamiento defacto
     * de las etiquetas horizontales y rotarlas 90 grados
     */
      public void configurePlot(Plot plot)
    {
            super.configurePlot(plot);
            CategoryPlot p = (CategoryPlot)plot;
            CategoryAxis axis = (CategoryAxis)p.getDomainAxis();
            axis.setCategoryLabelPositions(CategoryLabelPositions.UP_90);
    }

}
```

En otros casos, cuando hemos tenido mas tiempo para pensar mejor un diseño, lo que hacemos es proveer una clase genérica parametrizable vía configuración (config.xml), y de esa manera ni siquiera hay que escribir código, como es el caso de los reportes Excel sencillos, para los cuales basta añadir una configuración al archivo config.xml, y últimamente también lo hicimos para los reportes PDF de tablas simples, manejadas a través del template CRUD simple.

Producido por:

Martín Córdova y Asociados C.A.
www.martincordova.com
martin.cordova@gmail.com

Entonación básica del servidor (Tomcat 6)

Aproveche la compresión GZIP con Tomcat 6

No dejen de configurar la compresión de las salidas de texto generadas por Dinámica, solo tienen que activarlo en el conector HTTP o HTTPS, en el archivo /conf/server.xml de Tomcat, asi:

<compression="on"
compressableMimeType="text/html,text/xml,text/plain,text/javascript"/>

Si aunado al hecho de que las respuestas que genera Dinámica muchas veces son solo fragmentos de texto en vez de páginas completas (porque usa Ajax), también comprimimos la salida, entonces el cliente percibirá un tiempo de respuesta mucho más rápido, los paquetes que viajarán entre Tomcat y los browsers serán mucho más pequeños, y por lo tanto llegarán más rápido las respuestas a los browsers. La tasa compresión GZIP de texto es muy alta, cerca del 90%.

Aproveche el caché de recursos de Tomcat

Dinámica está todo el tiempo cargando plantillas y archivos XML que residen dentro de la aplicación, y Tomcat puede proveer un caché para estos recursos, optimizando el tiempo de carga. Solo deben editar el archivo /conf/context.xml y modificar el atributo "cachingAllowed".

<Context reloadable="true" cachingAllowed="true">

Esto es conveniente en producción, pero tiene el efecto adverso de que si actualiza recursos no verá los cambios sino cuando expire el caché.

Entonación de la JVM

Es conveniente complementar su servidor de servlets con una buena configuración para que Java opere rápido y estable por muchos días:

-server -Xms1024M -Xmx1024M -XX:+AggressiveOpts

El parámetro "-XX:+AggressiveOpts" indica que el compilador a código nativo use las últimas optimizaciones que han sido incorporadas en la versión de Java que esté utilizando. No es fundamental, si considera que podría ser la fuente de inestabilidad, quítelo. Darle suficiente memoria a la JVM si es fundamental. No ponga el parámetro –Xmx a mas del 50% de la memoria física del equipo.