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

Мар 19 2010**Создаем собственные события**
Мы также с легкостью можем применять собственные события в пределах наших приложений **Ext JS**. Мы можем зарегистрировать новые события, к которым конечный разработчик приставит свои слушателей, или же мы можем добавть собственые слушатели для запуска других методов в пределах нашего собственного компонента.
Пойдем по примерам. Давайте создадим новое событие обновления для нашего компонента. Мы настроим наш метод initComponent() для регистрации нового события:

Исходный код примера:

Исходный код примера:

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

Этот новый код и есть все что нужно для регистрации нового события. Также мы пойдем дальше и зарегистрируем новый, буквенный слушатель события, который будем использовать для вызова нового метода который напишем здесь же:

Исходный код примера:

Исходный код примера:

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

С этим, каждый раз когда происходит событие update наше приложение будет вызывать метод onUpdate() нашего компонента ContactDetails.

```
onUpdate: function(data){
  console.log('in onUpdate');
}
```

В этом случае метод onUpdate() только выводит короткие сообщения в консоль отладки, которую видно если вы отлаживаете ваше приложение в Firefox при помощи плагина Firebug (консоль в Internet Explorer не поддерживается). Последним шагом будет обновление оповещения события. У нас уже есть метод update(), таким образом имеет смысл оповестить о его завершении обновления:

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

Здесь мы передали событие update вызвав метод fireEvent(), и отправив его вместе с данными как аргумент к любому методу ждущему события обновления. У нас может быть любое количество слушателей настроенных на определенное событие. Они могут быть как внутренними (частями компонента) так и внешними, со скриптом, ссылающимся на наш компонент.

Наш первый компонент: Завершен
Вот окончательный вид компонента который мы так долго создавали. С этого момента нам не нужно событие обновления. Но оставим его, так как оно может пригодиться позднее, а пока, удалим только метод onUpdate().
Example 2: ContactDetails.js

Исходный код примера:

Исходный код примера:

```
Ext.namespace('CRM.panels');
CRM.panels.ContactDetails = Ext.extend(Ext.Panel,{
 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}'
   ]),
 initComponent: function(){
   CRM.panels.ContactDetails.superclass.initComponent.call(this);
   if (typeof this.tpl === 'string') {
     this.tpl = new Ext.XTemplate(this.tpl);
   }
   this.addEvents('update');
 },
 onRender: function(ct, position) {
   CRM.panels.ContactDetails.superclass.onRender.call
                        (this, ct, position);
   if (this.data) {
     this.update(this.data);
   }
 },
 update: function(data) {
   this.data = data;
   this.tpl.overwrite(this.body, this.data);
   this.fireEvent('update', this.data);
 }
});
```

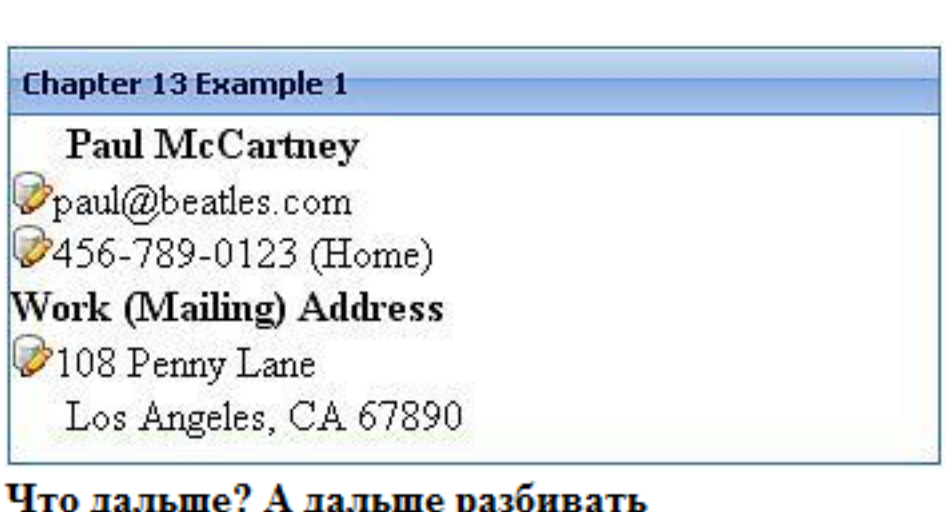
С новым компонентом у нас есть новый способ вызвать его в нашем приложении.
Example 2: ch13ex2.js

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',PHONETYPE:'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:'',STREE
T3:'',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 CRM.panels.ContactDetails({
    applyTo: 'chap13_ex01',
    title: 'Chapter 13 Example 1',
    data: userData[0]
  });
  updateContact = function(event,el,data){
   userDetail.update(data.data);
 }
  xt.get('actionLink').on('click',updateContact,this,
                          {data:userData[1]});
```

Мы взяли новый якорный элемент с id для actionLink из нашей страницы вызова и дали ему событие onclick которое обновляет данные в нашего объекта ContactDetails, userDetail. Нажатие по ссылке Update Data изменит контактную информацию с Джона на Пола.



Что дальше? А дальше разбивать
Итак мы создали наш первый собственный компонент. Это было не очень больно. Но посмотрев на сделанное кажется, что мы загнали себя в угол. Как выясняется каждый аспект нашего приложения, требующий контактных данных каждый раз будет отображать все что ему известно (имя, адрес, контактный телефон и т.д.). А что если нам нужен только адрес? Или имя?
Вот тут-то мы и займемся рефакторингом! Для наших целей мы по быстренькому разобьем это все на два компонента: один для UserDetail, а второй для AddressDetail.
Example 3: UserDetail.js

Исходный код примера:

Исходный код примера:

```
Ext.namespace('CRM.panels');
CRM.panels.UserDetail = Ext.extend(Ext.Panel,{
 width: 350,
 height: 125,
 data: {
  ID: 0,
  FIRSTNAME: '',
  LASTNAME: '',
  EMAIL: '',
  ADDRESSTYPE: 'Home (mailing)',
  STREET1: '',
  STREET2: '',
  STREET3: '',
  CITY: '',
  STATE: '',
  ZIP: '',
  PHONETYPE: 'Home',
   PHONE: ''
 },
 split: false,
 tpl: new Ext.Template([
     '<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 />'
   ]),
  initComponent: function(){
   CRM.panels.UserDetail.superclass.initComponent.call(this);
   if (typeof this.tpl === 'string') {
     this.tpl = new Ext.XTemplate(this.tpl);
   }
  },
  onRender: function(ct, position) {
   CRM.panels.UserDetail.superclass.onRender.call
                        (this, ct, position);
   if (this.data) {
     this.update(this.data);
   }
 },
 update: function(data) {
   this.data = data;
   this.tpl.overwrite(this.body, this.data);
 }
});
Ext.reg('userdetail',CRM.panels.UserDetail);
```

Example 3: AddressDetail.js

Исходный код примера:

Исходный код примера:

```
Ext.namespace('CRM.panels');
CRM.panels.AddressDetail = Ext.extend(Ext.Panel,{
 width:350,
 height:125,
 data: {
  ID: 0,
  FIRSTNAME: '',
  LASTNAME: '',
   EMAIL: '',
   ADDRESSTYPE: 'Home (mailing)',
   STREET1: '',
   STREET2: '',
   STREET3: '',
   CITY: '',
   STATE: '',
   ZIP: '',
   PHONETYPE: 'Home',
   PHONE: ''
 },
 split: false,
  tpl: new Ext.XTemplate([
    '<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}'
 ]),
 initComponent: function(){
   CRM.panels.AddressDetail.superclass.initComponent.call(this);
   if (typeof this.tpl === 'string') {
     this.tpl = new Ext.XTemplate(this.tpl);
   }
 },
 onRender: function(ct, position) {
   CRM.panels.UserDetail.superclass.onRender.call
                        (this, ct, position);
   if (this.data) {
     this.update(this.data);
   }
 },
 update: function(data) {
   this.data = data;
   this.tpl.overwrite(this.body, this.data);
 }
});
Ext.reg('addrdetail',CRM.panels.AddressDetail);
```

Разделив код на два компонента мы теперь можем использовать эти части в любом месте нашего приложения, и что важно, независимо.
Example 3: ch13ex3.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 Roa
d',STREET2:'',STREET3:'',CITY:'New York',
STATE:'NY',ZIP:'12345',PHONETYPE:'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:'',STREE
T3:'',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 CRM.panels.UserDetail({
   applyTo: 'chap13_ex03a',
   title: 'User Detail',
   data: userData[0]
 });
 var addrDetail = new CRM.panels.AddressDetail({
   applyTo: 'chap13_ex03b',
   title: 'Address Detail',
   data: userData[0]
 })
 updateContact = function(event,el,data){
   userDetail.update(data.data);
   addrDetail.update(data.data);
 }
  Ext.get('actionLink').on('click',updateContact,this,
                          {data:userData[1]});
```

ВАЖНО: *Практика лучше всего. Хорошо, посмотрите на два получившихся компонента. Довольно очевидно, что они все тот же объект с разными примененными XTemplate, и для них будет лучше иметь свой собственный родительский класс перегруженных, шаблонных методы и просто хранилищ умолчаний XTemplate и наших xtype-ов. Но для примеров мы постарается сделать все проще.*

Используем xtype: преимущества ленивой реализации
В предыдущем примере появилась новая строка в конце каждого файла класса:

```
Ext.reg('userdetail',CRM.panels.UserDetail);
```

Все что мы сделали здесь, это зарегестрировали новый компонент как xtype. Ну, а что это значит? Xtype это составляющий контейнерный элемент, зарегестрированный в библиотеке **Ext JS** для ленивой реализации объекта. Это значит, что мы можем использовать xtype как быстрый идентификатор объекта когда планируем наши приложения, и что эти типы загружаются в память браузера только когда используются. Это отлично увеличивает производительность всего приложения, особенно когда оно содержит большое количество объектов. Мы будем использовать xtype где только можно пока создаем настройки объекта, так что он не сразу отобразиться и не будет занимать столь ценные ресурсы памяти. Затем мы посмотрим все это на практике.

Использование наших компонентов внутри других объектов
Теперь, после создания собственных компонентов мы можем добавить их в любой другой контейнер внутри **Ext JS**. Мы можем использовать теперь наш xtype для ссылки на тип компонента для ленивой реализации и можно получить все преимущества от модульного дизайна. Давайте применим оба наших компонента с компоновкой границ для параллельного просмотра.
Example 4: ch14ex4.js

Исходный код примера:

Исходный код примера:

```
var ContactDetail = new Ext.Panel({
 title: 'Contact Details',
 applyTo: 'chap13_ex04',
 width: 400,
 height: 125,
 layout: 'border',
 frame: true,
 items:[{
   region: 'west',
   xtype: 'userdetail',
   width: 200,
   data: userData[0]
 },{
   region: 'center',
   xtype: 'addrdetail',
   width: 200,
   data: userData[0]
 }]
});
```

Мы определили Западный и Центральную области нашего BorderLayout как принадлежащие типам классов UserDetail и AddressDetail. Если эта компоновка была бы частью объекта window, эти два класса даже не загрузились бы в память до тех пор пока не появилось бы окно, снизив тем самым расход ресурсов.

Заключение
Расширение различных классов библиотеки **Ext JS** один из способов пойти по пути быстрой разработки приложений, позволяющей нам с легкостью создавать модульные, многократно используемые компоненты. В этой главе мы раскрыли некоторые аспекты библиотеки **Ext JS**.
Мы вкратце прошли ОО-разработку с **JavaScript**, и рассмотрели как ОО программный дизайн применим к библиотеке **Ext JS**. Мы потратили немного времени обсуждая основы ОО-програамировния (наследование замещение метода и кое-какую базовую терминологию).
Затем мы начали применять некоторые из этих концептов внутри **Ext JS** (определение собственных пространств имен, создание собственных компонентов и замещение методов наших суперклассов объектов).
Мы немного объяснили что представляет из себя приложение управляемое событиями и применили это на практике, создав собственные событие и слушатель для нашего компонента.
И наконец, мы рассмотрели xtype и как они важны для ленивой реализации в производительности приложения, и изменили код для того чтобы использовать xtype для наших новых компонентов и добавив их другие конейнеры объектов **Ext JS**.