Marionette.Application.module

.start()Marionette предоставляет возможность описать модуль внутри приложения, включая подмодули, зависящие от этого модуля. Это полезно для создания сложных приложений, где каждый модуль инкапсулирует свои собственные свойства и методы, а их функциональность, благодаря модульной структуре приложения, вынесена в отдельные файлы.

Структура модулей в Marionette предоставляет возможность включать в приложение неограниченное количество подмодулей. Все они зависят от поведения приложения, и сами по себе являются агрегаторами событий.

Основы использования

Модуль задается непосредственно из объекта Приложения - для этого модулю назначается имя:

```
var MyApp = new Backbone.Marionette.Application();
var myModule = MyApp.module("MyModule");
MyApp.MyModule; // => новый Marionette.Application объект
myModule === MyApp.MyModule; // => true
```

Если вы попытаетесь задать модуль с одним и тем же именем более одного раза, первый экземпляр будет зафиксирован в приложении и сохранен в памяти, а остальные просто не будут созданы.

Запуск и остановка модулей

Модули могут быть запущены/остановлены независимым образом: инициатором запуска/остановки может являться как само приложение, так и другой модуль. Это позволяет им независимо и асинхронно загружаться в приложение, и также дает возможность выгрузить их в один прекрасный момент, когда они больше не используются. Такой подход, помимо прочего, значительно облегчает тестирование: каждый модуль может быть запущен и протестирован изолированно.

Запуск модулей

Запуск модуля может быть произведен одним из двух способов:

1. Автоматически из внешнего модуля (или из Приложения), когда будет вызван метод `.start()` внешнего модуля (Приложения)
2. Вручную, с помощью вызова метода `.start()` самого модуля.

В этом примере модуль будет запущен автоматически, когда на объекте родительского приложения выполнится его метод `start`:

```
MyApp = new Backbone.Marionette.Application();
MyApp.module("Foo", function(){
 // здесь идет код модуля
});
MyApp.start();
```

Следует заметить, что модули, загруженные и определенные даже после вызова `app.start()` все равно будут запускаться автоматически.

События, связанные со стартом модуля

Во время запуска модуля произойдут два события: "before:start" и "start". До всяких действий по инициализации будет запущено событие

"before:start". Само событие "start" будет запущено только после инициализация модуля.

```
var mod = MyApp.module("MyMod");

mod.on("before:start", function(){
  // делаем что-то до старта модуля
});

mod.on("start", function(){
  // и что-то после того, как модуль запущен
});
```

Передача данных в события запуска

Метод `.start` принимает один-единственный параметр - `options`, который будет передан start-событиям и их функциональным эквивалентам (`onStart` and `onBeforeStart`.)

```
var mod = MyApp.module("MyMod");

mod.on("before:start", function(options){
  // делаем что-то до старта модуля
});

mod.on("start", function(options){
  // и что-то после того, как модуль запущен
});

var options = {
 // какие-то данные
};
mod.start(options);
```

Предотвращение автозапуска модулей

Если вы предполагаете запускать модуль вручную, вместо того, чтобы предоставить этот процесс приложению, вы можете явным образом указать в параметрах определения модуля:

```
var fooModule = MyApp.module("Foo", function(){

  // не запускать родителем
  this.startWithParent = false;

  // ... код модуля идет здесь
});

// запуск приложения
MyApp.start();

// и позже, запуск модуля
fooModule.start();
```

Обратите внимание на использование объектного литерала вместо просто функции для определения модуля, и присутствие аттрибута `startWithParent`, сообщающего модулю, что он не должен запускаться вместе с запуском приложения. Теперь, чтобы запустить модуль, метод модуля `start` долен быть вызван вручную.

Вы также можете получить ссылку на модуль в какой-то более поздний момент времени, чтобы произвести запуск:

```
MyApp.module("Foo", function(){
  this.startWithParent = false;
});

// получаем ссылку - и запускаем модуль
MyApp.module("Foo").start();
```

Задание **`startWithParent: false`** в объектном литерале

Второй способ задания директивы `startWithParent` в методе `.module` - это использование объектного литерала:

```
var fooModule = MyApp.module("Foo", { startWithParent: false });
```

Это наиболее удобный способ, когда у нас есть громоздкий модуль, для описания которого требуется не один файл, а несколько, и мы используем отдельное описане для задания директивы `startWithParent`.

Если вы хотите объединить объектный литерал, описывающий `startWithParent`, с дифиницией модуля, вы можете включить аттрибут `define` в объектный литерал, переданный в функцию создания модуля:

```
var fooModule = MyApp.module("Foo", {
  startWithParent: false,

  define: function(){
    // здесь идет код модуля
  }
});
```

Запуск подмодулей их родителем

Запуск подмодулей совершается в порядке, обратном порядку иерархии - то-есть из глубины. Например, в иерархии `Foo.Bar.Baz` снач ала будет запущен `Baz`-модуль, потом `Bar`, и, наконец, `Foo`.

По умолчанию, запуск подмодуля производится с запуском его родителя.

```
MyApp.module("Foo", function(){...});
MyApp.module("Foo.Bar", function(){...});

MyApp.start();
```

В этом примере модуль "Foo.Bar" будет запущен, когда метод `MyApp.start()` будет вызван, поскольку его модуль-родитель "Foo" обязан запуститься одновременно со стартом приложения.

Но, как уже должно быть ясно, это поведение в подмодуле может быть переопределено, если значение директивы `startWithParent` за дано как false. Это предохранит его от запуска в момент вызова метода `start` его родителя.

```
MyApp.module("Foo", function(){...});

MyApp.module("Foo.Bar", function(){
  this.startWithParent = false;
})

MyApp.start();
```

Теперь модуль "Foo" будет запущен, а подмодуль "Foo.Bar" - нет.

Подмодуль, тем не менее, может быть запущен вручную:

```
MyApp.module("Foo.Bar").start();
```

Выключение модулей

Модуль может быть остановлен, или "выключен", чтобы очистить занимаемые ресурсы, когда модуль больше не нужен. Подобно запуску, остановка происходит в порядке обратном порядку иерархии, "из глубины". Так, в иерархии модулей`Foo.Bar.Baz` сначала

будет остановлен `Baz`-модуль, потом `Bar`, и, наконец, `Foo`.

Чтобы выключить модуль и все его подмодули, достаточно вызвать метод модуля `stop()`.

```
MyApp.module("Foo").stop();
```

Модули не выключаются автоматически приложением. Если вы хотите остановить какой-либо модуль, вам нужно явно вызвать его метод `stop`. Исключение составляет то обстоятельство, что с выключением внешнего модуля будут выключены и все его подмодули.

```
MyApp.module("Foo.Bar.Baz");

MyApp.module("Foo").stop();
```

Вызов `stop` в данном случае повлечет выключение `Bar` и `Baz`, поскольку они являются подмодулями `Foo`. (Для более подробной информации по определению подмодулей см. раздел "Определение подмодулей при помощи .(dot) нотации").

События при остановке модуля

Когда модуль останавливается, прежде чем какие-либо "завершители" (finalizers) будут вызваны, разошлется специальное событие "before:stop". И, по аналогии со "start"-событием, событие "stop" будет разослано, после того, как сработают "завершители".

```
var mod = MyApp.module("MyMod");

mod.on("before:stop", function(){
  // делаем что-то прежде чем модуль остановится
});

mod.on("stop", function(){
  // делаем что-то после того как модуль остановился
});
```

Определение подмодулей с помощью .(dot) нотации

Подмодули, или потомки модуля, в один прием могут быть заданы в качестве иерархии модулей и подмодулей:

```
MyApp.module("Parent.Child.GrandChild");

MyApp.Parent; // => абсолютно валидный модуль
MyApp.Parent.Child; // => то же самое, абсолютно валидный модуль
MyApp.Parent.Child.GrandChild; // => то же самое, абсолютно валидный модуль
```

Таким образом, в момент задания подмодулей с помощью dot-нотации, родительские модули не обязательно должны уже существовать. Они будут автоматически созданы, если их еще нет, - если же они существуют, то будут задействованы именно они - новых же создано не будет.

Определение модуля

Чтобы определить модуль, вы можете задать функцию обратного вызова.

Данная функция принимает 6 различных параметров:

- сам данный модуль
- модуль-родитель, или объект приложения, вызывающий данный модуль
- Backbone
- Backbone.Marionette
- jQuery
- Underscore
- любые пользовательские аргументы

Вы можете также добавлять функции и данные напрямую в ваш модуль, чтобы сделать их доступными извне. Также в модуле могут использоваться данные и функции, определенные локально.

```
MyApp.module("MyModule", function(MyModule, MyApp, Backbone, Marionette, $, _){
 // Приватные свойства и методы
 // ---------------------------
 var myData = "this is private data";

 var myFunction = function(){
  console.log(myData);
 }

 // Публичные свойства и методы
 // ---------------------------
 MyModule.someData = "public data";
 MyModule.someFunction = function(){
  console.log(MyModule.someData);
 }
});
console.log(MyApp.MyModule.someData); //=> публичное свойство
MyApp.MyModule.someFunction(); //=> публичный метод
```

Инициализаторы модуля (module initializers)

Модули снабжены инициализаторами, подобно объекту Приложения. Инициализаторы модуля - это те функции, которые выполнятся, когда модуль будет запущен.

```
MyApp.module("Foo", function(Foo){

  Foo.addInitializer(function(){
    // код, который должен выполниться при запуске
  });

});
```

Любой способ запуска данного модуля заставит сработать его инициализатор. Модуль может иметь сколько угодно инициализаторов.

"Завершители" модуля (module finalizers)

Модуль также включает в свой состав "завершители" - т.е. функции, которые будут вызваны, когда модуль останавливается.

```
MyApp.module("Foo", function(Foo){
 Foo.addFinalizer(function(){
  //код, делающий все, что нужно сделать при выключении модуля
 });
});
```

Вызов метода `stop` на конкретном модуле приведет к вызову всех "завершителей" данного модуля. Модуль может иметь сколько угодно "завершителей".

Аргумент **`this`** модуля

Аргумент модуля `this` указывает на сам этот модуль.

```
MyApp.module("Foo", function(Foo){
  this === Foo; //=> true
});
```

Аргументы, определенные пользователем

Вы можете передать модулю любое количество пользовательских аргументов, сделать это при необходимости можно после определяющей модуль функции, как указано в следующем примере.
Это позволит вам импортировать в модуль сторонние библиотеки и другие ресурсы. После импорта они будут доступны внутри модуля, в виде тех локальных переменных, которые вы передадите в качестве аргументов в определяющую модуль функцию:

```
MyApp.module("MyModule", function(MyModule, MyApp, Backbone, Marionette, $, _, Lib1, Lib2, LibEtc){
 // Lib1 === LibraryNumber1;
 // Lib2 === LibraryNumber2;
 // LibEtc === LibraryNumberEtc;
}, LibraryNumber1, LibraryNumber2, LibraryNumberEtc);
```

Разбиение описания модуля на отдельные части

Иногда модуль становится слишком большим, и для его описания становится удобнее разбить модуль на отдельные фрагменты, которые представлены в ряде файлов:

```
MyApp.module("MyModule", function(MyModule){
MyModule.definition1 = true;
});
MyApp.module("MyModule", function(MyModule){
MyModule.definition2 = true;
});
MyApp.MyModule.definition1; //=> true
MyApp.MyModule.definition2; //=> true
```