


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

Мар 18 2010

 **Ext JS** является очень мощной кросс-браузерной библиотекой, предоставляющей разработчику прекрасный инструментарий для создания браузерных приложений. Но в этом заключается нечто большее чем просто коробки и таблицы. Как говорит название этой главы, все дело в данных! Приложение без данных будет не более чем интерактивной статичной страницей, а наши пользователи хотяя работать с реальными данными. Одно из чудес веб-приложения заключается в том, что они работают по кругу, возвращая нас в то время когда действовала модель приложения клиент-сервер. Объекты обработанные **AJAX** и принадлежащие **Ext JS** дают нам способы работы с данными в настоящем времени прямо на сервере и в этой главе мы рассмотрим различные способы ее получения и размещение данных в приложениях основанных на **Ext JS**.

Понимание форматов данных

В **Ext JS** у нас есть десятки доступных компонентов, и большинство из них получают с сервера динамические данные. Главное, это знать какой тип данных обрабатывает каждый компонент и какие форматы для них приемлемы.

Начальные данные удаленной панели

Вы, возможно, заметили, что большинство компонентов изначально просто блок. Панели Tab и Accordion, а также область с содержимым окна блоки побольше (элемент <div>), или панели. Каждый из этих уникальных объектов имеет собственные методы и настройки, хотя и продолжает объект Ext.Panel. Применение динамических данных в начальной панели сверхпросто, в основном потому, что она использует самый простой из форматов: обычный текст или **HTML**. Наш первый пример будет загружать простую HTML-страницу на панель. Сначала нам понадобится рендеринг страницы:

```
Пример 1: ch12ex1.html
...   <div id="mainContent">
        <div id="chap12_ex01"></div>
    </div>
...
Здесь мы показали что пойдет внутрь тега <body> нашего примера HTML.
Затем, нам понадобится серверная сторона для вызова содержимого:
Пример 1: chapter_12\example1ajax.html
    <b>William Shakespeare</b></i><Poet Lauraette</i><br />
```

И последнее, это скрипт для создания примера панели:
Пример 1: scripts\chapter12_01.js

Исходный код примера:

Исходный код примера:

```
Ext.onReady(function(){
    var example1 = new Ext.Panel({
        applyTo:'chap12_ex01',
        title:'Chapter 12: Example 1',
        width:250,
        height:250,
        frame:true,
        autoLoad:{
            url:'example1ajax.html'
        }
    });
});
```

Вызов шаблона ch12ex1.html в браузере запустит скрипт. Теперь давайте посмотрим на код и постараемся понять что же мы делаем..

1. Ждем пока **DOM** отрендерится (Ext.onReady() function).
2. Создаем новый объект Ext.Panel, отрендеренный в элемент chap12_ex01 (div на странице с рендерингом).
3. Загружаем содержимое внешнего URL example1ajax.html.

очень просто вставить шаблон **HTML** в качестве содержимого для предмета панели. Но мы хотим по настоящему динамические данные. Давайте создадим на их на основе этого примера и вытащим данные из сервера приложения. Мы можем использовать тот же самый атрибут autoLoad для вызова страницы обработки из сервера приложения для возврата данных.

ВАЖНО: *Ext JS является скриптовой библиотекой на клиентской стороне, и потому язык программирования сервера может быть любой удобный вам. Вы должно быть заметили, что некоторые из примеров этой книги написаны в PHP. Примеры в этой главе требуют сервера Adobe ColdFusion для обработки динамических данных, а посему вам придется скачать и установить бесплатное Developer's Edition для корректной работы примеров. Мы используем ColdFusion чтобы показать две разные вещи:*

- 1) Любой серверный процессор может скормить свои данные **Ext JS**.
- 2) Не каждый сервер приложения или удаленные приложение будут возвращать данные в стандартном формате. Наши примеры Custom Data Readers, (о них позже) продемонстрируют это.

Бесплатный сервер Adobe ColdFusion Developer's Edition доступен на <http://www.adobe.com>. Ознакомьтесь с README.txt для большей информации по доступу к файлам примеров этой главы.

Для Примера 2, мы изменим id нашего <div> на chap12_ex02, и используем файл chapter12_02.js для загрузки данных разными способами, и у нас будет следующее:

Example 2: scripts\chapter12_02.js

Исходный код примера:

Исходный код примера:

```
var example2 = new Ext.Panel({
    applyTo:'chap12_ex02',
    title:'Chapter 12: Example 2',
    width:250,
    height:250,
    frame:true,
    autoLoad:{
        url:'Chapter12Example.cfc',
        params:{
            method:'example1',
            returnFormat:'plain',
            id:1289
        }
    }
});
```

Вы заметите, что URL теперь называется **Adobe ColdFusion Component(CFC)**, отправляя некоторые параметры для получения результатов. У нас есть очень простой **CFC**, который использует маленькие запросы, основываясь на отправленных настройках для создания содержимого, отправляемого обратно через запрос **AJAX**.

Example 2: chapter_12\Chapter12Example.cfc

Исходный код примера:

Исходный код примера:

```
<cfcomponent output="false">
    <cffunction name="example2" access="remote" output="false"
        returntype="string">
        <cfargument name="id" type="numeric" required="true" />
        <cfset var output = "" />
        <cfset var q = "" />
        <cftry>
            <cfquery name="q" datasource="chapter12">
                SELECT firstName,
                    lastName,
                    occupation
                FROM People
                WHERE ID = <cfqueryparam cfsqltype="cf_sql_integer"
                    value="#ARGUMENTS.id#" />
            </cfquery>
            <cfcatch type="database">
                <!-- Place Error Handling Here --->
            </cfcatch>
        </cftry>
        <cff IsDefined("q.recordcount") and q.recordcount>
            <cfsavecontent variable="output"><cfoutput>
                #q.firstName# #q.lastName# #q.occupation#<br />
            </cfoutput></cfsavecontent>
        </cff>
        <cfreturn output />
    </cffunction>
</cfcomponent>
```

Так как цель этой книги не учить вас серверному языку, то на этом простом способе и закончим. **CFC** берет аргумент ID, который передается в запрос таблицы People. Если запись возвращается из запроса, то простая строка возвращается predetermined формате: FirstName LastName: Occupation. Вызов **AJAX** (autoLoad) отправляет несколько параметров: метод который следует запустить в CFC, тип формата для возврата (в нашем случае простой текст), и аргументы метода (в нашем случае id).

Глюки с данными HTML

Вы должны помнить о том, что надо вызывать данные только через **AJAX** домена вашего сайта. Попытка сослаться на данные с сайта за пределами вашего домена вызовет ошибку на браузерном уровне, так как это считается кросс-сайтовым скриптованием. Кросс-сайтовое скриптование является средством доставки данных, которые возвращают небольшие наборы данных могут быстро забить поток, из-за капризности синтаксиса XML. Другим затруднением с XML является движок браузера. Набор данных XML, который подходит для Mozilla Firefox может быть отклонен Internet Explorer, и парсинговый движок Internet Explorer для XML медленный. JSON, или JavaScript Object Notation, пакеты данных имеют склонность к гораздо меньшим размерам, и меньше забивают канал. Если используемый объект может принимать это, простой массив может быть еще меньше, хотя вы потеряете описательную природу, которую дает синтаксис JSON или XML.

Другие форматы

У **Ext JS** есть возможность использовать внешние данные в различных форматах:

Формат	Формат
Plain Text	Eric Clapton is a consummate guitarist
HTML	Jimi Hendrix is considered, by some, to have been one of the finest blues guitarists that ever lived
JSON	var theBeatles = { 'members': 4, 'band': [{'id':1,'first_name':'John', 'last_name':'Lennon'}, {'id':2,'first_name':'Paul', 'last_name':'McCartney'}, {'id':3,'first_name':'George', 'last_name':'Harrison'}, {'id':4,'first_name':'Ringo', 'last_name':'Starr'}] };
XML	<band> <members>4</members> <member> <firstname>Jimmy</firstname> <lastname>Paige</lastname> </member> <member> <firstname>Robert</firstname> <lastname>Plant</lastname> </member> ...
JavaScript Array	var PinkFloyd = [['1','David','Gilmour'], ['2','Roger','Waters'], ['3','Richard','Wright'], ['4','Nick','Mason']]

Решение о выборе формата зависит от того какой объект **Ext JS** вы используете. Большинство разработчиков любят XML, так как многие базы данных (например MS SQL, Oracle, MySQL, PostgreSQL, и DB2) по умолчанию работают с ними, так же как и многие веб-службы. Хотя это и хорошо, особенно когда пользуетесь различными внешними приложениями, XML иногда может быть очень капризным. Запросы данных, которые возвращают небольшие наборы данных могут быстро забить поток, из-за капризности синтаксиса XML. Другим затруднением с XML является движок браузера. Набор данных XML, который подходит для Mozilla Firefox может быть отклонен Internet Explorer, и парсинговый движок Internet Explorer для XML медленный. JSON, или JavaScript Object Notation, пакеты данных имеют склонность к гораздо меньшим размерам, и меньше забивают канал. Если используемый объект может принимать это, простой массив может быть еще меньше, хотя вы потеряете описательную природу, которую дает синтаксис JSON или XML.

Объект хранилища данных

Большинство объектов **Ext JS** (и даже панели, с некоторыми доработками) принимают данные как Records (записи), или Nodes (узлы). Записи обычно хранятся в объекте хранилища данных. Представьте, что приложение похоже на электронный журнал, а каждая запись это ряд таблицы.

Пакет данных содержит большое количество объектов для работы с данными. У вас есть несколько типов хранилищ:

- JsonStore: Специально для работы с данными JSON
- SimpleStore: Для работы с массивами и данными JSON
- GroupingStore: Содержит 'группы' наборов данных

Каждому хранилищу понадобится свой считыватель для парсинга входящих данных и как и хранилищ, их несколько:

- ArrayReader: Для работы с массивами JavaScript
- JsonReader: Для работы с наборами данных JSON
- XmlReader: Др работы с наборами данных XML

ВАЖНО: *Объект TreePanel не использует обычное хранилище данных, но у него есть собственное хранилище, называемое TreeLoader, которое отправляется в конфигурацию через опцию настройки загрузчика. TreeLoader принимает простые массивы определенных объектов, что больше похоже на ArrayReader. Для большей информации смотрите Ext JS API (<http://extjs.com/deploay/dev/docs/>).*

- 1
- 2
- 3
- [следующая](#) >
- [последняя](#) »