Marionette.Application

Объект `Backbone.Marionette.Application` является композиционным центром вашего приложения. Он организует, инициализирует и координирует различные части приложения. Он также воплощает в себе отправную точку для прямых вызовов кода, разбросанного по javascript-файлам, если вы предпочитаете такой стиль программирования. (но это плохой стиль: Marionette вполне поддерживает событийную архитектуру - прим. пер)

Добавление инициализаторов

Приложение предполагает выполнение ряда полезных действий, таких как отображение контента в регионах, обеспечения маршрутизации и т.д. Для достижения этой цели, а также для того, чтобы убедиться, что объект `Application` настроен как следует, вы можете добавить инициализирующие колбэки.

```
MyApp.addInitializer(function(options){
 //здесь делаем полезные вещи
 var myView = new MyView({
  model: options.someModel
 });
 MyApp.mainRegion.show(myView);
});
MyApp.addInitializer(function(options){
 new MyAppRouter();
 Backbone.history.start();
});
```

Данные колбэки будут выполнены как только приложение будет запущено, и связаны с объектом приложения как контекстом для их выполнения.
Другими словами, ключевое слово `this` внутри инициализирующей функции является не чем иным, как объектом `MyApp`.

Параметры `options` попадают в них из метода `start` (рассматривается ниже).

Инициализирующие колбэки гарантированно выполнятся, вне зависимости от того, в какой момент времени вы добавите их в объект приложения.
Если вы добавите их перед тем, как запустить приложение, они выполнятся, когда метод `start` будет вызван.
Если добавить их после запуска, они будут выполнены немедленно.

События объекта приложения

Объект `Application` порождает некоторые события в течение своего жизненного цикла, при помощи [Marionette.triggerMethod](./marionette.functions.md).
Эти события могут быть использованы для дополнительного процессинга вашего приложения. Например, вы можете предварительно обработать какие-то данные
перед тем, как произойдет инициализация объекта приложения. Или, например, вы хотите запустить `Backbone.history` только после того, как приложение будет до конца проинициализировано.

Вот эти события:

* **"initialize:before" / `onInitializeBefore`**: порождается до того, как инициализаторы приступят к действию

* **"initialize:after" / `onInitializeAfter`**: порождается сразу после того, как инициализаторы завершат работу
* **"start" / `onStart`**: порождается сразу после того, как инициализаторы завершат работу и будут порождены все инициализирующие события

```
MyApp.on("initialize:before", function(options){
 options.moreData = "Yo dawg, I heard you like options so I put some options in your options!"
});

MyApp.on("initialize:after", function(options){
 if (Backbone.history){
  Backbone.history.start();
 }
});
```

Параметр `options` попадет в колбэки из метода `start` объекта приложения (см. ниже).

Запуск приложения

Как только ваше приложение должным образом сконфигурировано, вы можете запустить его простым вызовом `MyApp.start(options)`

Данный метод принимает единственный опциональный параметр. Этот параметр будет передан во все инициализирующие колбэки, а также во все инициализирующие события. Благодаря этому обеспечивается дополнительная конфигурация для различных частей приложения.

```
var options = {
 something: "some value",
 another: "#some-selector"
};
MyApp.start(options);
```

app.vent: агрегатор событий

Каждый экземпляр приложения приходит в комплекте с экземпляром `Backbone.Wreqr.EventAggregator`, так называемым `app.vent`.

```
MyApp = new Backbone.Marionette.Application();
MyApp.vent.on("foo", function(){
 alert("bar");
});
MyApp.vent.trigger("foo"); //появится алерт "foo"
```

За дальнейшей информацией по `Backbone.Wreqr` обратитесь к документации.

Регионы и объект приложения

Объекты типа `Region` могут быть непосредственно добавлены в приложение посредствои вызова метода `addRegions`.

Предусмотрены три синтаксические формы для добавления региона в объект приложения.

jQuery селектор

Первый предполагает задание jQuery-селектора в качестве значения дифиниции региона.
Так будет непосредственно создан экземпляр Marionette.Region, и назначен селектору:

```
MyApp.addRegions({
 someRegion: "#some-div",
 anotherRegion: "#another-div"
});
```

Пользовательский тип региона

Согласно второму, определяется пользовательский тип региона, в котором селектор уже зарезервирован:

```
MyCustomRegion = Marionette.Region.extend({
 el: "#foo"
});

MyApp.addRegions({
 someRegion: MyCustomRegion
});
```

Пользовательский тип региона и селектор

Третий подразумевает одновременное задание пользовательского типа региона и jQuery селоктора для экземпляра региона -

используется объектный литерал:

```
MyCustomRegion = Marionette.Region.extend({});
MyApp.addRegions({
  someRegion: {
   selector: "#foo",
   regionType: MyCustomRegion
  },
  anotherRegion: {
   selector: "#bar",
   regionType: MyCustomRegion
  }
});
```

Удаление регионов

Регионы могут также быть удалены с помощью метода `removeRegion`, которому передается в качестве строкового значения имя региона.

```
MyApp.removeRegion('someRegion');
```

Удаление региона повлечет его корректное закрытие.