Главная › Блоги › Lertion's блог

[Обучение Ext JS] :: Часть 12. Все дело в данных

Мар 18 2010**Определение данных**
Объекты хранилища можно легко настроить, они требуют источник данных и описание записей. Наши приложения знают как ждать данные, но нам надо сказать им как они, эти данные, будут выглядеть. К примеру, предположим, что наше приложение работает с медиа-массивом на нашем сайте. Нам понадобится серверный объект, который будет запрашивать папки файловой системы и возвращать информацию о содержащихся там файлах. Данные в таком случае могут выглядеть так:

Исходный код примера:

Исходный код примера:

```
{
 files: [
     {name: 'beatles.jpg', path:'/images/', size:46.5, lastmod:
                                        '12/21/2001'},
     {name: 'led_zepplin.jpg', path:'/images/', size:43.2,
                       lastmod: '2001-12-21 00:00:00'},
     {name: 'the_doors.jpg', path: '/images/', size:24.6, lastmod:
                                   '2001-12-21 00:00:00'},
     {name: 'jimi_hendrix.jpg', path: '/images/', size:64.3,
                        lastmod: '2001-12-21 00:00:00'}
 ]
}
```

Это маленький набор данных **JSON**. Его корнем являются файлы за которыми следует массив объектов. Теперь нам надо определить эти данные для нашего приложения:

Исходный код примера:

Исходный код примера:

```
var recordObj = new Ext.data.Record.create([{
 name: 'Name',
 mapping: 'name'
},{
 name: 'FilePath',
 mapping: 'path'
},{
 name: 'FileSize',
 mapping: 'size',
 type: 'float'
},{
 name: 'LastModified',
 mapping: 'lastmod',
 type: 'date',
 dateFormat: 'm/d/Y'
}]);
```

Мы применили название, к которому в нашем приложении будут ссылаться все поля, отмечая его переменной в объекте набора данных. Много переменных будут вписаны автоматически, но можно добавить в переменную 'type' (тип) для большей свободы маневра. Различные типы переменных:

- auto (стоит по умолчанию и не несет никаких изменений)
- string
- int
- float
- boolean
- date

Мы также применили особый формат строки к нашему объекту данных для того чтобы получить требуемый выход.

ВАЖНО: *Даты обычно передается как строки, которые обычно добавляются в объекты Date для правильного управления. Указывая тип даты, **Ext JS** обработает конверсию используя заданный dateFormat. Документация **Ext JS** для объекта Date предоставляет экстенсивный список формата, который используется для определения сотен пермутаций строк данных с которыми мы можем столкнуться в нашем коде.*

Еще немного по отображению данных
В предыдущем примере мы рассмотрели определение простого объекта **JSON**. Та же самая техника используется в объектах **XML**, с разницей в отображении записи в определенном узле. Вот тут и появляются настройки отображения. Эта опция может проложить путь **DOM** до узла с **XML**. Возьмем следующий пример:

```
<band>
 <members>4</members>
 <member>
  <firstname>Jimmy</firstname>
 <lastname>Paige</lastname>
</member>
<member>
 <firstname>Robert</firstname>
 <lastname>Plant</lastname>
</member>
...
```

Для того чтобы создать отображение узла first_name надо чтобы настройки выглядели вот так:

```
mapping:'member > first_name'
```

Массивы JavaScript проще, поскольку им не требуется отображение, только определение каждого поля в том порядке, в котором это будет видно в массиве:

```
var PinkFloyd = [
 ['1','David','Gilmour'],
 ['2','Roger','Waters'],
 ['3','Richard','Wright'],
 ['4','Nick','Mason']
]
```

Обратите внимание что здесь мы не создаем объект записи, а только поля опции настройки для Store.

```
fields:[
 {name:'id'},
 {name:'first_name'},
 {name:'last_name'}
]
```

Размещение данных в хранилище
Это почти так же просто как и получить данные с сервера, поскольку оно было заполнено ранее объектом панели и мы можем достичь цели используя знакомый синтаксис.
Example 3: scripts\chapter12_03.js

Исходный код примера:

Исходный код примера:

```
var recordObj = new Ext.data.Record.create([{
   name: 'NAME',
   mapping: 'name'
  },{
   name: 'DIRECTORY',
   mapping: 'path'
  },{
   name: 'SIZE',
   mapping: 'size',
 type: 'float'
},{
 name: 'DATELASTMODIFIED',
 mapping: 'lastmod',
 type: 'date',
 dateFormat: 'm/d/Y'
}]);
var ourStore = new Ext.data.JsonStore({
 url:'Chapter12Example.cfc',
 baseParams:{
   method: 'getFileInfo',
   returnFormat:'JSON',
   startPath: '/images/'
 },
 root:'files',
 id:'name',
 fields: recordObj,
 listeners:{
   beforeload:{
   fn: function(store, options){
       if (options.startPath && (options.startPath.length > 0)){
         store.baseParams.startPath = fieldVal;
       }
     },
       scope: this
   }
 }
});
ourStore.load();
```

Это связывает все части вместе. создавая наш объект хранилища данных. Сначала. мы используем опцию настройки url для определения места, откуда мы будем получать данные. Затем мы устанавливаем начальный набор параметров для отправки запроса на загрузку. И наконец, то что мы, возможно, хотели, отправка разные значения параметров на каждый запрос. Для этого мы можем определить особый 'слушатель'. В этом случае , когда бы не вызывался метод load(), если свойство startPath отправляется методу как аргумент прежде чем хранилище полчуит данные, оно изменит базовый параметр startPath для того чтобы совпадать с отправленными данными.

ВАЖНО: *Где скриншоты? Ну да, хороший вопрос. У хранилища данных, самого по себе, нет видимого отображения в браузере. Вот почему такие инструменты как Firefox с плагином Firebug, или Aptana development IDE может оказаться так важна при разработке **JavaScript**. Эти инструменты позволяют напрямую обрабатывать выход через специальные окна и следить за тем, что происходит с приложением.*

Используем DataReader для отображения данных
Некоторые приложения могут изначально возвращать данные в **XML**, или даже **JSON**, но порой бывает так, что формат данных не является тем, который ждет **Ext JS**. В качестве примере JsonStore, со встроенным JsonReader, ожидает входящий набор данных в следующем формате:

Исходный код примера:

Исходный код примера:

```
{
 'rootName': [
   {
    'variableName1': 'First record',
    'variableName2': '0'
   },{
    'variableName1': 'Second record',
    'variableName2': '3.5'
   }
 ]
}
```

У этого объекта (JSON) есть свойство rootName, которое является названием корня набора данных, содержащего массив объектов. Каждый из этих объектов имеет одни и те же атрибуты. Названия атрибутов даны в кавычках. Значения вообще обычно заключают в кавычки. Исключение составляют цифры. Их можно не "закавычивать".
Итак, серверный запрос возвращает этот формат, затем базовый JsonReader входящий в JsonStore автоматически парсирует полученные данные для наполнения объекта Store. Но что будет,если серверное приложение использует несколько другой формат? Например сервер Adobe ColdFusion 8 может автоматически возвращать наборы данных **JSON** для удаленных запросов, переводя их в любой из типов, родных для **ColdFusion**, в данные **JSON**. Но формат **JSON** у ColdFusion отличается, особенно когда дело качается запроса данных (как раз то из чего обычно и создается набор данных). Вот пример набора данных **JSON** возвращенный из объекта запроса **ColdFusion**:

Исходный код примера:

Исходный код примера:

```
{
 "COLUMNS":["NAME","SIZE","TYPE","DATELASTMODIFIED","ATTRIBUTES",
                                   "MODE","DIRECTORY"],
 "DATA":[
   ["IMG1.jpg",582360,"File","June, 13 2003
    23:50:08","","","H:\\wwwroot\\ExtBook\\images"],
   ["IMG2.JPG",1108490,"File","June, 13 2003
    23:50:52","","","H:\\wwwroot\\ExtBook\\images"],
   ["IMG3.JPG",1136108,"File","June, 13 2003
    23:51:02","","","H:\\wwwroot\\ExtBook\\images"],
   ["IMG4.JPG",1538506,"File","June, 13 2003
    23:51:12","","","H:\\wwwroot\\ExtBook\\images"]
 ]
}
```

Все нужные нам данные находятся здесь, но формат не может быть правильно парсирован базой JsonReader. Так что же нам делать?

Использование собственного DataReader
Встроенные классы **Ext JS** с непревзойденной легкостью могут использоваться сразу после установки, но (как мы сейчас увидим) иногда нам требуется что-то более особенное, возможно по причине других инструментов для особого сервера приложения (например ColdFusion), или, возможно, по причине внешнего API приложения (Flickr нпример). Мы, возможно, и смогли бы применить кое-что на серверной части, для перевода данных в нужный формат, но это создает ненужную загруженность сервера. Так почему бы не использовать клиента для обработки этих небольших превращений? Это поможет распределить нагрузку в наших приложениях и сделает более эффективным использование всех доступных ресурсов.
Ext JS дает нам оборудование для создания собственного DataReaders для отображения наших данных, а также простой способ для определения этого читателя в нашем хранилище.
В нашем текущем упражнении, нам, к счастью, на надо писать собственный DataReader. По той причине, что серверная платформа Adobe ColdFusion используется так широко, сообщество **Ext JS** уже создало собственный читатель для этой задачи. Простой поиск по форуму **Ext JS** (http://extjs.com/forum/) поможет найти много пользовательских читателей данных в самых разных форматах. Просто найдите время на то чтобы проверить код, который вы будете использовать, ведь он идет от третьих лиц (а потому не безопасен).
Используя CFJsonReader, а также несколько малых модификаций нашего скрипта, мы можем с легкостью читать формат данных **JSON** возвращенный ColdFusion.
Сначала давайте сконвертируем наш объект Record в простой массив, который затем станет моделью наших записей. Включите тег скрипта в ваш шаблон вызова, чтобы включить туда же файл CFJsonReader.js который, в свою очередь, важен для тега скрипта вашего пользовательского скрипта. А затем внесите в скрипт следующие изменения:

Исходный код примера:

Исходный код примера:

```
var recordModel = [
   {name:'file_name',mapping:'NAME'},
   {name:'file_size',mapping:'SIZE'},
   {name:'type',mapping:'TYPE'},
   {name:'lastmod',mapping:'DATELASTMODIFIED'},
   {name:'file_attributes',mapping:'ATTRIBUTES'},
   {name:'mode',mapping:'MODE'},
   {name:'directory',mapping:'DIRECTORY'}
];
```

Теперь мы определили наш новый DataReader как объект CFJsonStore:

```
var ourReader = new Ext.data.CFJsonReader(recordModel,
                     {id: 'NAME', root: 'DATA'});
```

Затем мы изменили наше хранилище данных с JsonStore на тип объекта основного хранилища и применим наш опции настройки нашего читателя для ourReader:

```
var ourStore = new Ext.data.Store({
 ...
 reader: ourReader
```

Мы также удалили настройки полей, id и корней из определения хранилища, так как теперь оно обрабатывается из определения читателя.
Последнее что мы сделаем, так это применим другой пользовательский слушатель в наш скрипт, таким образом, чтобы мы могли подтвердить был ли загружен набор данных правильно. Давайте измени опции настройки ваших слушателей, таким образом мы можем попытаться работать как слушатели событий:

Исходный код примера:

Исходный код примера:

```
listeners:{
 beforeload:{
   fn: function(store, options){
      if (options.startPath && (options.startPath.length > 0)){
        store.baseParams.startPath = fieldVal;
      }
   },
   scope:this
 },
 load:{
   fn: function(store, records, options){
      console.log(records);
   },
   scope: this
 }
}
```

Если вы используете для разработки Internet Explorer тогда линия этого кода сломается еще на console.log(), так как этот метод изначально не поддерживается этой средой (можно включить дополнительные скрипты для использования метода console.log() в IE (url: http://www.moxleystratton.com/article/ie-console)). Firefox, с плагином Firebug, тем не менее даст немного данных, поскольку однажды данные были получены, парсированы и загружены в хранилище данных, таким образом мы можем увидеть, что эти данные теперь находятся в нашем хранилище.

ВАЖНО: Слово о событиях
Многие из объеков **Ext JS** имеют события, запускаемые когда некоторое время предпринимаются попытки запустить определенные действия предпринимаемые для достижения определенного состояния. Приложение, управляемое событиями, в отличии от процедурной программной модели, может слушать на предмет изменений в приложении, например прием данных или изменения значения Record. Ext **JS** предоставляет исчерпывающий **API**, давая нам возможность применить свой собственный слушатель событий для обозначения действий в приложении. Для большей информации читайте **Ext JS API**: http://extjs.com/deploy/dev/docs/.

Финальная версия нашего скрипта выглядит таким образом:

Исходный код примера:

Исходный код примера:

```
 var recordModel = [
    {name:'file_name',mapping:'NAME'},
    {name:'file_size',mapping:'SIZE'},
    {name:'type',mapping:'TYPE'},
    {name:'lastmod',mapping:'DATELASTMODIFIED'},
    {name:'file_attributes',mapping:'ATTRIBUTES'},
    {name:'mode',mapping:'MODE'},
    {name:'directory',mapping:'DIRECTORY'}
 ];
 var ourReader = new Ext.data.CFJsonReader(recordModel,{id:'NAME',ro
ot:'DATA'});
 var ourStore = new Ext.data.Store({
   url:'Chapter12Example.cfc',
   baseParams:{
     method: 'getFileInfoByPath',
     returnFormat: 'JSON',
     queryFormat: 'column',
     startPath: '/images/'
   },
   reader: ourReader,
   listeners:{
     beforeload:{
       fn: function(store, options){
         if (options.startPath && (options.startPath.length > 0)){
           store.baseParams.startPath = options.startPath;
         }
       },
       scope:this
     },
  load: {
      fn: function(store,records,options){
      console.log(records);
    }
  },
  scope:this
 }
});
ourStore.load();
```

Это упаковывает куски кода, определяя внешний вид наших данных, настраивая наш пользовательскй читатель и устонавливая наше хранилище данных, с которым будет работать **JSON**. Загрузженный слушатель будет отображать те записи, полученные с сервера, которые сейчас в хранилище.

Получение нужного: Поиск данных
Теперь у нас есть данные, и нам нужно ими управлять. Как только мы загрузили наше хранилище записями весь набор данных обитает в кеше браузера, готовый к управлению или замене. Эти данные неизменны пока мы не переместим их со страницы или не удалим набор данных из хранилища.
Ext JS дает много различных опций для управления данными, все они упомянуты в **API**. Здесь же, мы рассмотрим наиболее общие вещи.

Поиск данных по значению поля
Первое что нам может понадобится, это найти определенную запись. Предположим что нам надо знать какая из записей последнего пример содержит изображение Джими Хендрикса:

```
var jimiPicIndex = ourStore.find('NAME','Jimi',0,false,false);
```

Этот метод вернет индекс первой записи в поле NAME, где имеется искомое Jimi. Поиск начнется с первой записи. Так же он не чувствителен к регистру.

Поиск данных по индексу записи
Индекс это хорошо. По крайней мере мы знаем что это за запись. Но теперь нам адо ее найти:

```
var ourImg = ourStore.getAt(jimiPicIndex);
```

Метод getAt() объекта Store найдет запись в хранилище пользуясь индексом.

Поиск данных по ID
Самый лучший способ для поиска уникальных записей, это поиск по ID. Если вы уже знаете ID вашей записи, все становится еще проще. Мы используем поле NAME как наш ID, так что давайте найдем запись:

```
var ourImg = ourStore.getById('jimi_hendrix.jpg');
```

Итак, теперь мы можем найти запись по частичному значению в пределах поля, получить запись по ее индексу или по значению ID.

Getting what you want: Filtering data
Иногда требуется найти только определенный тип данных в хранилище. Оно содержит весь набор данных (для кэширования и простого доступа), но вам нужно отделить определённые записи. В качестве примера, тег cfdirectory, использованный в серверном вызове может вернуть листинг всей директории, включая поддиректории. После получения данных может стать так, что нам нужны названия файлов только с startPath. Для этого можно отфильтровать клиентский набор данных для получения только записи типа, файл:

```
ourStore.filter('TYPE','File',false,false);
```

Это фильтрует наш набор данных по полю TYPE. Набор данных будет теперь содержать только те записи, у которых есть такое поле с значением File (начинается с начала и не чувствительно к регистру).
После того как мы закончим работать с фильтрами, мы наверняка захотим вернуться к старому, полному, набору данных. Когда мы фильтруем, остальные записи не исчезают, они все еще находятся в кеше и ждут своего часа. Но вместо того чтобы заново запрашивать сервер мы лучше просто снимем наш фильтр:

```
ourStore.clearFilter();
```

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