19 Интеграция со сторонними API

Успешные сайты все чаще не полностью автономны. Для того чтобы заинтересовать существующих и найти новых пользователей, интеграция с социальными сетями обязательна. Для указания местонахождения магазинов и мест оказания прочих услуг очень большое значение имеет использование сервисов геолокации и отображения на карте. И на этом не останавливаются: все больше организаций осознают, что предоставление API помогает расширить перечень услуг и сделать их более полезными.

В этой главе мы обсудим две наиболее актуальные потребности интеграции: социальные медиа и геолокацию.

Социальные медиа

Социальные медиа — это отличный способ продвинуть ваш продукт или сервис: возможность для пользователей делиться вашим контентом в социальных медиа очень важна. Когда я пишу эту главу, доминирующие сервисы социальных сетей — Facebook и Twitter. Google+ может усиленно бороться за свою долю, но не следует совсем уж не брать его в расчет: в конце концов, он работает при поддержке одной из крупнейших и опытнейших интернет-компаний в мире. У сайтов вроде Pinterest, Instagram и Flickr есть свое место, но у них, как правило, небольшая и более специфическая аудитория (например, если ваш сайт связан с созданием изделий способом «сделай сам», то вы наверняка захотите поддерживать Pinterest). Смейтесь, если хотите, но я предсказываю, что MySpace еще вернется. Они замечательно «передизайнили» сайт, и, что стоит отметить, он сделан на Node.

Плагины социальных медиа и производительность сайта

Большая часть интеграции социальных медиа — это дело клиентской части. Вы ссылаетесь на соответствующие файлы JavaScript на вашей странице, и это включает как входящий контент (например, три верхних поста с вашей страницы

на Facebook), так и исходящий (например, можно сделать твит о странице, на которой вы находитесь). Зачастую это простой способ интеграции социальных медиа, но у него тоже есть своя цена: я видел, как страницы загружаются вдвое, а то и втрое дольше из-за дополнительных HTTP-запросов. Если производительность страницы для вас важна (а она должна быть важна, особенно если вы ориентируетесь на мобильных пользователей), нужно тщательно продумать, как интегрировать социальные медиа.

При этом код, который позволяет включить кнопку Facebook Нравится или кнопку Tweet, использует браузерные cookie, чтобы отправить сообщение от имени пользователя. Перемещение этого функционала на серверную сторону будет непростым, а в некоторых случаях и вовсе невозможным. Так что, если вам нужен именно этот функционал, наилучшим решением может быть подключение соответствующей сторонней библиотеки, даже несмотря на то, что это может повлиять на производительность страницы. Спасти ситуацию может то, что API-интерфейсы Facebook и Twitter весьма распространены: очень велика вероятность, что ваш браузер их уже закэшировал, а в этом случае они повлияют на производительность незначительно.

Поиск твитов

Предположим, мы хотим указать десять последних твитов, которые содержат хештег #meadowlarktravel. Мы могли бы использовать для этого компонент клиентской части, но он будет включать дополнительные запросы HTTP. Более того, если мы делаем это на серверной стороне, у нас появляется возможность кэширования твитов для повышения производительности. В этом случае мы также можем отправлять в черный список твиты недоброжелателей, тогда как на стороне клиента это сделать было бы существенно сложнее.

Twitter, как и Facebook, позволяет вам создать *приложения*. Это немного неподходящее название: приложение Twitter ничего не делает (в традиционном смысле). Это скорее набор учетных данных, которые вы можете использовать для создания реального приложения на вашем сайте. Простейший и наиболее универсальный способ получения доступа на Twitter API — создать приложение и использовать его для получения токена доступа.

Создайте приложение Twitter, зайдя на http://dev.twitter.com. Щелкните на значке в левом верхнем углу, затем выберите Мои приложения. Нажмите Создать новое приложение и следуйте инструкциям. Когда появится приложение, вы увидите, что у вас теперь есть *потребительский ключ* и *потребительский секрет*. Потребительский секрет, на что указывает название, должен хранить секрет: никогда не включайте его в ответы, отправляемые на сторону клиента. Если бы кто-то извне получил доступ к этому секрету, он мог бы создавать запросы от имени вашего

приложения, что могло бы иметь печальные последствия, если их использование вредоносное.

Теперь у нас есть потребительский ключ и потребительский секрет и мы можем общаться с Twitter REST API.

Чтобы сохранять код в аккуратном виде, поместим наш код Twitter в модуль, названный `lib/twitter.js`:

```
var https = require('https');

module.exports = function(twitterOptions){

    return {
        search: function(query, count, cb){
                // то, что нужно сделать
        }
    };
};
```

Этот шаблон наверняка начинает казаться вам знакомым. Наш модуль экспортирует функцию, в которую вызывающая сторона передает объект конфигурации. Возвращается объект, содержащий методы. Таким образом, мы можем добавить функционал в наш модуль. Сейчас мы предоставляем метод `search`. Вот как будем использовать библиотеку:

```
var twitter = require('./lib/twitter')({
        consumerKey: credentials.twitter.consumerKey,
        consumerSecret: credentials.twitter.consumerSecret,
});

twitter.search('#meadowlarktravel', 10, function(result){
        // твиты будут в result.statuses
});
```

(Не забывайте внести свойство `twitter` с `consumerKey` и `consumerSecret` в файл `credentials.js`.)

Прежде чем реализовать метод `search`, мы должны предоставить определенную функциональность для аутентификации самих себя в Twitter. Процесс прост: используем HTTPS для запроса токена доступа, базирующийся на нашем потребительском ключе и потребительском секрете. Мы должны сделать это только раз: токены у Twitter даются бессрочно (впрочем, можете аннулировать их вручную). Поскольку мы не хотим запрашивать токен доступа каждый раз, мы его закэшируем для дальнейшего использования.

Способ, посредством которого мы построили модуль, позволяет создать приватную функциональность, которая будет недоступна вызывающей стороне.

В частности, единственная функция, которая доступна вызывающей стороне, — это module.exports. Поскольку мы возвращаем функцию, вызывающей стороне будет доступна только она. Вызов функции дает в результате объект, и только свойства этого объекта доступны вызывающей стороне. Итак, мы собираемся создать переменную accessToken, которую будем использовать для кэширования нашего токена доступа, и функцию getAccessToken, которая этот токен получит. При первом запуске она создаст запрос Twitter API для получения токена доступа. Последующие вызовы просто возвращают значение accessToken:

```
var https = require('https');

module.exports = function(twitterOptions){

    // эта переменная будет невидимой вне этого модуля var accessToken;

    // эта функция будет невидимой за пределами этого модуля
    function getAccessToken(cb){
        if(accessToken) return cb(accessToken);
        // то, что нужно сделать: получение токена доступа
    }

    return {
        search: function(query, count, cb){
            // то, что нужно сделать
        },
    };
};
```

Поскольку getAccessToken может требовать асинхронный вызов к Twitter API, мы должны предоставить обратный вызов, который будет осуществляться, когда значение accessToken действительно. Теперь, когда мы установили базовую структуру, реализуем getAccessToken:

```
function getAccessToken(cb){
    if(accessToken) return cb(accessToken);

    var bearerToken = Buffer(
        encodeURIComponent(twitterOptions.consumerKey) + ':' +
        encodeURIComponent(twitterOptions.consumerSecret)
    ).toString('base64');

    var options = {
        hostname: 'api.twitter.com',
        port: 443,
        method: 'POST',
```

```
        path: '/oauth2/token?grant_type=client_credentials',
        headers: {
            'Authorization': 'Basic ' + bearerToken,
        },
    };

    https.request(options, function(res){
        var data = '';
        res.on('data', function(chunk){
            data += chunk;
        });
        res.on('end', function(){
            var auth = JSON.parse(data);
            if(auth.token_type!=='bearer') {
                console.log('Twitter auth failed.');
                return;
            }
            accessToken = auth.access_token;
            cb(accessToken);
        });
    }).end();
}
```

Подробности построения этого вызова доступны на странице документации для разработчиков Twitter для аутентификации «только для приложений». В принципе, мы должны сделать токен предъявителя (bearer token), что будет base64-кодированной комбинацией потребительского ключа и потребительского секрета. После того как мы сконструируем токен, можем вызвать `/oauth2/token` API с заголовком `Authorization`, содержащим токен предъявителя для запроса токена доступа. Обратите внимание на то, что мы должны использовать HTTPS: если вы попробуете сделать этот вызов посредством HTTP, то передадите свой секрет в незашифрованном виде, а API попросту отключит вас.

После того как получим полный ответ от API (мы ожидаем событие `end` потока ответа), мы можем разобрать JSON, убедиться, что тип токена `bearer`, и дальше идти своей дорогой. Мы кэшируем токен доступа, затем вызываем функцию обратного вызова.

Теперь, раз у нас есть механизм получения токена доступа, мы можем делать вызовы API.

Реализуем наш метод поиска:

```
search: function(query, count, cb){
    getAccessToken(function(accessToken){
        var options = {
            hostname: 'api.twitter.com',
```

```
                    port: 443,
                    method: 'GET',
                    path: '/1.1/search/tweets.json?q=' +
                          encodeURIComponent(query) +
                          '&count=' + (count || 10),
                    headers: {
                        'Authorization': 'Bearer ' + accessToken,
                    },
        };
        https.request(options, function(res){
            var data = '';
            res.on('data', function(chunk){
                data += chunk;
            });
            res.on('end', function(){
                cb(JSON.parse(data));
            });
        }).end();
    });
},
```

Отображение твитов

Теперь у нас есть возможность искать твиты — так как мы будем отображать их на нашем сайте? По большому счету, это остается на ваше усмотрение, но есть несколько вещей, которые мы должны рассмотреть. Twitter заинтересован в том, чтобы его данные использовали в соответствующем стиле. Для этого у него есть требования отображения, использующие функциональные элементы, которые вы должны включить для отображения в Twitter.

В требованиях возможны определенные маневры (например, если вы отображаете твиты на устройстве, которое не поддерживает изображения, нет надобности добавлять аватар), но по большей части конечный итог должен быть чем-то очень похожим на то, как выглядит встроенный твит. Это предполагает много работы, и есть способ ее сделать... Но он включает в себя ссылки на библиотеки виджетов Twitter, а это и есть тот самый HTTP-запрос, которого мы пытаемся избежать.

Если вам нужно отображать твиты, лучше всего использовать библиотеку виджетов Twitter, несмотря на то что это добавит дополнительный HTTP-запрос (опять же в силу вездесущности Twitter этот ресурс наверняка закэширован браузером, так что снижение производительности может оказаться незначительным). Для более сложного использования API вам нужен доступ к REST API с серверной стороны, так что вы в итоге наверняка будете использовать REST API в связке со скриптами в веб-интерфейсе.