Главная › Блоги › Lertion's блог

[Обучение Ext JS] :: Часть 13. Повторное использование кода: Расширяем Ext JS

Мар 19 2010

Хорошо, что мы будем расширять?
Типично, мы будем создавать приложение и поймаем себя на том что многократно пишем один и тот же код. Или же, нам повезет и мы, заметив что нам придется часто повторятся, выберем лучший кусок кода и двинемся дальше. Обычно в этом и заключается вопрос, какой компонент **Ext** стоит расширять.
Даайте вспомним прошлый случай, систему управления взаимоотношениями с клиентами и партнёрами. Мы знаем что у нас в приложении есть несколько объектов Person. Возможно нам понадобится что-нибудь для отображения контактной информации человека.
Существует несколько компонентов для отображения требуемой нам информации, и нам нудно выбрать правильный. Объект таблицы сработает, но табличный вывод стольких значений может затормозить наше приложение, и в таком случае нам придется просматривать данные только одного человека за раз. PropertyGrid немного отличается от стандартной таблицы, но нам надо вывести только имя и адрес. Нам не надо чтобы пользователь видел всю возможную информацию. По той де причине что и таблицу можно убрать DataView. Нам действительно нужна только одна запись за один отрезок времени.
Это подводит нас к форме или панели. Формы подразумевают редактирование информации, таким образом для простого отображения данных нам нужна панель. Объекты Panel очень гибкие и являются ядром большинства других объектов **Ext JS**. Телом объекта Window является панель. Различные части Accordian это объекты Panel. Тела закладок в TabPanel тоже панели. Создание собственного компонента, который расширяет объект Panel открывает двери ко многим различным областям в приложении **Ext JS**.

Создание собственных пространств имен
Мы хотим создать соответствующие компоненты с их собственными пространствами имен для инкапсуляции, быстрых ссылок для того чтобы организовать код. Может случиться так, что наши пакеты или классы будут назваться так же как те классы и пакеты, которые уже присутствуют в **Ext JS**. собственное пространство имени препятствует конфликтам происходящим между классами с одним и тем же названием, на протяжении того времени как они определены в пределах собственного отдельного пространства имени. Общепринято при названии для пользовательских объектов использовать пространство имени Ext.ux.

```
Ext.namespace('Ext.ux');
```

Поскольку мы собираемся создавать коллекцию панелей отображения для нашего приложения, мы отделим наше пространство имен от остальных.

```
Ext.namespace('CRM.panels');
```

Нам нужно разместить эту строку поверх каждого шаблона определения класса.

Наш первый собственный класс
Сначала, давайте рассмотрим довольно простой скрипт, который берет одну запись и выкладывает ее на панели, на экране.

Example 1: ch13ex1.js

Исходный код примера:

Исходный код примера:

```
  var userData = [
  {ID:1,FIRSTNAME:'John',LASTNAME:'Lennon',EMAIL:'john@beatles.com',
PASSWORD:'apple1',ADDRESSTYPE:'Home (Mailing)',STREET1:'117 Abbey Road
',STREET2:'',STREET3:'',CITY:'New York',STATE:'NY',ZIP:'12345',PHONETY
PE:'Cell',PHONE:'123-456-7890'},
  {ID:2,FIRSTNAME:'Paul',LASTNAME:'McCartney',
EMAIL:'paul@beatles.com',PASSWORD:'linda',ADDRESSTYPE:'Work
(Mailing)',STREET1:'108 Penny Lane',STREET2:'',STREET3:'',
CITY:'Los Angeles',STATE:'CA',ZIP:'67890',PHONETYPE:'Home',
PHONE:'456-789-0123'},
  {ID:3,FIRSTNAME:'George',LASTNAME:'Harrison',
EMAIL:'george@beatles.com',PASSWORD:'timebandit',
ADDRESSTYPE:'Home (Shipping)',STREET1:'302 Space
Way',STREET2:'',STREET3:'',CITY:'Billings',STATE:'MT',
ZIP:'98765',PHONETYPE:'Office',PHONE:'890-123-4567'},
  {ID:4,FIRSTNAME:'Ringo',LASTNAME:'Starr',
EMAIL:'bignose@beatles.com',PASSWORD:'barbie',
ADDRESSTYPE:'Home (Mailing)',STREET1:'789 Zildizhan Pl',
STREET2:'',STREET3:'',CITY:'Malibu',
STATE:'CA',ZIP:'43210',PHONETYPE:'Home',PHONE:'567-890-1234'}
  ];
   var userDetail = new Ext.Panel({
     applyTo: 'chap13_ex01',
     width: 350,
     height: 250,
     title: 'Chapter 13 Example 1',
     data: userData[0],
     tpl: new Ext.XTemplate([
       '<img src="/resources/images/s.gif" width="21" height="16"
                    /><b>{FIRSTNAME} {LASTNAME}</b><br />',
        '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="emailEdit_{ID}" class="iconLnk" align="Edit
Email Address" border="0" />{EMAIL}<br />',
         '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="phoneEdit_{ID}" class="iconLnk" align="Edit
Phone" border="0" />{PHONE} ({PHONETYPE})<br />',
       '<b>{ADDRESSTYPE} Address</b><br />',
       '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="addrEdit_{ID}" class="iconLnk" align="Edit
Address" border="0" />{STREET1}<br />',
       '<tpl if="STREET2.length &gt; 0">',
       '<img src="/resources/images/s.gif" width="21" height="16"
/>{STREET2}<br />',
       '</tpl>',
       '<tpl if="STREET3.length &gt; 0">',
         '<img src="/resources/images/s.gif" width="21" height="16"
                                        />{STREET3}<br />',
       '</tpl>',
       '<img src="/resources/images/s.gif" width="21" height="16"
                                     />{CITY}, {STATE} {ZIP}'
     ]),
     listeners:{
       render:{
           fn: function(el){
             this.tpl.overwrite(this.body,this.data);
           }
       }
     }
   });
```

У нас есть простой массив объектов данных и определение Panel. Мы отправляем один предмет данных в настройки панели, определяя XTemplate для отображения записи и применяем слушатель который в свою очередь применит данные к XTemplate когда закончится рендеринг панели.

Chapter 13 Example 1

John Lennon
john@beatles.com
123-456-7890 (Cell)
Home (Mailing) Address
117 Abbey Road
New York, NY 12345

Превратить это в многократно используемый компонент очень просто, так как большая часть нашего кода просто переедет в наше определение собственного класса. Сначала мы создадим новый шаблон класса, ContactDetails.js, и определим его начальное классовое объявление как расширение класса Ext.Panel. **Ext JS** предоставляет методы для расширения компонентов.

Example 2: ContactDetails.js

```
Ext.namespace('CRM.panels');
CRM.panels.ContactDetails = Ext.extend(Ext.Panel,{
  // The panel definition goes here
});
```

Нашим следующим шагом будет начать определять свойств и методы нашего компонента. Мы начнем со свойств по умолчанию, которые будут характерны для него. Некоторые из этих свойств могут быть замещены в настройках объекта, но эти умолчания позволят нам только отправить то, что мы хотим изменить в нашем индивидуальном приложении.

Исходный код примера:

Исходный код примера:

```
width: 350,
height: 250,
data: {
  ID: 0,
  FIRSTNAME: '',
  LASTNAME: '',
  EMAIL: '',
  ADDRESSTYPE: 'Home (mailing)',
  STREET1: '',
  STREET2: '',
  STREET3: '',
  CITY: '',
  STATE: '',
  ZIP: '',
  PHONETYPE: 'Home',
  PHONE: ''
},
tpl: new Ext.XTemplate([
   '<img src="/resources/images/s.gif" width="21" height="16"
                  /><b>{FIRSTNAME} {LASTNAME}</b><br />',
   '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="emailEdit_{ID}" class="iconLnk" align="Edit
Email Address" border="0" />{EMAIL}<br />',
   '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="phoneEdit_{ID}" class="iconLnk" align="Edit
Phone" border="0" />{PHONE} ({PHONETYPE})<br />',
  '<b>{ADDRESSTYPE} Address</b><br />',
  '<img src="/resources/images/icons/silk/database_edit.gif"
width="16" height="16" id="addrEdit_{ID}" class="iconLnk" align="Edit
Address" border="0" />{STREET1}<br />',
  '<tpl if="STREET2.length &gt; 0">',
    '<img src="/resources/images/s.gif" width="21" height="16"
                                  />{STREET2}<br />',
  '</tpl>',
  '<tpl if="STREET3.length &gt; 0">',
    '<img src="/resources/images/s.gif" width="21" height="16"
                                  />{STREET3}<br />',
  '</tpl>',
  '<img src="/resources/images/s.gif" width="21" height="16"
                              />{CITY}, {STATE} {ZIP}'
 ]),
```

ВАЖНО: *Привал! Ничего не кажется знакомым? Да, должно бы, особенно после внимательного изучения 'записей' в userData. Класс это объект, как и каждая 'запись' в массиве userData. На самом базовом уровне, объект это набор пар имя-значение. Настоящее различие заключается в том, что у клсс может работать как значение имени атрибута объекта, а объект может ссылаться на свой класс (название класса). Каждая из пар имя-значение образует конструктор класса ContactDetails.*

Методы замещения
Продолжая создавать наш собственный класс мы подошли к методам замещения. как уже упоминалось ранее, мы можем заместить метод нашего родительского класса определяя метод с тем же названием в дочернем классе. Ключевым компонентом класса Panel является метод initComponent(), который (как нам подсказывает его название) запускает компонент Panel. Большинство методов никогда не потребуется замещать, но иногда нам может потребоваться заместить какой либо определенный метод для того чтобы добавить или изменить поведение компонента.

```
initComponent: function(){
  CRM.panels.ContactDetails.superclass.initComponent.call(this);
    if (typeof this.tpl === 'string') {
    this.tpl = new Ext.XTemplate(this.tpl);
  }
},
```

ВАЖНО: *Ext JS использует суперкласс как программную ссылку на родительский объект объекта класса. Например CRM.panels.ContactDetails является подклассом родительской (суперкласс!) Ext.Panel.*

Теперь нам надо найти способ применить новый XTemplate и передат ьего в наш класс ContactDetails. Поскольку разработчик может отправить шаблон настройки через аргумент tpl, вместо объекта XTemplate, важно утвердить такой ввод и соответственно настроить. Нашим первым действием станет вызов метода initComponent() из родительского класса Panel и затем, настройка значения аргумента tpl.

Понимание порядка событий
В предыдущем примере (Example 1), мы использовали событие render применяя наш XTemplate в объект Panel. В нашем собственном (пользовательском) компоненте мы сделаем точно так же, написав метод замещения для метода onRender().

Исходный код примера:

Исходный код примера:

```
onRender: function(ct, position) {
  CRM.panels.ContactDetails.superclass.onRender.call
                          (this, ct, position);
  if (this.data) {
    this.update(this.data);
  }
},
```

Здесь мы вызываем метод onRender() нашего суперкласса Panel. После чего мы подтверждаем, то у нам есть значение для атрибутов данных прежде чем вызывать собственный метод update() нашего компонента.

```
update: function(data) {
  this.data = data;
  this.tpl.overwrite(this.body, this.data);
}
```

Метод update() берет аргумент данных (запись из массива), применяет этот аргумент к атрибуту компонента данных и затем применяет XTemplate компонента к "телу" компонента. Это метод компонента, а не замещенный метод родительского класса. Для нас важно то, что необходимо для этого метода.

Уогда мы можем сделать что?
Наш XTemplate не применяется к компоненту немедленно, так как он не может заместить тело Panel до тех пор пока Panel не будет отрендерена. Вот что мы имели в виду, говоря "порядок событий", и сначала может показаться сложным, но только пока вы не постараетесь разобраться. Например, представьте, что мы пытались динамично применть собственный аргумент ID как чать нашего конструктора:

```
id: 'ContactDetails_' + this.data.ID,
```

Это бы сломало наш компонент. Почему? Потому что this.data также предмет в конструкторе и не сущестыует до тех пор, пока компонент не будет инициализирован в памяти. То же самое может случиться если попытаться применить наш XTemplate в методе initComponent():

```
initComponent: function(){
  CRM.panels.ContactDetails.superclass.initComponent.call(this);
  this.tpl.overwrite(this.body, this.data);
},
```

Все развалится когда initComponent() начнет создание компонента. Компонент на этом этапе не является частью **DOM** браузера, так что еще не существует "тела" для применения XTemplate. понять принцип работы "порядка событий" одна из самых больших трудностей в обучении **Ext JS**, если вы ничего об этом не знаете и порядок событий отличается в каждом из классов.

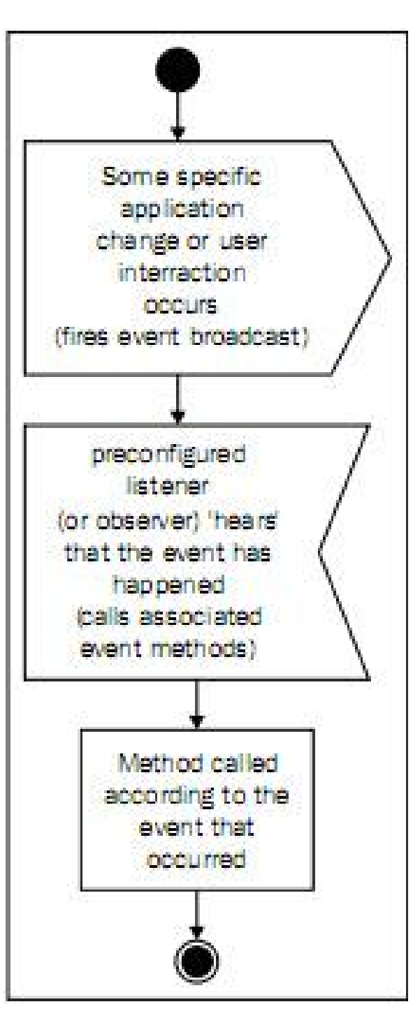
Что за приложение управляемое событиями?
Вот теперь мы подошли к сути. Одно огромное препятствие, которое надо преодолеть при переходе от процедурного стиля программирования к объектно-ориентированной разработке и это модель приложений управляемых событиями. не все OO-программы управляются событиями, но все явно к тому идет. И **Ext JS** не исключение. Изначально поток приложения определяется по ощущениям изменений состояния, или действий пользователя вызвавшего событие. Когда происходит событие, приложение оповещает что оно свершилось. Другая часть приложения (слушатель, также известный как наблюдатель) ждет этого оповещения. И когда до нее (части) доходит что событие состоялось, выполняется какое либо действие.
Наш класс ContactDetails не является исключением. Будучи расширением класса Panel, он автоматически содержит все события и слушатели которые являются частью класса Panel. Событие render было определено раньше. Процесс построение отображения начинается как только поступает оповещение о событии render.

```
this.fireEvent('render');
```

Объект Ext.Panel уже определил слушателя события для него. как только событие было обработано оно вызывает метод onRender().

```
this.addListener('render',this.onRender,this,
         {ct:this.ct,position:this.position});
```

Событие было получено, а затем передано. Слушатель ждал этой передачи, услышав выполнил дополнительное действие. Понимание этого процесса ключевой элемент при создании приложения управляемого событиями, а также приложений в **Ext JS**.



- « первая
- ‹ предыдущая
- 1
- 2
- 3
- следующая ›
- последняя »

- 🇬🇧 English

Голос