

# Реферат по JavaScript-фреймворку Meteor

А. А. Андреев, 22609

24 сентября 2016 г.

## Введение

Трендом последних нескольких лет в Web-разработке являются приложения реального времени и их разновидность — реактивные приложения. Реактивность подразумевает мгновенное обновление данных на клиентской стороне при обновлении их на сервере (без необходимости обновлять страницу). Часто такие приложения выполняются в виде одностраничных приложений, т.е. переходы между ресурсами приложения выполняются без загрузки новой страницы, чаще всего с помощью асинхронной загрузки данных с сервера.

Закономерно, для облегчения создания таких приложений появился ряд программных каркасов (фреймворков) для языка JavaScript, как основного языка клиентской Web-Разработки. Некоторые из них предоставляют уровень абстракции только для уровня представления (например, ReactJS), другие предлагают вариацию паттерна MVC (BackboneJS, AngularJS) и опираются на серверную сторону через REST- или JSON-интерфейс.

Еще одна вариация фреймворк-архитектуры предлагается Meteor (MeteorJS). Этот программный каркас предлагает full stack JavaScript разработку с NodeJS на серверной стороне. В данном реферате рассмат-

риваются подробности архитектуры фреймворка Meteor, основы и тонкости создания приложений на его основе (на примере приложения для размещения новостей).

## 1. Архитектура MeteorJS

Meteor предлагает использование одного языка – JavaScript – и на клиентской стороне (web-обозреватель, web-view на мобильных платформах), и на серверной стороне (NodeJS). При этом все API (в том числе и к базе данных) – нативные для этого языка.

Для общения между двумя сторонами в Meteor используется собственный протокол – Distributed Data Protocol (DDP). Он подразумевает общение между двумя сторонами посредством JSON-объектов и пропагандирует модель публикация-подписка, что позволяет асинхронно получать данные при обновлении их на сервере без необходимости постоянного опроса сервера. В рамках протокола создается постоянное соединение между клиентской и серверной стороной и сервер посылает обновленные данные без инициации обмена клиентом.

Кроме того, Meteor поддерживает и стандартный для других фреймворков обмен данными посредством HTTP-сообщений.

На серверной стороне Meteor, несмотря на занятие только одного порта, работает сразу два web-сервера: сервер DDP (на основе SockJS и технологии Web-сокетов), который обеспечивает реактивное взаимодействие, и сервер HTTP (на основе API NodeJS), который обеспечивает передачу статических файлов и обработку классических HTTP-запросов.

В качестве СУБД Meteor предлагает использовать MongoDB, для чего имеется собственный API, интегрированный с другими компонентами фреймворка (например, протоколом DDP). Путем установки расширений

Meteor позволяет организовать взаимодействие с другими СУБД, такими как PostgreSQL.

Фреймворком предоставляется возможность установки пакетов расширений с помощью собственного менеджера пакетов Atmosphere.

## **2. Основы разработки с помощью Meteor на примере новостного приложения**

Для обозначения основных моментов и тонкостей разработки приложений с помощью Meteor в данном реферате будет описан процесс разработки приложения для размещения Новостей. Приложение будет предоставлять возможность размещения новостей (заголовок + текст) авторизованными пользователями и чтения новостей всеми пользователями.

### **2.1. Установка Meteor**

Дистрибутив фреймворка включает в себя не только файлы самого фреймворка, в него интегрированы NodeJS и MongoDB, что облегчает первоначальную настройку приложения (однако усложняет установку приложений на production-сервер в плане гибкости настроек). После установки дистрибутива, соответствующего целевой операционной системе, из командной строки будет доступна команда “\$ meteor create имя\_приложения”, с помощью которой происходит инициализация начальной структуры нового приложения.

Рассматриваемый фреймворк имеет также собственный менеджер пакетов по названию Atmosphere с соответствующим репозиторием пакетов (см. [?]). Установка пакетов осуществляется с помощью команды “\$ meteor add имя\_пакета”, а удалить с помощью

“\$ meteor remove имя\_пакета”.

## 2.2. Файловая структура приложений

После инициализации приложения будет создана следующая файловая структура:

- `package.json` — файл, описывающий приложение для npm;
- `server/main.js` — файл, являющийся точкой входа для всего server-side кода;
- `client/main.js` — файл, являющийся точкой входа для всего client-side кода;
- `client/main.html` — файл с описанием представления приложения, включая шаблоны;
- `client/main.css` — основной файл стилей приложения;
- `.meteor` — каталог со служебными файлами фреймворка, включая список зависимостей, настройки приложения и т.п.;
- `.meteor/local` — каталог со служебными локальными файлами приложения, включая файлы базы данных, установленные пакеты расширений, скомпилированные js-файлы.

В Meteor поощряется использование директив `import/export` из спецификации ES2015. Так, создателям фреймворка предлагается разделять приложение на небольшие модули и импортировать их друг из друга. Исходя из данных рекомендаций, в Новостном приложении будет использоваться следующая дополнительная файловая структура:

- `imports` — каталог с модулями приложения;

- `imports/startup/{client и server}` — код, который должен выполниться при старте приложения на клиентской и на серверной стороне соответственно;
- `imports/api` — модули, описывающие предметную область приложения и вспомогательные функции;
- `imports/api/accounts-config.js` — инициализация и настройка системы аккаунтов пользователей;
- `imports/ui` — модули уровня представления, включая шаблоны (`.html`) и инициализирующий и вспомогательный код для шаблонов (`.js`).

Все соответствующие файлы из каталога “startup” (и других, которые требуются) должны быть импортированы в “client/main.js” и “server/main.js”.

## 2.3. Работа с базой данных

В качестве СУБД по умолчанию Meteor использует MongoDB [?]. Данные хранятся в виде коллекций документов произвольной структуры. Доступ к БД осуществляется через объект “Mongo”, который подключается следующим образом:

```
import { Mongo } from 'meteor/mongo';
```

Документы БД в Meteor представлены, как объекты JavaScript. Работа с коллекциями осуществляется следующим образом:

```
// создание
var coll = new Mongo.Collection('collection_name');
```

```
// получение всех документов
coll.find();

// запрос по документам
coll.find({someProperty: someVal});

// получение одного найденного документа
coll.findOne({someProperty: someVal});

// вставка документа
coll.insert({someProperty: someVal});

// удаление всех удовлетворяющих элементов
coll.remove({someProperty: someVal});
```

Подробнее см. общий API MongoDB. Заметим одну особенность Meteor: коллекции по умолчанию доступны и на стороне сервера, и на стороне клиента. При этом, при создании коллекции на стороне сервера данная коллекция будет создана в БД, а при создании коллекции на стороне клиента будет создано кеширующее подключение к коллекции на сервере. На клиенте при этом по умолчанию будут доступны все операции CRUD.

MongoDB является NoSQL базой данных и данные в ней по умолчанию не валидируются. Для включения проверки данных в соответствии с некоторой схемой данных можно использовать собственный (сложный!) механизм MongoDB или использовать валидацию на стороне приложения перед вставкой данных. Для Meteor для этого есть, например, пакет `aldeed:simple-schema`.

Определение схемы для коллекции осуществляется следующим образом:

```
import { SimpleSchema } from 'meteor/aldeed:simple-schema';

coll.schema = new SimpleSchema({...});
```

Включение автоматической валидации документов при операциях insert и update можно осуществить следующим образом:

```
coll.attachSchema(coll.schema);
```

При работе с некорректными данными будет выброшено `ValidationError`.

Для рассматриваемого новостного приложения спроектируем единственную требуемую коллекцию — коллекцию новостей. Meteor предлагает размещать файлы с описанием коллекций в “imports/api/”. В этом каталоге создадим файл “News.js” с описанием коллекции новостей. Схема будет следующая:

```
News.schema = new SimpleSchema({
// идентификатор новости
  _id: { type: String, regEx: SimpleSchema.RegEx.Id },
  // заголовок новости
  title: { type: String },
  // основной новостной текст новости (в виде HTML)
  text: { type: String },
  // дата создания новости
  date: { type: Date },
  // имя пользователя, который создал новость
  username : { type: String }
});
```

Чтобы сделать коллекцию доступной в других модулях приложения, в верхней части файла добавим:

```
export const News = new Mongo.Collection('News');
```

## 2.4. Роутинг

Meteor ориентирован на создание одностраничных приложений, для которых не нужен роутинг. Однако, для поддержки некоторых возможностей обработка адресной строки и перенаправления между страницами все таки необходимо. В нашем приложении предполагается существование отдельной страницы для каждой новости и уникального адреса для них (чтобы доступ к отдельной новости можно было бы легко получить извне, например для индексации в поисковых системах).

Одним из доступных роутеров (которые распространяются в виде пакетов) является “kadira:flow-router”.

Для определения маршрута, который будет обрабатываться приложением (на стороне клиента!), можно использовать следующую конструкцию:

```
import { FlowRouter } from 'meteor/kadira:flow-router';

FlowRouter.route('/some/path/:someVariable', {
  name: 'SomeName.show',
  action(params, queryParams) {
    // какие-либо действия
    console.log('some log');
  }
});
```

В данном случае, при совпадении запрашиваемого адреса с определенным адресом будут выполнены действия, определенные в методе “action”. Параметры адресного запроса указываются начиная с двоеточия. Объект “params” будет содержать имена и значения определенных параметров ад-



ресной строки, а объект “queryParams” — параметров GET запроса (если есть).

Для нашего приложения потребуются два маршрута. Маршрут для корня приложения (адрес '/') и маршрут для отдельной новости (адрес '/post/:\_id').

```
FlowRouter.route('/post/:_id', {  
  name: 'Post.show',  
  action() {  
    ...  
  }  
});
```

```
FlowRouter.route('/', {  
  name: 'main.show',  
  action() {  
    ...  
  }  
});
```

Предполагается, что файл с описанием маршрутов будет размещен в “imports/startup/client”, а модуль, описываемый файлом, будет импортирован “client/main.js”.

## 2.5. Аутентификация и авторизация

Для работы с учетными записями пользователей Meteor предлагает использовать набор пакетов, среди которых “accounts-ui” для предоставления интерфейса для доступа к аккаунту (например, форма аутентифи-

кации) и “accounts-password” для авторизации с помощью пароля (альтернативой могут быть OAuth и данные внешних сервисов, таких как Facebook).

Для настройки механизма авторизации необходимо предлагается создать файл в “imports/startup”. Файл нашего приложения будет содержать только две инструкции: одна для подключения механизма авторизации, другая для включения использования имени пользователя вместо e-mail.

```
import { Accounts } from 'meteor/accounts-base';

Accounts.ui.config({
  passwordSignupFields: 'USERNAME_ONLY',
});
```

Модуль нужно импортировать из “client/main.js”.

В приложении данные о пользователе будут доступны в виде коллекции MongoDB Meteor.users, функций “Meteor.userId()” для получения идентификатора текущего пользователя, “Meteor.user()” для получения объекта, представляющего текущего пользователя. Из представлений доступен объект “currentUser”.

## 2.6. Шаблоны представлений

...

## 2.7. Дальнейшая работа

...

## **Заключение**

...

## **Список литературы**