# MontageJS加载过程

**备注:** "mopped"和'mop'相关，[Monrage优化工具](https://github.com/montagejs/mop)

MontageJS在初始化一些变量之后调用exports.initMontage方法，然后又会调用getPlatform方法，getPlatform会返回一个跟平台相关的对象（已经支持浏览器和node.js使用方式）

## 浏览器

### 开发模式（没有进行代码打包优化的模式）

#### platform.bootstrap

platform.bootstrap会被调用，它的回调函数会在初始化完成以后被调用。这时浏览器开始处理Montage进程，在标签中的'data-'属性被识别，montage.js被调用。一个被称作'resolve'的方法也会被创建，它会根据相对路径返回document作为应用的根。

接下来我们添加一个DOMContentLoaded的监听器，监听器的callbackIfReady方法在上述过程完成之后会被调用。

Montage需要另外的3个文件来完成进一步的加载：

* 使用来自[Mr](https://github.com/montagejs/mr)的require.js 和 browser.js实现浏览器CommonJS模块的支持。
* q.js - 对promises的支持

它们是通过脚本表签声明的方式注入的，它们在完成加载后会被导出成模块，同事全局的bootstrap的方法会被调用，调用参数是它们的id。Mopped：如果代码已经通过Mop进行打包优化，这些文件会直接在项目初始化时被加载，所以就不在需要脚本标签注入。

全局的bootstrap方法会一直监视这3个文件的加载情况，一旦它们三个都被加载完成allModulesLoaded方法会被调用.

allModulesLoaded使用一个简版的require模块bootRequire来加载promisse以及导入其他的模块。最终callbackIfReady会被调用。

callbackIfReady检查DOM和模块是否都全部加载完成，如果是就调用这个方法的回调函数。

### 回调

首先我们根据Montage的配置加载Montage的各种包，这表示需要加载器直接读取.reel文件(比如require("montage/ui/text.reel"))，接下来编译器会将Montage的元数据附加到被读取的模块中（SerializationCompiler），然后将读取的HTML内容作为内容(TemplateCompiler)

[如果是进过mop打包优化的，Montage的加载过程是这样的。](http://docs.montagestudio.com/montagejs/bootstrapping.html#mopped-callback)

接下来我们使用Require来加载Montage包，当这个过程完成以后我们就可以用使用Montage的require方法montageRequire。我们用它来加载Q（Promise）包来获得完整的Promise包的内容。之后我们把已经加载完成的promise模块放入其中，避免重复加载。我们还会用到linter，当加载的文件中有语法错误时，它会给我们清晰的错误报告。

加载到这个步骤的时候可以有一段远端的代码控制Montage引导程序，比如用来测试。这里我们就不讲诉了。

如果我们使用data-auto-package属性定义了配置包的描述。那么package.json就不需要了，否则我们会检查本地是否有这个json文件（通过data-pacakge属性定义），如果存在就使用他。最后，我们开始加载应用程序包。

当上述过程完成以后，montageRequire和appliationRequire就可以使用了，加下来我们在initMontage方法中完成Montage的最终初始化。

#### initMontage

现在我们需要加载Montage最后需要的依赖，当这些依赖被家在完成之后我们需要配置应用，这意味着设置堆栈长度(设置为0最优)，设置事件管理器以及调用montageWillLoad方法。

接下来我们检查package.json是否定义了一个应用原型，没有的话使用“core/application”。application实例中的\_load方法会被调用，它用来加载Montage的组件和模版并让模板中的序列划定义被解析。

最后，我们检查data-module属性是否被定义，如果有的话，加载这些模块。

Montage的加载过程到这就完成了，被加载的模版和模块已经开始正常工作。

### 产品模式（使用mop优化打包的应用）

When Mopped the bootstrapping bundle defines a global BUNDLE array, which contains a list of bundle filenames to load.

当Mop时，会生成一个包含所有需要加载的包的数组，名为BUNDLE。

#### platform.bootstrap

和普通初始化过程相比，3个初始化需要的文件不再需要从脚本标签注入，它们已经存在与初始化包中。

#### 回调

在读取Montage各种包之前，BUNDEL变量会被查看，如果它存在，那么它描述的文件都会被生成脚本标签注入。每个bundle会传入自己的名字作为参数调用全局的bundleLoaded方法。一个名叫preloaded的promise会被执行，当所有包加载完成以后promise会做resolved. Mr会在promise完成以后才开始下一步的处理，这意味着Montage的各种包会等到所有bundle加载完成之后才会开始被加载。

上述就是开发模式和产品模式Montage初始化的区别

[查看开发模式下的初始化过程](http://docs.montagestudio.com/montagejs/bootstrapping.html#un-mopped-load-montage)

## Node.js

TBD