Окна и фреймы

Открытие окон и методы window

Всплывающие окна используются нечасто. Ведь загрузить новую информацию можно динамически, а показать – в элементе , расположенным над страницей (z-index). Ещё одна альтернатива – тег <iframe>.
Если мы открываем попап, хорошей практикой будет предупредить пользователя об этом. Иконка открывающегося окошка на ссылке поможет посетителю понять, что происходит и не потерять оба окна из поля зрения.

В прошлом злонамеренные сайты заваливали посетителей всплывающими окнами. Такие страницы могли открывать сотни попапов с рекламой. Поэтому теперь большинство браузеров пытаются заблокировать всплывающие окна, чтобы защитить пользователя.

Всплывающее окно **блокируется** в том случае, если вызов window.open произошёл не в результате действия посетителя (например, события onclick). Что, если попап должен открываться в результате onclick, но не сразу, а только после выполнения setTimeout? Здесь все не так-то просто. Firefox «допускает» таймаут в 2000 мс или менее, но все, что свыше этого – не вызывает его доверия, т.к. предполагается, что в таком случае открытие окна происходит без ведома пользователя.

Синтаксис открытия нового окна:

window.open(url, name, params):

где:

- **url** - URL для загрузки в новом окне.
- **name** -Имя нового окна. У каждого окна есть свойство window.name, в котором можно задавать, какое окно использовать для попапа. Таким образом, если уже существует окно с заданным именем – указанный в параметрах URL откроется в нем, в противном случае откроется новое окно.
- **params** -Строка параметров для нового окна. Содержит настройки, разделённые запятыми. Важно помнить, что в данной строке **не должно быть пробелов**. Например width=200,height=100.
  Параметры в строке params:
  - Позиция окна:
    - **left/top** (числа) – координаты верхнего левого угла нового окна на экране. Существует ограничение: новое окно не может быть позиционировано вне видимой области экрана.
    - **width/height** (числа) – ширина и высота нового окна. Существуют ограничения на минимальную высоту и ширину, которые делают невозможным создание невидимого окна.
  - Панели окна:
    - **menubar** (yes/no) – позволяет отобразить или скрыть меню браузера в новом окне.
    - **toolbar** (yes/no) – позволяет отобразить или скрыть панель навигации браузера (кнопки вперёд, назад, перезагрузки страницы) нового окна.
    - **location** (yes/no) – позволяет отобразить или скрыть адресную строку нового окна. Firefox и IE не позволяют скрывать эту панель по умолчанию.
    - **status** (yes/no) – позволяет отобразить или скрыть строку состояния. Как и с адресной строкой, большинство браузеров будут принудительно показывать её.
    - **resizable** (yes/no) – позволяет отключить возможность изменения размера нового окна. Не рекомендуется.
    - **scrollbars** (yes/no) – позволяет отключить полосы прокрутки для нового окна. Не рекомендуется.
    -

  Правила для опущенных параметров:
  - Если третий аргумент при вызове open отсутствует или он пустой, будут использованы настройки окна по умолчанию.
  - Если строка параметров передана, но некоторые параметры yes/no пропущены, то считается, что указано no, так что соответствующие возможности будут отключены, если на это нет ограничений со стороны браузера. Поэтому при задании параметров убедитесь, что вы явно указали все необходимые yes.
  - Если координаты `left/top` не заданы, браузер попытается открыть новое окно рядом с предыдущим открытым окном.
  - Если не заданы размеры окна `width/height`, браузер откроет новое окно с теми же размерами, что и предыдущее открытое окно.

Вызов open возвращает ссылку на новое окно. Эта ссылка может быть использована для управления свойствами окна, например, изменения положения и др.

Здесь содержимое окна модифицируется после загрузки:

```
let newWindow = open('/', 'example', 'width=300,height=300')
newWindow.focus();
alert(newWindow.location.href); // (*) about:blank, загрузка ещё не началась
newWindow.onload = function() {
  let html = `<div style="font-size:30px">Добро пожаловать!</div>`;
  newWindow.document.body.insertAdjacentHTML('afterbegin', html);
};
```

Политика одного источника

Политика «Одинакового источника» (Same Origin) ограничивает доступ окон и фреймов друг к другу. Окна имеют свободный доступ к содержимому друг друга только если они с одного источника (у них совпадают домен, протокол и порт (**protocol://domain:port -** www. важен, https важен). Иначе, например, если основное окно с site.com, а попап с gmail.com, это невозможно по соображениям пользовательской безопасности.

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).

Попап также может обратиться к открывшему его окну по ссылке **window.opener**. Она равна null для всех окон, кроме попапов.

Чтобы закрыть окно**: win.close()**

Для проверки, закрыто ли окно: **win.closed.**

Технически метод **close()** доступен для любого окна, но window.close() будет игнорироваться большинством браузеров, если window не было создано с помощью window.open(). Так что он сработает только для попапов.

Прокрутка и изменение размеров

win.moveBy(x,y) - Переместить окно относительно текущей позиции на x пикселей вправо и y пикселей вниз. Допустимы отрицательные значения (для перемещения окна влево и вверх).

win.moveTo(x,y) - Переместить окно на координаты экрана (x,y).

win.resizeBy(width,height) -Изменить размер окна на указанные значения width/height относительно текущего размера. Допустимы отрицательные значения.

win.resizeTo(width,height) - Изменить размер окна до указанных значений.

Также существует событие **window.onresize**.

Чтобы предотвратить возможные злоупотребления, браузер обычно блокирует эти методы. Они гарантированно работают только с попапами, которые мы открыли сами и у которых нет дополнительных вкладок.

Методами JavaScript нельзя свернуть или развернуть («максимизировать») **окно на весь экран**. За это отвечают функции уровня операционной системы, и они скрыты от фронтенд-разработчиков.

Методы перемещения и изменения размера окна не работают для **свернутых и развёрнутых на весь экран окон**.

Прокрутка окна

win.scrollBy(x,y) -Прокрутить окно на x пикселей вправо и y пикселей вниз относительно текущей прокрутки. Допустимы отрицательные значения.

win.scrollTo(x,y) - Прокрутить окно до заданных координат (x,y).

elem.scrollIntoView(top = true) - Прокрутить окно так, чтобы elem для elem.scrollIntoView(false) появился вверху (по умолчанию) или внизу.

Также существует событие **window.onscroll**.

Установка и потеря фокуса

Теоретически, установить попап в фокус можно с помощью **метода window.focus(),** а убрать из фокуса – с помощью **window.blur().** Также существуют **события focus/blur**, которые позволяют отследить, когда фокус переводится на какое-то другое окно.

Раньше на «плохих» сайтах эти методы могли становиться средством манипуляции. Например:

```
window.onblur = () => window.focus();
```

Когда пользователь пытается перевести фокус на другое окно, этот код возвращает фокус назад. Таким образом, фокус как бы «блокируется» в попапе, который не нужен пользователю. Из-за этого в браузерах и появились ограничения, которые препятствуют такого рода поведению фокуса. Эти ограничения нужны для защиты пользователя от назойливой рекламы и «плохих» страниц, и их работа различается в зависимости от конкретного браузера. Например, мобильный браузер обычно полностью игнорирует такие вызовы метода window.focus(). Также фокусировка не работает, когда попап открыт в отдельной вкладке (в отличие от открытия в отдельном окне).

Но все-таки иногда методы фокусировки бывают полезны. Например:
Когда мы **открываем попап**, может быть хорошей идеей запустить для него **newWindow.focus().** Для некоторых комбинаций браузера и операционной системы это устранит неоднозначность – заметит ли пользователь это новое окно.
Если нужно отследить, **когда посетитель использует веб-приложение**, можно отслеживать **window.onfocus/onblur**. Это позволит ставить на паузу и продолжать выполнение анимаций и других интерактивных действий на странице. При этом важно помнить, что **blur** означает, что окно больше не в фокусе, но пользователь может по-прежнему видеть его.

<iframe>

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).

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!

```
<iframe src="/" id="iframe"></iframe>
<script>
  let oldDoc = iframe.contentDocument;
  iframe.onload = function() {
    let newDoc = iframe.contentDocument;
    // загруженный document - не тот, который был в iframe при создании изначально!
    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 нам больше не нужен
  }, 100);
</script>
```

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

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".

```
<iframe src="/" style="height:80px" name="win" id="iframe"></iframe>
<script>
  alert(iframe.contentWindow == frames[0]); // true
  alert(iframe.contentWindow == frames.win); // true
</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.

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-popup** - Позволяет открывать всплывающие окна из ифрейма с помощью window.open.

Больше опций можно найти в справочнике.

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

postMessage

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.

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>
```

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.

```
window.addEventListener("message", function(event) {
 if (event.origin != 'http://javascript.info') {
  // что-то пришло с неизвестного домена. Давайте проигнорируем это
  return;
 }
 alert( "received: " + event.data );
 // can message back using event.source.postMessage(...)
});
```

Таким образом:
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**.