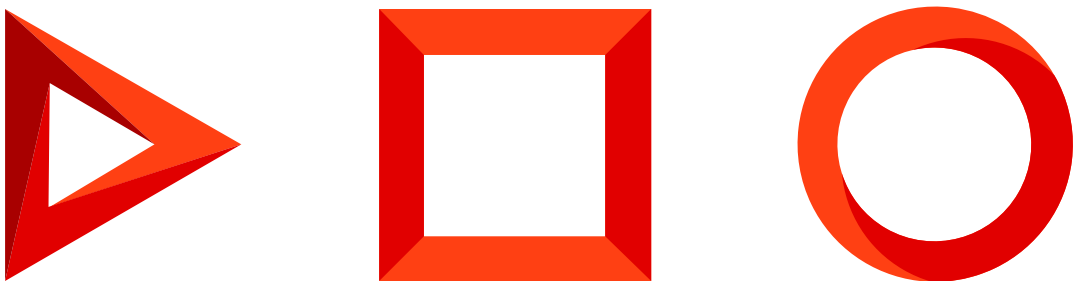
Creatio Academy

Понятие 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

Понятие модуля

Сложный

Концепция AMD

Front-end часть приложения Creatio представляет собой набор блоков функциональности, каждый из которых реализован в отдельном **модуле**. Согласно **концепции** Asynchronous Module Definition (AMD), в процессе работы приложения выполняется асинхронная загрузка модулей и их зависимостей. Таким образом, концепция AMD позволяет подгружать только те данные, которые необходимы для работы в текущий момент.

Концепция AMD поддерживается различными JavaScript-фреймворками. В Creatio для работы с модулями используется **загрузчик** RequireJS.

Модульная разработка в Creatio

Модуль — фрагмент кода, инкапсулированный в обособленный блок, который может быть загружен и самостоятельно выполнен.

Создание модулей в специфике JavaScript декларируется **паттерном программирования** "Модуль". Классическая **реализация паттерна** — использование анонимных функций, возвращающих определенное значение (объект, функцию и т. д.), которое ассоциируется с модулем. При этом значение модуля экспортируется в глобальный объект.

Пример экспорта значения модуля в глобальный объект

```
// Немедленно вызываемое функциональное выражение (IIFE). Анонимная функция,
// которая инициализирует свойство myGlobalModule глобального объекта функцией,
// возвращающей значение модуля. Таким образом, фактически происходит загрузка модуля,
// к которому в дальнейшем можно обращаться через глобальное свойство myGlobalModule.
(function () {
    // Обращение к некоторому модулю, от которого зависит текущий модуль.
    // Этот модуль на момент обращения к нему должен быть загружен
    // в глобальную переменную SomeModuleDependency.
    // Контекст this в данном случае – глобальный объект.
    var moduleDependency = this.SomeModuleDependency;
    // Объявление в свойстве глобального объекта функции, возвращающей значение модуля.
    this.myGlobalModule = function () { return someModuleValue; };
}());
```

Интерпретатор, обнаруживая в коде такое функциональное выражение, сразу вычисляет его. В результате выполнения в свойство `myGlobalModule` глобального объекта будет помещена функция, которая будет возвращать значение модуля.

Особенности подхода:

- Сложность декларирования и использования модулей-зависимостей.
- В момент выполнения анонимной функции все зависимости модуля должны быть загружены.
- Загрузка модулей-зависимостей выполняется в заголовке страницы через HTML-элемент `<script>`. Обращение к модулям-зависимостям осуществляется через имена глобальных переменных. При этом разработчик должен четко представлять и реализовывать порядок загрузки модулей-зависимостей.
- Как следствие предыдущего пункта — модули загружаются до начала рендеринга страницы, поэтому в модулях нельзя обращаться к элементам управления страницы для реализации пользовательской логики.

Особенности использования подхода в Creatio:

- Отсутствие возможности динамической загрузки модулей.
- Применение дополнительной логики при загрузке модулей.
- Сложность управления большим количеством модулей со многими зависимостями, которые могут перекрывать друг друга.

Загрузчик RequireJS

Загрузчик RequireJS предоставляет механизм объявления и загрузки модулей, базирующийся на концепции AMD, и позволяющий избежать перечисленных выше недостатков.

Принципы работы механизма загрузчика RequireJS:

- Объявление модуля выполняется в функции `define()`, которая регистрирует функцию-фабрику для инстанцирования модуля, но при этом не загружает его в момент вызова.
- Зависимости модуля передаются как массив строковых значений, а не через свойства глобального объекта.
- Загрузчик выполняет загрузку модулей-зависимостей, переданных в качестве аргументов в функции `define()`. Модули загружаются асинхронно, при этом порядок их загрузки произвольно определяется загрузчиком.
- После загрузки указанных зависимостей модуля будет вызвана функция-фабрика, которая вернет значение модуля. При этом в нее в качестве аргументов будут переданы загруженные модули-зависимости.

Пример объявления модуля

Основы

Пример использования функции `define()` для объявления модуля `SumModule`

```
// Модуль с именем SumModule реализует функциональность суммирования двух чисел.
// SumModule не имеет зависимостей.
// Поэтому в качестве второго аргумента передается пустой массив, а
// анонимной функции-фабрике не передаются никакие параметры.
define("SumModule", [], function () {
```

```
    // Тело анонимной функции содержит внутреннюю реализацию функциональности модуля.
    var calculate = function (a, b) { return a + b; };
    // Возвращаемое функцией значение — объект, которым является модуль для системы.
    return {
        // Описание объекта. В данном случае модуль представляет собой объект со свойством summ.
        // Значение этого свойства — функция с двумя аргументами, которая возвращает сумму этих
        summ: calculate
    };
});
```

Функция define()

Основы

Назначение функции `define()` — объявление асинхронного модуля в исходном коде, с которым будет работать загрузчик.

Объявление модуля

```
define(
        ModuleName,
        [dependencies],
        function (dependencies) {
        }
);
```

Параметры

`ModuleName`

Строка с именем модуля. Необязательный параметр.

Если не указать параметр, загрузчик самостоятельно присвоит модулю имя в зависимости от его расположения в дереве скриптов приложения. Для обращения к модулю из других частей приложения (в том числе для асинхронной загрузки как зависимости другого модуля), имя модуля должно быть однозначно определено.

`dependencies`

Массив имен модулей, от которых зависит модуль. Необязательный параметр.

RequireJS выполняет асинхронную загрузку зависимостей, переданных в массиве. Порядок перечисления зависимостей в массиве `dependencies` должен соответствовать порядку перечисления параметров, передаваемых в фабричную функцию. Фабричная функция будет вызвана после загрузки зависимостей, перечисленных в `dependencies`.

`function(dependencies)`

Анонимная фабричная функция, которая инстанцирует модуль. Обязательный параметр.

В качестве параметров в функцию передаются объекты, которые ассоциируются загрузчиком с модулями–зависимостями, перечисленными в аргументе `dependencies`. Через эти аргументы осуществляется доступ к свойствам и методам модулей–зависимостей внутри создаваемого модуля. Порядок перечисления модулей в `dependencies` должен соответствовать порядку параметров фабричной функции.

Фабричная функция должна возвращать значение, которое загрузчик будет ассоциировать как экспортируемое значение создаваемого модуля.

Виды возвращаемого значения фабричной функции:

- Объект, которым является модуль для системы. Модуль сохраняется в кэше браузера после первичной загрузки на клиент. Если объявление модуля было изменено после загрузки на клиент (например, в процессе реализации конфигурационной логики), то необходимо очистить кэш и заново загрузить модуль.
- Функция-конструктор модуля. В качестве аргумента в конструктор передается объект контекста, в котором будет создаваться модуль. Загрузка модуля приведет к созданию на клиенте экземпляра модуля (**инстанцируемого модуля**). Повторная загрузка модуля на клиент функцией `require()` приведет к созданию еще одного экземпляра модуля. Эти экземпляры одного и того же модуля система будет воспринимать как два самостоятельных модуля. Примером объявления инстанцируемого модуля является модуль `CardModule` пакета `NUI`.