

Опыт разработки виджетов для сторонних сайтов

Если ваш продукт предоставляет услуги для бизнеса, рано или поздно появится задача создать встраиваемый виджет для сайтов клиентов. Это может быть виджет покупки билетов, прогноза погоды, курса валют, отзывов, комментариев и много чего другого.

В этой статье разберемся, как же сделать качественный виджет, который можно будет легко поддерживать и расширять.

Библиотека

Не нужно думать, что виджет у вас будет один.

Даже если сейчас это так и других виджетов не предвидится. Или вы уверены, что один виджет может устанавливаться один раз на страницу.

Главная сложность разработки виджетов для сторонних сайтов — сразу верно заложить архитектуру так, чтобы при развитии виджетов не нужно было изменять код, установленный на сайтах. Убедить пользователей виджета заменить код довольно сложно, долго и вызывает волну негатива.

Поэтому сразу проектируем код таким образом, чтобы он позволял вставлять неограниченное число разных виджетов на одну страницу без ограничений. Первое, что приходит в голову: «а давайте просто выведем `iframe` с нашего сайта?». И сделаем код вида:

```
<iframe src="https://company.name/my-widget" frameborder="0" scrolling="no" width="300"
height="200">
  Ваш браузер не поддерживает фреймы!
</iframe>
```

И это будет большой ошибкой по нескольким причинам.

Невозможность расширения

У `iframe` довольно много ограничений, связанных с защитой конфиденциальности в браузере. Даже просто растянуть `iframe` под размер его содержимого без внешнего `javascript` кода не получится. Стилизовать можно будет только то, что лежит непосредственно в `iframe`, на сам тэг и его обертку никак нельзя будет повлиять.

Может, для начала это не критично, но по мере развития в это легко можно упереться, а код на сайтах уже установлен.

Про то, какие проблемы есть в работе с `iframe` и их решении, поговорим попозже.

Лишние запросы на бэкенд

Если на сайт будет установлено 5 таких одинаковых виджетов, то на ваш сервер придет 5 одинаковых запросов, хотя по факту нужен был только один. Конечно, можно сделать кеш на `nginx` и не пропускать запрос дальше, но зачем нам самим себе делать паразитные запросы? С таким кодом изменить логику получения данных не получится без изменения кода вставки виджета. Если виджет будет установлен на десятках сайтов, даже не самых популярных, в сумме они могут давать заметную нагрузку.

А как надо делать?

Если вы когда-нибудь устанавливали на сайт какой-нибудь виджет, например, от ВК, то могли заметить, что код виджета разделен на две части: библиотеку (SDK) и инициализацию виджетов:

Код для вставки:

```
<!-- Put this script tag to the <head> of your page
-->
<script type="text/javascript"
src="https://vk.com/js/api/openapi.js?168">
</script>

<script type="text/javascript">
  VK.init({apild: 4540930, onlyWidgets: true});
</script>

<!-- Put this div tag to the place, where the
Comments block will be -->
<div id="vk_comments"></div>
<script type="text/javascript">
VK.Widgets.Comments("vk_comments", {limit:
10, attach: "*"});
</script>
```

В целом, хорошая практика — посмотреть, как делают другие, так можно сразу перескочить через множество граблей. В этом коде все хорошо, кроме того, что он не асинхронный. Это их право — давать вставлять по умолчанию код, который может заблокировать загрузку страницы, но мы так делать не будем, попозже поговорим об этом. Нам в этом коде важна идея.

Мы видим, что для вставки любого виджета нужно один раз подключить SDK и добавить один пустой тэг с инициализацией виджета. А дальше все делает javascript: он может делать любые запросы и любое их количество на бэкенд, и разработчики виджета могут в любой момент эту логику изменить без изменения кода виджета на сайте. В итоге из html на сайте для виджета должен быть только пустой контейнер с уникальным id. Чтобы не томить вас ожиданием, давайте сразу напишем пример SDK и рендер простого виджета.

А потом поговорим о том, что же у нас получилось и на что стоит обратить внимание.

код библиотеки

```
namespace MyCompany {
  /**
   * Виджет кнопки
   */
  class Button {
    /**
     * Внутренний id кнопки
```

```

    */
    protected id: number;

    /**
     * DOM элемент контейнера
     */
    protected containerElement: HTMLElement;

    /**
     * Инстанс api
     */
    protected apiInstance: Api;

    /**
     * Constructor
     * @param {Api} instance
     * @param {string} containerId
     */
    public constructor(instance: Api, containerId: string) {
        this.apiInstance = instance;
        this.containerElement = document.getElementById(containerId);
    }

    /**
     * Инициализация
     */
    public init(): void {
        this.containerElement.innerHTML = '<button>Виджет кнопки</button>';
    }
}

/**
 * Основной класс Api
 */
export class Api {

    /**
     * Виджет кнопки
     * @param {string} containerId
     * @return {MyCompany.Button}

```

```

        */
    public button(containerId: string): Button {
        const widget = new Button(this, containerId);
        widget.init();
        return widget;
    }

    /**
     * Запуск колбеков инициализации
     */
    public runInitCallbacks(): void
    {
        let myCompanyApiInitCallbacks = (window as
any).myCompanyApiInitCallbacks;
        if (myCompanyApiInitCallbacks && myCompanyApiInitCallbacks.length)
        {
            setTimeout(function () {
                let callback;
                while (callback = myCompanyApiInitCallbacks.shift())
            {
                try {
                    callback();
                } catch (e) {
                    console.error(e);
                }
            }
            }, 0);
        }
    }
}

/**
 * Инициализация Api
 */
if (typeof (window as any)['myCompanyApi'] === 'undefined') {
    (window as any).myCompanyApi = new MyCompany.Api();
    (window as any).myCompanyApi.runInitCallbacks();
}

```

```

<!-- в head один раз -->
<script async type="text/javascript" src="https://mycompany.site/js/api/api.min.js"></script>

<!-- в место вызова виджета -->
<div id="button-container-5ef9b197c865f"></div>
<script type="text/javascript">
    (function() {
        var init = function() {
            myCompanyApi.button('button-container-5ef9b197c865f');
        };
        if (typeof myCompanyApi !== 'undefined') {
            init();
        } else {
            (myCompanyApiInitCallbacks = window.myCompanyApiInitCallbacks ||
            []).push(init);
        }
    })();
</script>

```

Выглядит страшно и как-то избыточно, давайте разбираться, зачем все это.

Асинхронность

Сразу в глаза бросается атрибут **async** у тэга `script`. Он позволит браузеру не ждать загрузки нашего скрипта и продолжить отрисовывать сайт. Это важно: если по каким-то причинам наш скрипт будет недоступен (недоступен сервер, фаервол компании), это не должно влиять на скорость загрузки сайта клиента. Но все не так просто. Раз скрипт загружается асинхронно, это значит, что когда браузер дойдет до места, где инициализируется наш виджет, наш SDK может быть еще не загружен, и если просто вызвать метод из библиотеки — будет ошибка, причем плавающая, в зависимости от того, успел загрузиться скрипт или нет.

Поэтому в месте вызова виджета мы должны обработать оба сценария, когда SDK загрузилось и еще нет.

В первом случае мы просто вызываем функцию `init()`. Во втором — откладываем выполнение этой функции до момента, когда скрипт загрузится, добавляя замыкание в очередь. А последней строчкой в нашем SDK вызывается метод **runInitCallbacks**, который как раз и выполнит все отложенные инициализации.

Тут же есть защита от повторного подключения SDK, ведь пользователи могут проигнорировать ваши требования и вставить скрипт библиотеки десять раз.

Теперь наш код запускается всегда и не блокирует отрисовку страницы!

Изоляция

Название объекта SDK и id контейнеров должны быть уникальными, ведь наш код будет выполняться на совершенно разных сайтах. Ни в коем случае нельзя нарваться на совпадения. ID контейнеров желательно генерировать уникальными, например, через `uniqid()`. Нельзя надеяться и на сторонние библиотеки, установленные на сайте, и совсем не желательно приносить их с собой. Да, я о jQuery, как вы уже догадались.

Код виджетов должен быть легковесным и универсальным, сейчас уже не сложно писать кроссбраузерный код нативно. Кроме того, я настоятельно рекомендую использовать TypeScript, но это есть множество причин.

На кодировку сайта тоже не стоит полагаться, и даже в наше время встречаются сайты на `cp1251`. Поэтому кодировку скрипта нужно явно задать в ответе сервера в заголовке **Content-Type**.

Код, написанный нами выше, позволяет не останавливаться на одном виджете: сейчас у нас есть только **`myCompanyApi.button()`**, но ничего не мешает добавить другие виджеты.

Кеширование

Мы будем постоянно дорабатывать наш SDK, но браузеры кэшируют скрипты, если разработчик не дал других инструкций. Мы должны сами задать время, на которое можно кешировать нашу библиотеку, через заголовок **Expires**, например, час — адекватное время. С кешированием на фронтенде разобрались, теперь поговорим про бэкенд. Как уже обсуждали выше, обслуживание запросов со сторонних виджетов может создавать ощутимую нагрузку просто от количества сайтов, где виджеты установлены. Но чаще всего данные для всех пользователей в этих виджетах одинаковые, нет смысла запускать приложение, а тем более ходить в базу данных за ними на каждый запрос. Такие запросы вообще дальше `nginx` можно не пропускать, настроив кеширование на нем.

Если для отрисовки виджета нужны данные с бэкенда, но в целом можно отрисовать минимальную версию и без него (например, кнопку покупки билетов, но без признака наличия), хорошим тоном будет сделать fallback: если данные не загрузились за полсекунды, рисовать обрезанную версию виджета, а как только данные получены — дорисовывать. Это визуально ускорит загрузку и покажет виджет даже без работающего бэкенда (вдруг он при взаимодействии уже поднимется?).

Немного про iframe

Iframe — по сути, отображение сайта в сайте. Вернемся к нашему кейсу с кнопкой покупки билетов. Если мы хотим при клике открывать попап со страницей выбора места — без iframe нам не обойтись. Какие же там есть нюансы?

Неработающие cookie

Уже давно многие браузеры по умолчанию начинают запрещать использование cookie для сторонних сайтов (это когда домен iframe отличается от родительского сайта). Это значит, что при переходе между страницами внутри фрейма не получится отследить сессию (localStorage тоже не работает). Тут выход простой — не перезагружать страницы и делать SPA. Идентификатор сессии можно будет легко сохранить в переменной в js.

Общение с SDK

Часто требуется организовать общение нашего SDK с приложением внутри iframe, например, мы хотим при открытии виджета растянуть размер фрейма под размер контента. Для этого нам нужно сообщить размер контента из iframe в родительское окно. Это можно легко сделать через [postMessage](#). Будьте внимательны при передаче конфиденциальных данных и верно указывайте **targetOrigin**, иначе данные могут «подслушать» другие сайты.

Спасибо за внимание, надеюсь, вы узнали для себя что-то новое.

Сергей Никитченко, Студия Валерия Комягина

Версия #1

Seryak создал Fri, Apr 2, 2021 11:33 AM

Seryak обновил Fri, Apr 2, 2021 11:36 AM