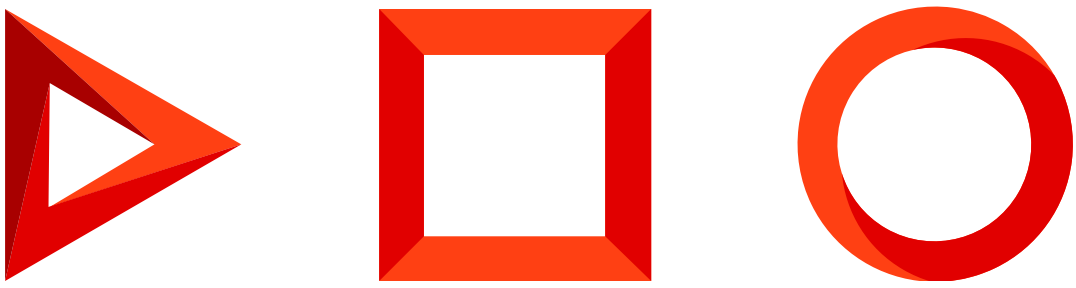
Creatio Academy

Модули

Класс модуля

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Класс модуля

Средний

Объявление класса модуля

Объявление классов — функция JavaScript-фреймворка `ExtJs`. Для объявления классов используется стандартный механизм библиотеки — метод `define()` глобального объекта `Ext`.

Пример объявления класса с помощью метода `define()`

```
// Название класса с соблюдением пространства имен.
Ext.define("Terrasoft.configuration.ExampleClass", {
    // Сокращенное название класса.
    alternateClassName: "Terrasoft.ExampleClass",
    // Название класса, от которого происходит наследование.
    extend: "Terrasoft.BaseClass",
    // Блок для объявления статических свойств и методов.
    static: {
        // Пример статического свойства.
        myStaticProperty: true,

        // Пример статического метода.
        getMyStaticProperty: function () {
            // Пример доступа к статическому свойству.
            return Terrasoft.ExampleClass.myStaticProperty;
        }
    },
    // Пример динамического свойства.
    myProperty: 12,
    // Пример динамического метода класса.
    getMyProperty: function () {
        return this.myProperty;
    }
});
```

Примеры создания экземпляров класса представлены ниже.

Пример создания экземпляра класса по полному имени

```
// Создание экземпляра класса по полному имени.
var exampleObject = Ext.create("Terrasoft.configuration.ExampleClass");
```

Пример создания экземпляра класса по псевдониму

```
// Создание экземпляра класса по сокращенному названию – псевдониму.
var exampleObject = Ext.create("Terrasoft.ExampleClass");
```

Наследование класса модуля

В большинстве случаев класс модуля правильно наследовать от `Terrasoft.configuration.BaseModule` или `Terrasoft.configuration.BaseSchemaModule`, в которых реализованы **методы**:

- `init()` — метод инициализации модуля. Отвечает за инициализацию свойств объекта класса, а также за подписку на сообщения.
- `render(renderTo)` — метод отрисовки представления модуля в DOM. Возвращает представление модуля. Принимает единственный аргумент `renderTo`, в который будет добавлено представление объекта модуля.
- `destroy()` — метод, отвечающий за удаление представления модуля, удаление модели представления, отписку от ранее подписанных сообщений и уничтожение объекта класса модуля.

Ниже приведен пример класса модуля, наследуемого от `Terrasoft.BaseModule`. Данный модуль добавляет кнопку в DOM. При клике на кнопку выводится сообщение, после чего она удаляется из DOM.

Пример класса модуля, наследуемого от `Terrasoft.BaseModule`

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        // Короткое название класса.
        alternateClassName: "Terrasoft.ModuleExample",
        // Класс, от которого происходит наследование.
        extend: "Terrasoft.BaseModule",
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        Ext: null,
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        sandbox: null,
        /* Обязательное свойство. Если не определено, будет сгенерирована ошибка на уровне
           "Terrasoft.core.BaseObject", так как класс наследуется от "Terrasoft.BaseModule".*/
        Terrasoft: null,
        // Модель представления.
        viewModel: null,
        // Представление. В качестве примера используется кнопка.
        view: null,
        /* Если не реализовать метод init() в текущем классе,
           то при создании экземпляра текущего класса будет вызван метод
```

```
            init() класса-родителя Terrasoft.BaseModule.*/
        init: function () {
            // Вызывает выполнение логики метода init() класса-родителя.
            this.callParent(arguments);
            this.initViewModel();
        },
        // Инициализирует модель представления.
        initViewModel: function () {
            /* Сохранение контекста класса модуля
               для доступа к нему из модели представления.*/
            var self = this;
            // Создание модели представления.
            this.viewModel = Ext.create("Terrasoft.BaseViewModel", {
                values: {
                    // Заголовок кнопки.
                    captionBtn: "Click Me"
                },
                methods: {
                    // Обработчик нажатия на кнопку.
                    onClickBtn: function () {
                        var captionBtn = this.get("captionBtn");
                        alert(captionBtn + " button was pressed");
                        /* Вызывает метод выгрузки представления и модели представления,
                           что приводит к удалению кнопки из DOM.*/
                        self.destroy();
                    }
                }
            });
        },
        /* Создает представление (кнопку),
           связывает ее с моделью представления и вставляет в DOM.*/
        render: function (renderTo) {
            // В качестве представления создается кнопка.
            this.view = this.Ext.create("Terrasoft.Button", {
                // Контейнер, в который будет помещена кнопка.
                renderTo: renderTo,
                // HTML-атрибут id.
                id: "example-btn",
                // Название класса.
                className: "Terrasoft.Button",
                // Заголовок.
                caption: {
                    // Связывает заголовок кнопки
                    // со свойством captionBtn модели представления.
                    bindTo: "captionBtn"
                },
                // Метод-обработчик события нажатия на кнопку.
                click: {
                    /* Связывает обработчик события нажатия на кнопку
```

```
                        с методом onClickBtn() модели представления.*/
                    bindTo: "onClickBtn"
                },
                /* Стиль кнопки. Возможные стили определены в перечислении
                   Terrasoft.controls.ButtonEnums.style.*/
                style: this.Terrasoft.controls.ButtonEnums.style.GREEN
            });
            // Связывает представление и модель представления.
            this.view.bind(this.viewModel);
            // Возвращает представление, которое будет вставлено в DOM.
            return this.view;
        },
        // Удаляет неиспользуемые объекты.
        destroy: function () {
            // Уничтожает представление, что приводит к удалению кнопки из DOM.
            this.view.destroy();
            // Удаляет неиспользуемую модель представления.
            this.viewModel.destroy();
        }
    });
    // Возвращает объект модуля.
    return Terrasoft.ModuleExample;
});
```

Мониторинг переопределения членов класса модуля

При наследовании класса модуля в классе-наследнике могут быть переопределены как публичные, так и приватные свойства и методы базового модуля.

В Creatio **приватными свойствами или методами класса** считаются те, названия которых начинаются с нижнего подчеркивания, например, `_privateMemberName`.

В Creatio присутствует функциональность **мониторинга переопределения приватных членов класса** — класс `Terrasoft.PrivateMemberWatcher`.

Назначение мониторинга — при определении пользовательского класса проверять выполнение переопределений приватных свойств или методов, которые объявлены в родительских классах. При этом в режиме отладки отображается предупреждение в консоли браузера.

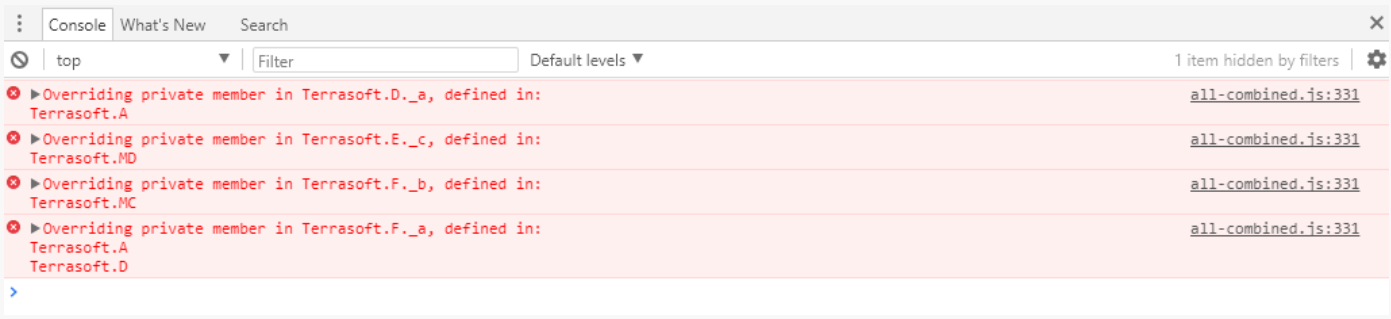
Например, в пользовательский пакет добавлена схема модуля, исходный код которой приведен ниже.

Пример переопределения приватных свойств класса модуля

```
define("UsrPrivateMemberWatcher", [], function() {
    Ext.define("Terrasoft.A", {_a: 1});
    Ext.define("Terrasoft.B", {extend: "Terrasoft.A"});
    Ext.define("Terrasoft.MC", {_b: 1});
    Ext.define("Terrasoft.C", {extend: "Terrasoft.B", mixins: {ma: "Terrasoft.MC"}});
    Ext.define("Terrasoft.MD", {_c: 1});
```

```
    // Переопределение свойства _a.
    Ext.define("Terrasoft.D", {extend: "Terrasoft.C", _a: 3, mixins: {mb: "Terrasoft.MD"}});
    // Переопределение свойства _c.
    Ext.define("Terrasoft.E", {extend: "Terrasoft.D", _c: 3});
    // Переопределение свойств _a и _b.
    Ext.define("Terrasoft.F", {extend: "Terrasoft.E", _b: 3, _a: 0});
});
```

После загрузки этого модуля в консоли отобразятся предупреждения о том, что приватные члены базовых классов были переопределены.



Инициализация экземпляра класса модуля

Синхронная инициализация

Модуль инициализируется синхронно, если при его загрузке явно не указано свойство `isAsync: true` конфигурационного объекта, передаваемого в качестве параметра метода `loadModule()`. Например, при выполнении

```
this.sandbox.loadModule([moduleName])
```

методы класса модуля будут загружены синхронно. Первым будет вызван метод `init()`, после которого сразу же будет вызван метод `render()`.

Пример реализации синхронно инициализируемого модуля

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        alternateClassName: "Terrasoft.ModuleExample",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
        init: function () {
            // При инициализации выполнится первым.
```

```
        },
        render: function (renderTo) {
            // При инициализации модуля выполнится сразу же после метода init().
        }
    });
});
```

Асинхронная инициализация

Модуль инициализируется асинхронно, если при его загрузке явно указано свойство `isAsync: true` конфигурационного объекта, передаваемого в качестве параметра метода `loadModule()`. Например, при выполнении

```
this.sandbox.loadModule([moduleName], {
    isAsync: true
})
```

первым будет вызван метод `init()`, в который будет передан единственный параметр — callback-функция с контекстом текущего модуля. При вызове callback-функции будет вызван метод `render()` загружаемого модуля. Представление будет добавлено в DOM только после выполнения метода `render()`.

Пример реализации асинхронно инициализируемого модуля

```
define("ModuleExample", [], function () {
    Ext.define("Terrasoft.configuration.ModuleExample", {
        alternateClassName: "Terrasoft.ModuleExample",
        Ext: null,
        sandbox: null,
        Terrasoft: null,
        // При инициализации модуля выполнится первым.
        init: function (callback) {
            setTimeout(callback, 2000);
        },
        render: function (renderTo) {
            // Метод выполнится с задержкой в 2 секунды,
            // Задержка указана в аргументе функции setTimeout() в методе init().
        }
    });
});
```

Цепочки модулей

Цепочки модулей — механизм, который позволяет отобразить представление одной модели на месте представления другой модели. Например, для установки значения поля на текущей странице необходимо отобразить страницу `SelectData` для выбора значения из справочника. То есть, на месте контейнера модуля текущей страницы должно отобразиться представление модуля страницы выбора из справочника.

Для построения цепочки необходимо добавить свойство `keepAlive` в конфигурационный объект загружаемого модуля. Например, в модуле текущей страницы `CardModule` необходимо вызвать модуль выбора из справочника `selectDataModule`.

Пример вызова модуля выбора из справочника `selectDataModule` в модуле страницы `CardModu`

```
sandbox.loadModule("selectDataModule", {
    // Id представления загружаемого модуля.
    id: "selectDataModule_id",
    // Представление будет добавлено в контейнер текущей страницы.
    renderTo: "cardModuleContainer",
    // Указывает, чтобы текущий модуль не выгружался.
    keepAlive: true
});
```

После выполнения кода будет построена цепочка из модуля текущей страницы и модуля страницы выбора из справочника. Добавление еще одного элемента в цепочку позволяет из модуля текущей страницы `selectData` по нажатию на кнопку [*Добавить новую запись*] ([*Add new record*]) открыть новую страницу. Добавив в цепочку еще один элемент, из модуля текущей страницы `selectData` по нажатию на кнопку [*Добавить новую запись*] ([*Add new record*]) откроется новая страница. Таким образом, в цепочку модулей можно добавлять неограниченное количество экземпляров модулей.

Активный модуль (тот, который отображен на странице) — последний элемент цепочки. Если установить активным элемент из середины цепочки, то будут уничтожены все элементы, находящиеся в цепочке после него. Активация элемента цепочки выполняется путем вызова функции `loadModule()`, в параметр которой необходимо передать идентификатор модуля.

Пример вызова функции `loadModule()`

```
sandbox.loadModule("someModule", {
    id: "someModuleId"
});
```

Ядро уничтожит все элементы цепочки, после чего вызовет методы `init()` и `render()`. При этом в метод `render()` будет передан контейнер, который содержит предыдущий активный модуль. Модули цепочки могут работать, как и раньше — принимать и отправлять сообщения, сохранять данные и т. д.

Если при вызове метода `loadModule()` в конфигурационный объект не добавлять свойство `keepAlive` или добавить его со значением `keepAlive: false`, то цепочка модулей будет уничтожена.