# Ext JS 4

## Ext JS 4 与之前版本的比较

### **渲染效率提高**

所有类都经过调优，包括最影响渲染效率的布局引擎重写。

### **命名空间**

命名空间是 Ext JS 4 的 MVC 的基础，自此 Ext JS 类能按作用域分开存放了。.NET 或 Java 开发者应该熟悉命名空间带来的好处：命名空间让全类名映射到类文件路径变得很容易，将类按作用域分文件夹存放使得类更容易管理。以 MVC 为例，Ext JS 类将按作用域：模型、视图和控制器分为三类，分别存放于对应文件夹中。

### **按需加载类**

这个特性其实是基于全新设计的类系统的，详见下面的小结。不同于先前版本：即使用到 Ext JS 框架中很少一部分单元，Ext JS 也会加载所有的框架，按需加载只加载需要的类。因此按需加载类为 JS 优化和减少内存消耗提供了一个有效途径。Sencha 为此还提供了 SDK 工具对 JS 代码进行 Minify，在部署前运行 minify 对 JS 代码最小化后，将得到一个最小 JS 集合。

### **MVC 架构**

用 Ext JS 4 之前的版本写大的客户端应用，您会发现越来越“难”，您会发现有四难：难写、难读、难维护、难扩展。随着越来越多的功能添加进来，代码越来越失控，一个 JS 文件几千行可能很普遍了，当然也不排除代码组织得很好易于扩展的情况，但这些都需要开发者付出额外的开发代价去组织自己的架构。从 Ext JS 4 开始有了自己的 MVC 架构，开发者不必再付出这种额外的代价也能写出漂亮的代码。Ext JS 4 对 MVC 有自己的定义，以下定义来自 Sencha 官网的文档：

#### **Model**

：一组字段的集合以及它们对应的数据（例如：“User”类 model 有“username”和“password”字段），通过 data 包 (store,proxy 等 )Model 能序列化自己，并能通过关联关系从一个 Model 导航到另一个 Model。Model 的工作原理类似 Ext JS 3 中的 Record 类，通常结合 Store 为表格控件或其它控件提供显示数据。

#### **View**

任意组件，如 Grid, Tree 和 Panel 都是视图。

#### **Controllers**

在这里写所有的逻辑代码：如渲染视图、实例化模型、加载并初始化其它控制器

### MVC 的概念

MVC 是一种成熟的软件设计模式。MVC 模式的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外，此模式通过对复杂度的简化，使程 序结构更加直观。软件系统通过对自身基本部份分离的同时也赋予了各个基本部分应有的功能。专业人员可以通过自身的专长分组：

### MVC 模式的特点：

#### **重用性**

Ext JS 4 中，数据模型、视图组件和存储组件能重复使用，使用时只需将它们的名称作为引用添加到特定的 controller 中。

#### **职能分离**

模型、视图、控制器以及存储之间的职能分离，使得每个 JS 文件的职能单一化、最小化。开发人员只需要引用这些必需的职能单元即可构建新的功能。

#### **职责清晰**

由于每一个 JS 单元文件的职责清晰，不同类型的“职责”被划分为不同的组件。在集成开发工具中，开发人员很容易利用这些组件构建自己的应用。

#### **复杂性**

运用 MVC 模式时有一定的复杂性，因为开发者需要付出额外的努力去学习 Ext JS 4 的 MVC 框架，但在大型项目中这个付出是值得的。另外 Sencha 提供了相应的集成开发工具 Sencha Architect 协助基于 MVC 框架的开发，一定程度上减轻了因运用 MVC 模式带给开发者的压力。

### 运用 MVC 以后

#### **代码层次结构清晰**

* JavaScript 脚本都按职责存放在不同的 JS 文件中。
* 不同的 JS 文件按 MVC 目录命名约定"controller"、"store"、"view"、"model"分类存放。
* 整个工程里只有一个 HTML 文件，而且只作为入口文件，里面不写任何脚本。

#### **整体逻辑更加条理**

入口定义在 Ext.application({...}) 中，这是 Ext JS 4 为开发 MVC 应用新增的函数，我们可以在里面引入初始时所需的 controller、view 或 model，在 launch 函数中编写初始代码，和以前在 Ext.onReady() 中一样。因为 Ext 引擎会在 Ext.application({...}) 函数执行时创建一个全局 application 实例并能在 controller 中被引用得到 (this.application.\*)，所以我们也可以在 application 中定义一些公共函数，甚至注册一些事件，方便其它 controller 调用。

"model"文件夹中只有模型相关的类定义，每个模型定义必须包含字段，还能定义字段校验规则，不同模型之间的关联，以及数据代理 ( 连接服务器存取数据 )。比如雇员\部门对应两类模型，体现为两个类文件"Employee"和"Department"。

"view"文件夹中定义了所有的 widget，每一个 widget 对应一个类文件。view 的代码属于静态代码，后面将提到怎样用工具自动生成。

"controller"文件夹中的控制器按管理范围的不同划分为不同的类文件，其中每一个控制逻辑都包括初始化、组件事件监听以及引 用等。比如，导航菜单的控制器会包含对主页面菜单按钮的动作监听”click“，并在该事件中负责创建相应的子页面；而子页面对应的控制器只负责该页面中 的组件的行为，如提交按钮的点击等。另外控制器本身能被动态加载，在下文的例子中我们还可以看到，不同的控制器被加载的时机和顺序是由用户行为（如点击某 按钮）控制的。其实这个特性是基于依赖于 Ext JS 4 的动态加载类的新特性的。详见"按需加载类"一节。

由此可见，运用 MVC 后 HTML 里不再直接写 JS 了，JS 按逻辑、职责分门别类存于不同的目录，对应到不同的文件中。

## 示例和代码

### 一个非常实用的集表单提交，表格应用和图表显示的综合运用的例子

#### **入口代码**

入口代码的功能与 Ext.onReady() 是一样的，不同在于，新版的 Ext JS 4 将命名空间和全局 application 变量自然地融合在入口函数中，开发者很容易从 Model、View 或 Controller 中调用 application 中的函数、变量和事件，并从使用命名空间的过程中获益，使得 MVC 的开发变得更方便。

本示例使用了默认的 appFolder:app, 实际中用户可以覆盖此属性，使用符合项目要求的路径名。我在项目中倾向于 appFolder 中的所有类由 Sencha Architect 工具维护，通过配置 mvn，在 compile 时将 appFolder 中生成的类拷贝到 webApp 中。手工维护的 JS 文件放在独立的命名空间中（称其为扩展空间吧），并在入口中声明，这样能被 application 引用并加载，同时在扩展空间的类也能 require 到 application 对应命名空间中的类，这样做的好处是，您能将 override 的代码移出来放到扩展空间中，另外还能放一些项目中用到的插件。

#### 代码结构

 主菜单控制器

控制主菜单中三个按钮的点击事件，以及它们的状态：如按下和浮起。

 表单子页面控制器

表单提交相关，本示例中，点击提交按钮后将弹出一个“Save”提示框。

 表格子页面控制器

通过查询按钮控制表格的 store 重新加载新数据。

 图表子页面控制器

通过下拉选择框控制图表的刷新