Жизненный цикл компонентов

Все компоненты Ext JS имеют определенный жизненный цикл, который включает следующие стадии:

1. **Инициализация**
2. **Рендеринг**
3. **Уничтожение**

Посмотрим на примере простого объекта панели, что они представляют:

```
Ext.onReady(function(){
var panel=Ext.create('Ext.Panel', {
            title: 'Приложение',
            width: 150,
            id: 'myPanel',
            html: '<h2>Привет мир!</h2>',
            renderTo: Ext.getBody()
        });
});
```

Инициализация

На этом этапе происходит создание компонента, которое включает ряд шагов:

1. *Применение конфигурационных настроек*. На этом этапе компонент применяет все установленные параметры и свойства, которые мы передаем, например, через конструктор.

   Так, на этом этапе определенные нами свойства `title`, `width` и `renderTo` копируются в объект панели.

2. *Регистрация базовых событий компонента*. Каждый подкласс класса Ext.Component по умолчанию имеет некоторый набор основных событий: enable, disable, beforeshow, show, beforehide, hide, beforerender, render, beforedestroy, destroy.

3. *Регистрация компонента*. Каждому создаваемому компоненту присваивается идентификатор, если мы его не определим явным образом в качестве свойства.

   В случае с вышеопределенной панелькой используется определенный в качестве свойства id.

4. *Загрузка плагинов*. Если для компонента нужны или определены какие-нибудь плагины, они загружаются на этом этапе. Если плагины не определены, как в случае с нашей панелью, то этап пропускается.

5. *Вызов метода initComponent*. Этот метод выполняет большую часть работы по инициализации компонентов и классов

6. *Добавление нового экземпляра класса в объект Ext.ComponentManager*. Благодаря чему затем можно будет получать компонент по его id через метод `Ext.getCmp`: `var myPanel = Ext.getCmp('myPanel');`

7. *Прямой рендеринг (если применяется)*. Если для компонента указан параметр `renderTo/applyTo`, то применяется рендеринг, иначе компонент не визуализируется, пока не будет вызван метод render

Рендеринг

Стадия рендеринга также проходит через ряд этапов:

1. *Вызов собыия beforerender*. Компонент вызывает событие beforerender и проверяет результаты всех зарегистрированных обработчиков этого события. Если обработчик события возвращает `false`, то компонент останавливает рендеринг.

2. *Установка свойства z-index*. Если компонент является "плавающим", например, меню или окно, то у него устанавливается CSS-свойство `z-index` для корректного отображения.

3. *Установка контейнера*. Для компонента устанавливается контейнер, в котором будет производится рендеринг компонента.

4. *Вызов метода onRender*. На этом этапе добавляются все необходимые элементы DOM к компоненту, и он приобретает те черты, которые мы потом можем лицезреть на экране
5. *Компонент отображается на экране*. По умолчанию многие компоненты не отображаются и остаются скрытыми, используя стандартные классы Ext CSS, например, класс "x-hidden". Но если у компонента свойство **autoShow** имеет значение `true`, то все классы CSS, скрывающие элемент, удаляются
6. *Применение пользовательских классов и стилей CSS*. На этом этапе применяются пользовательские стили и классы через использование свойств `cls` и `style`
7. *Инициализация события render*. На этом этапе все необходимые элементы уже добавлены в структуру DOM страницы, а стили уже применены. А событие уведомляет систему, что компонент готов к использованию.
8. *Инициализация контента*. Существует несколько способов добавления контента: через свойство `html`, через свойство `contentEl` (которое содержит id элемента, используемого в качестве содержимого для компонента) и через свойство `tpl` (которое принимает шаблон контента)

   В случае с использованной выше панелькой задано свойство `html`, поэтому контент для панели берется из него.
9. *Вызов метода afterRender*. Этот метод автоматически вызывается в методе `render` компонента. В нем выполняется вся необходимая логика пострендеринга
10. *Возникновение события afterRender*.
11. *Скрытие или отключение компонента*. Если у компонента свойства `hidden` или `disabled` имеют значение `true`, то вызываются соответственно методы `hide` или `disable`

После этого компонент готов к принятию ввода от пользователя и с ним можно взаимодействовать.

Уничтожение

На этом этапе происходит удаление компонента из структуры DOM веб-страницы, отмена его регистрации и отмена регистрации обработчиков событий. Метод destroy, производящий уничтожение компонента, производится либо вручную в коде, либо вызывается контейнером и также предполагает ряд этапов:

1. *Инициализация события beforedestroy*. Если обработчик этого события возвращает `false`, то уничтожение компонента отменяется
2. *Вызов метода beforeDestroy*. Выполняет некоторую логику до уничтожения компонента
3. *Удаление элемента и всех зарегистрированных обработчиков событий элемента*.
4. *Вызов метода onDestroy*. Этот метод выполняет дополнительные действия по уничтожению компонента, например, удаление хранилищ данных
5. *Уничтожение всех плагинов компонента*
6. *Удаление всех связанных узлов из DOM*, если компонент прошел рендеринг
7. *Инициализация события destroy*. Срабатывают все зарегистрированные обработчики данного события. которые сигнализируют, что компонент удален из структуры DOM
8. *Удаление обработчиков событий компонента*.

Пользуясь готовыми компонентами, не приходится ломать голову над всеми этими стадиями жизненного цикла. Однако при разработке собственных компонентов следует обратить на это внимание, чтобы потом избежать непредвиденных проблем в работе приложения. Например, разработчики могут забыть провести чистку разных объектов при уничтожении своего компонента - хранилищ данных, которых продолжат опрашивать веб-сервер на предмет получения данных и т.д.

Назад Содержание Вперед