Главная › Блоги › Lertion's блог

[Обучение Ext JS] :: Часть 3. Создание форм в Ext

Фев 18 2010**Кнопки и действия формы**
Теперь у нас есть неплохая форма и одна небольшая проблема – она не отправляет данные на сервер,что есть не правильно. Для исправления этого досадного недоразумения мы создадим несколько кнопок.
Наши кнопочки будут добавлены в настройки объекта кнопок, по тому же принципу что мы добавляли поля формы. Для этих кнопок нужны две вещи - текст, который будет на них написан и функция (называется обработчик") для корректного исполнения кода при нажатии кнопки.

Исходный код примера:

Исходный код примера:

```
buttons: [{
  text: 'Save',
  handler: function(){
    movie_form.getForm().submit({
        success: function(f,a){
          Ext.Msg.alert('Success', 'It worked');
        },
        failure: function(f,a){
          Ext.Msg.alert('Warning', 'Error');
        }
    });
  }
}, {
  text: 'Reset',
  handler: function(){
    movie_form.getForm().reset();
  }
}]
```

Обработчик снабжен функцией (или ссылкой на нее), которая будет выполнена когда кто-нибудь нажмет на кнопку. В этом случае нам понадобиться анонимная функция.

Добавление формы
В нашей FormPanel существует настройка url которая содержит название файла, формирующий данные подлежащие отправке. Она довольно простая, совсем как форма **HTML**, все наши поля будут размещены по этой url и таким образом могут быть переданы серверу. Внутри кнопки Save мы разместили анонимную функцию исполняющую следующий код,который запустит функцию отправки для нашей формы и отправит данные на сервер используя **AJAX**. Для отправки формы не требуется обновления страницы, все происходит фоном, в то время как страница остается без изменений.:

Исходный код примера:

Исходный код примера:

```
movie_form.getForm().submit({
   success: function(f,a){
     Ext.Msg.alert('Success', 'It worked');
   },
   failure: function(f,a){
     Ext.Msg.alert('Warning', 'Error');
   }
});
```

ВАЖНО: *Для корректной работы отправки формы, отправлять следует с веб-сервера.*

Success и *failure* применены для того чтобы вывести реакцию сервера. Это тоже анонимные функции, которые также могут быть отсылками на ранее встречавшиеся функции.

Вы заметили что у функция есть пара значений? они используются для того, чтобы выяснить какой ответ дал сервер.

Ответ от сервера
Когда вы добавляете вашу форму на сервер, скрипт серверной стороны обрабатывает полученную информацию и решает true она или false. Сообщение 'success' должно отправляться клиенту. Сообщение об ошибке может быть отправлено назад вместе с нашим ответом, и содержать текст, соответствующий названиям полей нашей формы.
Когда используют формы и подтверждение серверной стороны, успех должен быть булевым. Ответ от сервера будет выглядеть примерно так:

```
{
  success: false,
  errors: {
     title: "Sounds like a Chick Flick"
  }
}
```

Когда флаг *success* выставлен на *false*, это заставляет форму **Ext** выдать пользователю окно об ошибке.
Серверная проверка правильности нашего добавления формы дает нам способ искать информацию на стороны сервера, и возвращает ошибки, основанные на этом. Скажем, у нас есть база данных с названиями плохих фильмов, и мы не хотим, чтобы пользователи добавляли их в нашу базу данных. Мы можем отправить форму нашему скрипту, который проверяет базу данных и возвращает ответ, основываясь на результатах поиске в базы данных этого названия.
Если мы хотим отфильтровать "чик-флики" то мы можем сделать что-то вроде этого:

```
{
  success: false,
  errors: {
     title: "Sounds like a Chick Flick"
  },
   errormsg: "That movie title sounds like a chick flick."
}
```

Успешный false-ответ задействует форму и та выводит отчет об ошибке. Объект ошибок передается ответом. Форма использует этот объект для определения каждого сообщения об ошибке. Пара название/значение, существует в объекте ошибок для каждого случая поля формы.
Наш ответ в примере так же передает errormsg, который не используется формой, но к которому будет отдельный доступ для того чтобы выводить свое собственное сообщение об ошибке.
Возьмем дополнительное это сообщение, которое мы отсылаем назад и отобразим его в окне сообщений.

Исходный код примера:

Исходный код примера:

```
buttons: [{
  text: 'Save',
  handler: function(){
    movie_form.getForm().submit({
        success: function(f,a){
          Ext.Msg.alert('Success', 'It worked');
        },
        failure: function(f,a){
          Ext.Msg.alert('Warning', a.result.errormsg);
        }
    });
  }
}, {
  text: 'Reset',
  handler: function(){
    movie_form.getForm().reset();
  }
}]
```

Действие добавления формы отправляет информацию обратне к операторам success или failure. Первый парметр является объектом формы Ext, а второй объектом действия. В этом объекте действия доступно следующее:

Настройка		Описание
failureType	String	Сообщает о серверных и клиентских ошибках
response	Object	Содержит необработанную информацию об ответе сервера. В том числе полезную информацию заголовка.
result	Object	Проанализированный объект **JSON** основывающийся на ответе от сервера
type	String	Тип выполненного действия –submit или load

Теперь мы знаем что могут делать операторы ошибки, и можем выставить несколько простых проверок:

Исходный код примера:

Исходный код примера:

```
failure: function(f,a){
   if (a.failureType === Ext.form.Action.CONNECT_FAILURE)
         {Ext.Msg.alert('Failure', 'Server reported:
         '+a.response.status+' '+a.response.statusText);
   }
   if (a.failureType === Ext.form.Action.SERVER_INVALID){
     Ext.Msg.alert('Warning', a.result.errormsg);
   }
}
```

Проверяя тип ошибки мы выясняем была ли это ошибка соединения с сервером идействовать соответственно, порой даже предоставляя детали об ошибке (при правильном использовании результатов).

Загрузка формы из данных
Существуют три простых способа в которых форма используется в пользовательском интерфейсе:
- Ввод данных для единичного действия— поиск Google
- Создание новых данных
- Изменение существующих данных

Нам интересен третий пункт. Чтобы это сделать нам надо научиться загружать данные из источника (статичного или базы данных) в наш пользовательский интерфейс.

Загрузка статичных данных
Мы можем взять данные откуда-нибудь из нашего кода, и отобразить их как значение в поле формы. Одна линия кода установит значнеие полей:

```
movie_form.getForm().findField('title').
           setValue('Dumb & Dumber');
```

как только мы нанем работать с более сложными формами, этот метод станет проблемой. Поэтому нам селедуетпредусмотреть возможность всегда загружать данные через запросы **AJAX**. Сервер работает по том же принципу что и в тот раз когда мы загружали комбинированное окно:

Исходный код примера:

Исходный код примера:

```
<?php
// connection to database goes here
$result = mysql_query('SELECT * FROM movies WHERE id = '.$_
REQUEST['id']);
If (mysql_num_rows($result) > 0) {
  $obj = mysql_fetch_object($result);
  Echo '{success: true, data:'.json_encode($obj).'}';
}else{
  Echo '{success: false}';
}
?>
```

Это должно вернуть объект **JSON** с флагом успеха, и объект данных, который будет использован для заполнения значений полей формы. И выглядит примерно так:

Исходный код примера:

Исходный код примера:

```
{
  success: true,
  data:{
    "id":"1",
    "title":"Office Space",
    "director":"Mike Judge",
    "released":"1999-02-19",
    "genre":"1",
    "tagline":"Work Sucks",
    "coverthumb":"84m.jpg",
    "price":"19.95",
    "available":"1"
  }
}
```

Чтобы задействовать это, нам необходимо использовать обработчик загрузки формы:

```
movie_form.getForm().load({
   url:'data/movie.php',
   params:{
     id: 1
   }
});
```

Снабдить его url и конфигурацией настроек, как раз то, что нужно. Конфигурация настроек представляет собой то, что отправляется скриптам серверной части как параметры post/get. По умолчанию отсылаются параметры post.

Ссылки на объекты или конфигурация компонентов
По ходу первых глав мы начали использовать все больше и больше объектов конфигурации для настроек наших компонентов **Ext JS** вместо того чтобы подвергать их обработке. Давайте сравним оба метода.

Обработка

```
var test = new Ext.form.TextField({
   fieldLabel: 'Title',
   name: 'title',
   allowBlank: false
});
```

В данном случае компонент был создан с использованием памяти, даже если это еще не отобразилось на экране. В зависимости от того как конечный пользователь работает с вашим приложением, он может никогда и ни увидеть это самое поле. Но в то же время, если придет пора показать его, то оно появиться очень быстро.

Конфигурация компонента

```
{
  xtype: 'textfield',
  fieldLabel: 'Title',
  name: 'title',
  allowBlank: false
}
```

При помощи конфигурации компонента мы описываем то, что должно произойти, когда используется поле. Память требуется непосредственно на исполнение. Поле обрабатывается только тогда, когда пользователь задействовал его, и это может слегка замедлить его отображение.
У метода настройки компонентов есть много других преимуществ. Одно из них заключается в возможности отправлять конфигурации "по проводу". Это значит то, что код серверной стороны может создать конфигурацию для того чтобы образовать клиентский компонент.

Заключение
Мы заложили основание классического веб-приложения (формы) наделив силой **Ext JS** и создав уникально гибкий и мощный пользовательский интерфейс. Наша форма может подтверждать ввод данных пользователем, загружать информацию из базы данных и отправлять ее обратно на сервер. При помощи методов описанных в этой главе мы сможем создавать простые текстовые поисковики или сложные системы подтверждения ввода данных.

- « первая
- ‹ предыдущая
- 1
- 2
- 3
- 4

- English

Голос