ZCode

Juan Delgado Zárate
http://zarate.tv/proyectos/zcode

Introducción

Después de crear un par de frameworks para las empresas en las que he trabajado, decidí que ya era hora de hacer uno propio. ZCode es la idea básica sobre la que trabajé para los talleres Subflash 2007.

ZCode tiene una finalidad prácticamente educativa. No está pensado para ofrecer millones de funcionalidades si no para mostrar la potencia de compartir código entre nuestras aplicaciones. Échale un ojo, coje lo que te interese y deshecha lo que no. La idea es que crees tu propio framework.

Qué es un framework

Un framework es un conjunto de clases para facilitar el desarrollo de aplicaciones. Ruby on Rails es uno de los más conocidos últimamente. Hay unos cuantos frameworks para Flash: ARP, Vegas, ActionStep, AsWing, GAIA, etc.

Por qué necesitamos un framework

Un framework nos ayuda (o debería) a programar más rápido y mejor. Aquí un listado de los beneficios que nos debería aportar:

- Evitar duplicar código, resolver los problemas una sola vez. Básicamente Don't Repeat Yourself.
- Coherencia entre distintas aplicaciones. Se comportan de una forma predecible.
- Una vez superada la curva de aprendizaje, facilidad para crear rápidamente nuevas aplicaciones invirtiendo el tiempo en la parte importante, no en tareas de bajo nivel .
- Al compartir todas nuestras aplicaciones un código común, implementar una mejora para todas es mucho más sencillo y viable.
- Otra ventaja de compartir código entre aplicaciones es que el código se expone mucho más a situaciones no previstas, a la vida real . A medida que más se usa, el número de bugs por descubrir disminuye. El número de usuarios se multiplica por el número de distintas aplicaciones que lo usen.

Crear un framework propio puede ser considerado como reinventar la rueda, pero yo creo que tiene sus ventajas:

- Hace exactamente lo que necesitamos, ni más ni menos.
- Al ser propio, la implementación de mejoras o la corrección de bugs es mucho más sencilla y viable.

ZCode xplained

ZCode consiste en 3 objetos básicos o estructurales mas una serie de clases adyacentes . La diferencia es que los objetos estructurales te obligan a que tus aplicaciones tengan una cierta estructura. Las clases adyacentes (FlashVars, RightClick, etc) se encargan de tareas específicas y se pueden usar sin mayores problemas en cualquier otro proyecto fuera de ZCode.

Estructurales

Son las clases Model, View y Config.

El modelo debería encargarse de mantener el estado de la aplicación. Cosas como página actual , elemento seleccionado o el acceso a datos yo las hago a través del modelo y algunas clases auxiliares. Es el cerebro , decide qué hacer en cada momento.

La vista se encarga de pintar lo que el modelo le diga. Sólo toma decisiones relacionadas con la parte gráfica. Yo, **en una decisión personal con la que mucha gente puede no estar de acuerdo**, normalmente prescindo de una clase controlador para capturar los eventos del usuario. Por ejemplo, si tengo una pequeña clase para mostrar avisos (como el típico ¿Quiere usted borrar este usuario Sí/No?) hago que esa clase sea listener del objeto Key para que capture ENTER (y ejecute aceptar) y ESC (cancelar el diálogo). Dicho esto y, dependiendo de la envergadura de la aplicación, hay veces que sí separo el controlador a una clase como propia.

El objeto de configuración se encarga de leer el xml de la aplicación, parsearlo y exponer sus valores a quien lo necesite. ¿Qué es exponer los valores del xml? Supongamos este xml básico:

```
<?xml version=”1.0” ?>
<data>
      <user age=”15”>
</data>
```

Ahora supongamos que necesitamos acceder al valor de la variable age en 10 clases distintas de la aplicación. Si en esos 10 sitios accedemos directamente al xml así:

```
var age:Number = Number(dataXML.firstChild.childNodes[0].attributes[“age”]);
```

Cualquier cambio en la estructura del xml nos obligaría a ajustar nuestro código 10 veces. Para evitar lo podemos crear en el objeto de configuración una variable pública age y acceder desde el código a esa variable y no al xml directamente. El mismo cambio de estructura sería mucho menos problemático porque sólo habría que cambiar el acceso al xml en el objeto de configuración. En pseudo código:

```
class Config{

      public var age:Number;
      .....
      age = Number(dataXML.firstChild.childNodes[0].attributes[“age”]);

}
```

```
class Wadus{

      ....
      var age:Number = config.age;

}
```

Aún así, por motivos de flexibilidad, siempre dejo como pública una variable en el objeto de configuración con el xml de los datos.

Clases adyacentes

Las clases adyacentes se pueden utilizar sin utilizar ZCode. La mayoría de ellas se encuentran dentro del package Utils . A día de hoy son automáticamente incluidas con ZCode las siguientes:

- Delegate
- FlashVars
- RightClick
- Trace
- MovieclipUtils

Es conveniente que les eches un ojo para ver qué métodos ofrecen. Cuanto más las uses, mejor ya que igualmente se van a compilar en el swf.

Diseño líquido

Si la aplicación es la principal (no ha sido cargada por ninguna otra, es decir, su línea de tiempo es _root), la vista automáticamente se hace listener de Stage para reaccionar al evento onResize.

En el caso de que no lo sea, la aplicación tiene que esperar a que la aplicación que la contiene llame a su método setSize. De esta forma son las aplicaciones o clases superiores en la jerarquía las que deciden qué tamaño ofrecen a las aplicaciones que contienen. Por ejemplo, imaginemos un player de vídeo. Si está sólo en la pantalla, probablemente quiera/necesite ocuparla completamente. Sin embargo, si el mismo player de vídeo es incluído por otra aplicación, tiene que ser la aplicación que lo contiene la que decida qué parte de la pantalla le adjudica.

Relacionado con esto, la vista tiene 2 métodos importantes. initialLayout es llamado una vez por aplicación. En este método debes crear los objetos pero NO posicionarlos. El posicionamiento de los elementos hay que hacerlo en el método layout ya que es automáticamente llamado cada vez que se ejecuta setSize.

Acceso a datos

La forma de acceder a los datos normalmente varía o escala con la complejidad de la aplicación. En cualquier caso evita usar objetos anónimos y crea clases para tus entidades . ¿Qué son entidades? Supongamos una aplicación para la gestión de coches. Tenemos el siguiente xml:

```
<?xml version=”1.0” ?>
<data>
      <car puertas=”3” price=”20000” />
      < car puertas=”5” price=”45000” />
      < car puertas=”5” price=”13000” />
</data>
```

Una entidad sería una clase Car con las propiedades que vayas a utilizar:

```
class Car{
```

```
      var doors:Number = 0;
      var price:Number = 0;
      ....

      public function setXML(carNode:XMLNode):Void{
            doors = Number(carNode.attributes["doors"]);
            price = Number(carNode.attributes["price"]);
      }

}
```

Al parsear el xml, haz lo siguiente:

....

```
var totalCars:Number = dataXML.childNodes.length;
for(var x:Number=0;x<totalCars,x++){

      var car:Car = new Car();
      car.setXML(dataXML.childNodes[x]);

}
```

De esta forma, el parseo del xml se hace dentro del propio objeto (encapsulación) y desde otras partes de la aplicación accederemos a las propiedades de la clase Car ganando validación de tipos y evitando errores sintácticos.

A mi personalmente me gusta tener una clase manager por cada entidad. En este caso, CarManager. Las funciones de la clase manager serían la gestión de sus entidades. Por ejemplo:

```
class CarManager{
      .....
      public function parseDataXML(carsNode:XMLNode):Void{} // el bucle anterior
      public function getCars():Array{} // array con todos los coches
      public function getCarsByPrice():Array{} // ordenados por precio
      public function removeCar(carID:String):Boolean{} // eliminar por id
}
```

Links

- Otros framworks en Flash
  - ARP (Ariaware RIA Platform): http://www.ariaware.com/products/arp/
  - Vegas, http://code.google.com/p/vegas/
  - ActionStep,
- El patrón Singleton en la Wikipedia: http://en.wikipedia.org/wiki/Singleton_pattern
-