Общение между окнами

Политика «Одинакового источника» (Same Origin) ограничивает доступ окон и фреймов друг к другу.

Идея заключается в том, что если у пользователя открыто две страницы: `john-smith.com` и `gmail.com`, то у скрипта со страницы `john-smith.com` не будет возможности прочитать письма из `gmail.com`. Таким образом, задача политики «Одинакового источника» – защитить данные пользователя от возможной кражи.

Политика "Одинакового источника"

Два URL имеют «одинаковый источник» в том случае, если они имеют совпадающие протокол, домен и порт.

Эти URL имеют одинаковый источник:

- `http://site.com`
- `http://site.com/`
- `http://site.com/my/page.html`

А эти – разные источники:

- `http://www.site.com` (другой домен: `www.` важен)
- `http://site.org` (другой домен: `.org` важен)
- `https://site.com` (другой протокол: `https`)
- `http://site.com:8080` (другой порт: `8080`)

Политика «Одинакового источника» говорит, что:

- если у нас есть ссылка на другой объект `window`, например, на всплывающее окно, созданное с помощью `window.open` или на `window` из `<iframe>` и у этого окна тот же источник, то к нему будет полный доступ.
- в противном случае, если у него другой источник, мы не сможем обращаться к его переменным, объекту `document` и так далее. Единственное исключение – объект `location`: его можно изменять (таким образом перенаправляя пользователя). Но нельзя читать `location` (нельзя узнать, где находится пользователь, чтобы не было никаких утечек информации).

Доступ к содержимому ифрейма

Внутри `<iframe>` находится по сути отдельное окно с собственными объектами `document` и `window`.

Мы можем обращаться к ним, используя свойства:

- `iframe.contentWindow` ссылка на объект `window` внутри `<iframe>`.
- `iframe.contentDocument` – ссылка на объект `document` внутри `<iframe>`, короткая запись для `iframe.contentWindow.document`.

Когда мы обращаемся к встроенному в ифрейм окну, браузер проверяет, имеет ли ифрейм тот же источник. Если это не так, тогда доступ будет запрещён (разрешена лишь запись в `location`, это исключение).

Для примера давайте попробуем чтение и запись в ифрейм с другим источником:

```
<iframe src="https://example.com" id="iframe"></iframe>

<script>
  iframe.onload = function() {
    // можно получить ссылку на внутренний window
```


```
  let iframeWindow = iframe.contentWindow; // OK
  try {
    // ...но не на document внутри него
    let doc = iframe.contentDocument; // ОШИБКА
  } catch(e) {
    alert(e); // Security Error
  }

  // также мы не можем прочитать URL страницы в ифрей
  try {
    // Нельзя читать из объекта Location
    let href = iframe.contentWindow.location.href; //
  } catch(e) {
    alert(e); // Security Error
  }

  // ...но можно писать в него (и загрузить что-то др
  iframe.contentWindow.location = '/'; // OK

  iframe.onload = null; // уберём обработчик, чтобы н
};
</script>
```

Код выше выведет ошибку для любых операций, кроме:

- Получения ссылки на внутренний объект `window` из `iframe.contentWindow`
- Изменения `location`.

С другой стороны, если у ифрейма тот же источник, то с ним можно делать всё, что угодно:

```
<!-- ифрейм с того же сайта -->
<iframe src="/" id="iframe"></iframe>

<script>
  iframe.onload = function() {
    // делаем с ним что угодно
    iframe.contentDocument.body.prepend("Привет, мир!")
  };
</script>
```

`iframe.onload` и `iframe.contentWindow.onload`

Событие `iframe.onload` – по сути то же, что и `iframe.contentWindow.onload`. Оно сработает, когда встроенное окно полностью загрузится со всеми ресурсами.

...Но `iframe.onload` всегда доступно извне ифрейма, в то время как доступ к `iframe.contentWindow.onload` разрешён только из окна с тем же источником.

Окна на поддоменах: document.domain

По определению, если у двух URL разный домен, то у них разный источник.

Но если в окнах открыты страницы с поддоменов одного домена 2-го уровня, например `john.site.com`, `peter.site.com` и `site.com` (так что их общий домен `site.com`), то можно заставить браузер игнорировать это отличие. Так что браузер сможет считать их пришедшими с одного источника при проверке возможности доступа друг к другу.

Для этого в каждом таком окне нужно запустить:

```
document.domain = 'site.com';
```

После этого они смогут взаимодействовать без ограничений. Ещё раз заметим, что это доступно только для страниц с одинаковым доменом второго уровня.

Ифрейм: подождите документ

Когда ифрейм – с того же источника, мы имеем доступ к документу в нём. Но есть подвох. Не связанный с кросс-доменными особенностями, но достаточно важный, чтобы о нём знать.

Когда ифрейм создан, в нём сразу есть документ. Но этот документ – другой, не тот, который в него будет загружен!

Так что если мы тут же сделаем что-то с этим документом, то наши изменения, скорее всего, пропадут.

Вот, взгляните:

```
<iframe src="/" id="iframe"></iframe>

<script>
  let oldDoc = iframe.contentDocument;
  iframe.onload = function() {
    let newDoc = iframe.contentDocument;
    // загруженный document - не тот, который был в ifr
    alert(oldDoc == newDoc); // false
  };
</script>
```

Нам не следует работать с документом ещё не загруженного ифрейма, так как это не тот документ. Если мы поставим на него обработчики событий – они будут проигнорированы.

Как поймать момент, когда появится правильный документ?

Можно проверять через `setInterval`:

```
<iframe src="/" id="iframe"></iframe>

<script>
  let oldDoc = iframe.contentDocument;

  // каждый 100 мс проверяем, не изменился ли документ
  let timer = setInterval(() => {
    let newDoc = iframe.contentDocument;
    if (newDoc == oldDoc) return;

    alert("New document is here!");

    clearInterval(timer); // отключим setInterval, он н
  }, 100);
</script>
```

Коллекция window.frames

Другой способ получить объект `window` из `<iframe>` – забрать его из именованной коллекции `window.frames`:

- По номеру: `window.frames[0]` – объект `window` для первого фрейма в документе.
- По имени: `window.frames.iframeName` – объект `window` для фрейма со свойством `name="iframeName"`.

Например:

```
<iframe src="/" style="height:80px" name="win" id="ifra

<script>
  alert(iframe.contentWindow == frames[0]); // true
```

```
5    alert(iframe.contentWindow == frames.win); // true
6  </script>
```


Ифрейм может иметь другие ифреймы внутри. Таким образом, объекты `window` создают иерархию.

Навигация по ним выглядит так:

- `window.frames` – коллекция «дочерних» `window` (для вложенных фреймов).
- `window.parent` – ссылка на «родительский» (внешний) `window`.
- `window.top` – ссылка на самого верхнего родителя.

Например:

```
1  window.frames[0].parent === window; // true
```

Можно использовать свойство `top`, чтобы проверять, открыт ли текущий документ внутри ифрейма или нет:

```
1  if (window == top) { // текущий window == window.top?
2    alert('Скрипт находится в самом верхнем объекте windo
3  } else {
4    alert('Скрипт запущен во фрейме!');
5  }
```

Атрибут ифрейма sandbox

Атрибут `sandbox` позволяет наложить ограничения на действия внутри `<iframe>`, чтобы предотвратить выполнение ненадёжного кода. Атрибут помещает ифрейм в «песочницу», отмечая его как имеющий другой источник и/или накладывая на него дополнительные ограничения.

Существует список «по умолчанию» ограничений, которые накладываются на `<iframe sandbox src="...">`. Их можно уменьшить, если указать в атрибуте список исключений (специальными ключевыми словами), которые не нужно применять, например: `<iframe sandbox="allow-forms allow-popups">`.

Другими словами, если у атрибута `"sandbox"` нет значения, то браузер применяет максимум ограничений, но через пробел можно указать те из них, которые мы не хотим применять.

Вот список ограничений:

`allow-same-origin`

`"sandbox"` принудительно устанавливает «другой источник» для ифрейма. Другими словами, он заставляет браузер воспринимать `iframe`, как пришедший из другого источника, даже если `src` содержит тот же сайт. Со всеми сопутствующими ограничениями для скриптов. Эта опция отключает это ограничение.

`allow-top-navigation`

Позволяет ифрейму менять `parent.location`.

`allow-forms`

Позволяет отправлять формы из ифрейма.

`allow-scripts`

Позволяет запускать скрипты из ифрейма.

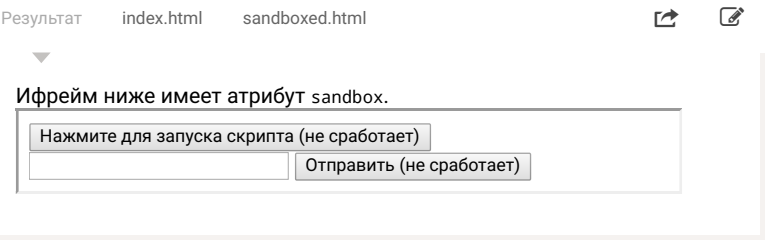
`allow-popups`

Позволяет открывать всплывающие окна из ифрейма с помощью `window.open`.

Больше опций можно найти в справочнике.

Пример ниже демонстрирует ифрейм, помещённый в песочницу со стандартным набором ограничений: `<iframe sandbox src="...">`. На странице содержится JavaScript и форма.

Обратите внимание, что ничего не работает. Таким образом, набор ограничений по умолчанию очень строгий:



На заметку:

Атрибут `"sandbox"` создан только для того, чтобы добавлять ограничения. Он не может удалять их. В частности, он не может ослабить ограничения, накладываемые браузером на ифрейм, приходящий с другого источника.

Обмен сообщениями между окнами

Интерфейс `postMessage` позволяет окнам общаться между собой независимо от их происхождения.

Это способ обойти политику «Одинакового источника». Он позволяет обмениваться информацией, скажем `john-smith.com` и `gmail.com`, но только в том случае, если оба сайта согласны и вызывают соответствующие JavaScript-функции. Это делает общение безопасным для пользователя.

Интерфейс имеет две части.

postMessage

Окно, которое хочет отправить сообщение, должно вызвать метод postMessage окна получателя. Другими словами, если мы хотим отправить сообщение в окно `win`, тогда нам следует вызвать `win.postMessage(data, targetOrigin)`.

Аргументы:

`data`

Данные для отправки. Может быть любым объектом, данные клонируются с использованием «алгоритма структурированного клонирования». IE поддерживает только строки, поэтому мы должны использовать метод `JSON.stringify` на сложных объектах, чтобы поддержать этот браузер.

`targetOrigin`

Определяет источник для окна-получателя, только окно с данного источника имеет право получить сообщение.

Указание `targetOrigin` является мерой безопасности. Как мы помним, если окно (получатель) происходит из другого источника, мы из окна-отправителя не можем прочитать его `location`. Таким образом, мы не можем быть уверены, какой сайт открыт в заданном окне прямо сейчас: пользователь мог перейти куда-то, окно-отправитель не может это знать.

Если указать `targetOrigin`, то мы можем быть уверены, что окно получит данные только в том случае, если в нём правильный сайт. Особенно это важно, если данные конфиденциальные.

Например, здесь `win` получит сообщения только в том случае, если в нём открыт документ из источника `http://example.com`:

```
<iframe src="http://example.com" name="example">

```


```
<script>
  let win = window.frames.example;

  win.postMessage("message", "http://example.com");
</script>
```

Если мы не хотим проверять, то в `targetOrigin` можно указать `*`.

```
<iframe src="http://example.com" name="example">

<script>
  let win = window.frames.example;

  win.postMessage("message", "*");
</script>
```

Событие message

Чтобы получать сообщения, окно-получатель должно иметь обработчик события `message` (сообщение). Оно срабатывает, когда был вызван метод `postMessage` (и проверка `targetOrigin` пройдена успешно).

Объект события имеет специфичные свойства:

`data`

Данные из `postMessage`.

`origin`

Источник отправителя, например, `http://javascript.info`.

`source`

Ссылка на окно-отправитель. Можно сразу отправить что-то в ответ, вызвав `source.postMessage(...)`.

Чтобы добавить обработчик, следует использовать метод `addEventListener`, короткий синтаксис `window.onmessage` не работает.

Вот пример:

```
window.addEventListener("message", function(event) {
  if (event.origin != 'http://javascript.info') {
    // что-то пришло с неизвестного домена. Давайте про
    return;
  }

  alert( "received: " + event.data );

  // can message back using event.source.postMessage(..
});
```

Полный пример:

Результат iframe.html index.html

Введите сообщение | Нажмите для отправки

Получение ифрейма.

Без задержек

Между `postMessage` и событием `message` не существует задержки. Событие происходит синхронно, быстрее, чем `setTimeout(...,0)`.

Итого

Чтобы вызвать метод или получить содержимое из другого окна, нам, во-первых, необходимо иметь ссылку на него.

Для всплывающих окон (попапов) доступны ссылки в обе стороны:

- При открытии окна: `window.open` открывает новое окно и возвращает ссылку на него,
- Изнутри открытого окна: `window.opener` – ссылка на открывающее окно.

Для ифреймов мы можем иметь доступ к родителям/потомкам, используя:

- `window.frames` – коллекция объектов `window` вложенных ифреймов,
- `window.parent`, `window.top` – это ссылки на родительское окно и окно самого верхнего уровня,
- `iframe.contentWindow` – это объект `window` внутри тега `<iframe>`.

Если окна имеют одинаковый источник (протокол, домен, порт), то они могут делать друг с другом всё, что угодно.

В противном случае возможны только следующие действия:

- Изменение свойства location другого окна (доступ только на запись).
- Отправить туда сообщение.

Исключения:

- Окна, которые имеют общий домен второго уровня: `a.site.com` и `b.site.com`. Установка свойства `document.domain='site.com'` в обоих окнах переведёт их в состояние «Одинакового источника».
- Если у ифрейма установлен атрибут `sandbox`, это принудительно переведёт окна в состояние «разных источников», если не установить в атрибут значение `allow-same-origin`. Это можно использовать для запуска ненадёжного кода в ифрейме с того же сайта.

Метод `postMessage` позволяет общаться двум окнам с любыми источниками:

1. Отправитель вызывает `targetWin.postMessage(data, targetOrigin)`.
2. Если `targetOrigin` не `'*'`, тогда браузер проверяет имеет ли `targetWin` источник `targetOrigin`.
3. Если это так, тогда `targetWin` вызывает событие `message` со специальными свойствами:
   - `origin` – источник окна отправителя (например, `http://my.site.com`)
   - `source` – ссылка на окно отправитель.
   - `data` – данные, может быть объектом везде, кроме IE (в IE только строки).

   В окне-получателе следует добавить обработчик для этого события с помощью метода `addEventListener`.

Комментарии

перед тем как писать...