Atech Computers

Development Reference Manual

Versions

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Brief** | **Author** | **Date** |
| 1.0.0 | Initial version | Xiaochen Li | 17/4/2016 |
|  |  |  |  |
|  |  |  |  |

1. Introduction

Atech Computers website is built on ReactionCommerce, which is an open source platform based on Meteor framework. The platform provides web developers the functionalities that are necessary for all kinds of e-commerce business applications, such as account, product and order management, routing helper tools, and so on.

ReactionCommerce is a Meteor project, and it inherits all the features of Meteor. Meteor is a full-stack JavaScript platform for developing modern web and mobile applications. Meteor includes a key set of technologies for building connected-client reactive applications, a build tool, and a curated set of packages from the Node.js and general JavaScript community.

1. Project Structure

ReactionCommerce is built based on Meteor framework, so it follows its way of structuring its project. This section will introduce some of the important folders that are needed to be cared about.

***2.1 meteor***

Meteor facility folder in every Meteor project, including the information of Meteor and the project, such as Meteor version, all the packages and their versions supported by Meteor, and the collection of packages defined by the current project.

In addition, once we have built a package via build command, cache files for the product package will also be stored under .meteor/local folder.

***2.2 client***

Any directory named client/ is not loaded on the server. Similar to wrapping your code in if (Meteor.isClient) { ...}. All files loaded on the client are automatically concatenated and minified when in production mode. In development mode, JavaScript and CSS files are not minified, to make debugging easier. CSS files are still combined into a single file for consistency between production and development, because changing the CSS file’s URL affects how URLs in it are processed.

***2.3 packages***

It collects project-defined Atmosphere packages. Reaction functionalities are all defined as Atmosphere packages, their names starting with “reaction-”. Most Reaction packages can be obtained via its GitHub repository https://github.com/reactioncommerce/reaction.git. But payment-related packages have not been moved to this repository and they must be pulled from their own repository using “Reaction pull” command.

Packages are managed and built with the help of package.js file, which collects all the package information, and manages dependencies and versions. All files in the same package are combined into one JavaScript source file during the building process, in the appropriate order which all the dependencies are handled correctly.

***2.4 private***

All files inside a top-level directory called private/ are only accessible from server code and can be loaded via the [Assets](http://docs.meteor.com/#/full/assets_getText) API. This can be used for private data files and any files that are in your project directory that you don’t want to be accessible from the outside.

***2.5 public***

All files inside a top-level directory called public/ are served as-is to the client. When referencing these assets, do not include public/ in the URL, write the URL as if they were all in the top level. For example, reference public/bg.png as <img src='/bg.png' /> (be careful of the first ‘/’ meaning starting with root path). This is the best place for favicon.ico, robots.txt, and similar files.

***2.6 server***

Any directory named server/ is not loaded on the client. Similar to wrapping your code in if (Meteor.isServer) { ... }, except the client never even receives the code. Any sensitive code that you don’t want served to the client, such as code containing passwords or authentication mechanisms, should be kept in the server/ directory.

Meteor gathers all your JavaScript files, excluding anything under the client, public, and private subdirectories, and loads them into a Node.js server instance. In Meteor, your server code runs in a single thread per request, not in the asynchronous callback style typical of Node.

***2.7 test***

Any directory named tests/ is not loaded anywhere. Use this for any test code you want to run using a test runner outside of [Meteor’s built-in test tools](http://guide.meteor.com/testing.html).

***2.8 bin***

This folder includes Reaction scripts, such as run, install, reset and pull, all of which are called when running the “Reaction-like” commands.

***2.9 docker***

It includes scripts for manipulating Docker images. Since the project may be deployed to Docker engine hosts, so the tools are every useful to help us build Docker images.

1. Routing

Reaction implements the routing functionality provided by [kadira:flow-router](https://github.com/kadirahq/flow-router) for routing in the reaction-router package. It is a carefully designed client side router for Meteor applications.

Once imported as the dependency, flow router becomes available via the global variable called “FlowRouter”. The Reaction router, exported as “ReactionRouter”, references the “FlowRouter” variable.

***3.1 Carefully designed client side router for Meteor***

FlowRouter is a very simple router for Meteor. It does routing for client-side apps, but it does not handle rendering itself.

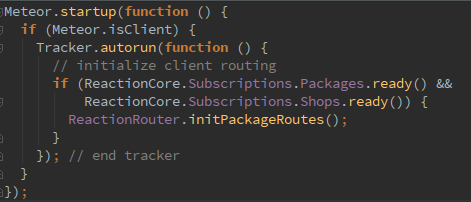
It exposes a great API for changing the URL and reactively getting data from the URL. However, inside the router, it's not reactive. Most importantly, FlowRouter is designed with performance in mind and it focuses on what it does best: routing.

***3.2 How Reaction implements routing***

Reaction-router package is responsible for initialization of ReactionRouter utility class, which is exposed as a global variable of the same name “ReactionRouter” that can be accessible throughout the application. But it should be noticed that most the helper functions, such as go and redirect, are only built to client side. Any route defined on server side will not function in the way it is supposed to.

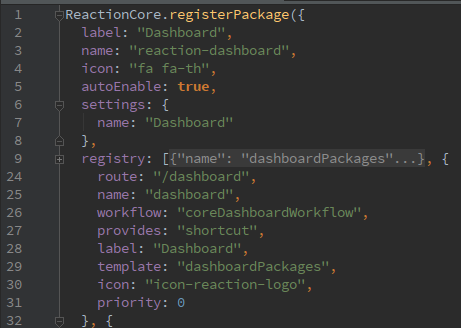
This manual will not involve much of its usage (please refer to the official website for further details <https://github.com/kadirahq/flow-router>). Instead, it will explain at code level how Reaction defines routes in “reaction-router” package and how it can be modified and extended, as well as some points which developers find interesting and worth attentions during their work with the project.

The router initialization starts with client router calling ReactionRouter to initialize package routes at Meteor startup (common/router.js).

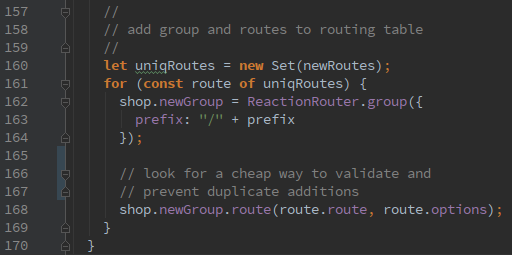


Note that Reaction only initializes Reaction-related package routes on client side. All Reaction routes are defined in Reaction Registry in “package-name/server/registry.js”.

Let’s take dashboard as an example. Dashboard defines its route as “/dashboard”, and then router initialization will search all the Reaction packages, extract the route from the registry and add the new route to the router.

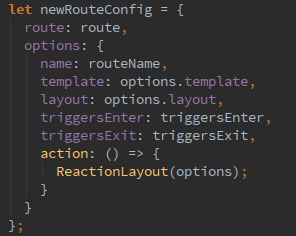


In addition, Reaction supports multi shops, which is reflected in router that all routes are organized into route groups (supported by FlowRouter) and use their corresponding shop name as path prefixes. However, we only manage one shop for our project, and this prefix seems to be redundant. It can be removed by setting the route group prefix to empty.



***3.3 How Reaction deals with rendering***

As mentioned above, FlowRouter does not handle rendering. Rather, it provides a callback for other rendering mechanisms, which gives developers flexibilities to choose the one they like. In render options, an action function allows us to call any rendering function.



Reaction, by default, uses Blaze-Layout package (released with FlowRouter, in the snapshot above, it is called in ReactionLayout function). It facilitates page rendering by only passing several template names and avoiding developers involving too much in the Blaze rendering, because Blaze render method requires DOM and template instances as arguments.

Meteor applications are single-page web applications, which will also be explained in perspectives in the proceeding sections of this manual. Each route only renders one template per page (templates will be explained in next section), and there is not necessary to register sub templates which will be automatically dealt with by Blaze (one template rendering mechanism in meteor).

1. Templates

Blaze is a powerful built in template library of Meteor. It provides the same live-updating capability as Angular and React, but it is much easier to use.

Templates are defined in <template name=”my\_template\_name”> tag, with its name being exclusive in the entire application scope. They can also have sub templates.

During the start-up of the web application, templates as html files are loaded first. During that time, all templates are loaded into JavaScript objects, and from then on they are no longer html pages.

HTML files in a Meteor application are treated quite a bit differently from a server-side framework. Meteor scans all the HTML files in your directory for three top-level elements: <head>, <body>, and <template>. The head and body sections in all the html files are separately concatenated into a single head and body, which are transmitted to the client on initial page load. In this aspect, defining head and body sections in more than one html files is not necessary, and potentially, it leads to confusions.

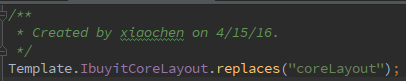
Helpers are variables and functions that belongs to a certain template. They are usually defined in JavaScript files that have the same names to their html files, and are loaded after all the templates being loaded.

Details of how to use Blaze libraries and space bars can be found at the official website https://www.meteor.com/blaze.

* 1. ***Template replacement***

In order to keep the original functionalities, and in the meantime, customize the page design in a way that will not affect the Reaction updating process (Reaction is officially updated once a week), it is best to replace the built-in templates in /client directory.

Reaction includes the “[meteor-template-extension](https://github.com/aldeed/meteor-template-extension)” package which provides functionality for extending Meteor templates. With the help of this package, replacing an existing template is very simple in just one line of JavaScript code. For example, add a template extension map to client/templates.js:



As the code shows, a customized template called “IbuyitCoreLayout” replaces the Reaction built-in template called “coreLayout”. However, in this case, helpers, events, etc, that defined for “IbuyitCoreLayout” will not be available. Reaction still uses those defined for “coreLayout”.

In the Meteor framework, all html templates are translated to JavaScript objects. In the implementation of replaces() function, it replaces the “coreLayout” object with the “IbuyitCoreLayout” object, and nothing else changes.

It is suggested that developers mirror the existing structure. For example, if you are extending the template located in

packages/reaction-core/client/templates/products/products.html

then you would mirror that by creating your new template at

reaction/client/templates/products/myTemplate.html.

Build & Deployment

Test

Appendix

Useful JavaScript features (Including ECMAScript6)

# References

[1] ReactionCommerce doc, <https://docs.reactioncommerce.com/>

[2] FlowRouter, <https://github.com/kadirahq/flow-router>

[3] Blaze, <https://www.meteor.com/blaze>