**Тег «script»**

Программы на JavaScript могут быть вставлены в любое место HTML-документа с помощью тега <script>.

Для примера:

<!DOCTYPE HTML>

<html>

<body>

<p>Перед скриптом...</p>

**<script>**

**alert( 'Привет, мир!' );**

**</script>**

<p>...После скрипта.</p>

</body>

</html>

Тег <script> содержит JavaScript-код, который автоматически выполнится, когда браузер его обработает.

**Современная разметка**

Тег <script> имеет несколько атрибутов, которые редко используются, но всё ещё могут встретиться в старом коде:

Атрибут type: <script type=…>

Старый стандарт HTML, HTML4, требовал наличия этого атрибута в теге <script>. Обычно он имел значение type="text/javascript". На текущий момент этого больше не требуется.

Атрибут language: <script language=…>

Этот атрибут должен был задавать язык, на котором написан скрипт. Но так как JavaScript является языком по умолчанию, в этом атрибуте уже нет необходимости.

Обёртывание скрипта в HTML-комментарии.

В очень древних книгах и руководствах вы сможете найти комментарии внутри тега <script>, например, такие:

<script type="text/javascript"><!--

...

//--></script>

Этот комментарий скрывал код JavaScript в старых браузерах, которые не знали, как обрабатывать тег <script>. Поскольку все браузеры, выпущенные за последние 15 лет, не содержат данной проблемы, такие комментарии уже не нужны. Если они есть, то это признак, что перед нами очень древний код.

**Внешние скрипты**

Если у вас много JavaScript-кода, вы можете поместить его в отдельный файл.

Файл скрипта можно подключить к HTML с помощью атрибута src:

<script src="/path/to/script.js"></script>

Здесь /path/to/script.js – это абсолютный путь до скрипта от корня сайта. Также можно указать относительный путь от текущей страницы. Например, src="script.js" будет означать, что файл "script.js" находится в текущей папке.

Можно указать и полный URL-адрес. Например:

<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js"></script>

Для подключения нескольких скриптов используйте несколько тегов:

<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>

…

На заметку:

Как правило, только простейшие скрипты помещаются в HTML. Более сложные выделяются в отдельные файлы.

Польза от отдельных файлов в том, что браузер загрузит скрипт отдельно и сможет хранить его в кеше.

Другие страницы, которые подключают тот же скрипт, смогут брать его из кеша вместо повторной загрузки из сети. И таким образом файл будет загружаться с сервера только один раз.

Это сокращает расход трафика и ускоряет загрузку страниц.

Если атрибут src установлен, содержимое тега script будет игнорироваться.

В одном теге <script> нельзя использовать одновременно атрибут src и код внутри.

Нижеприведённый пример не работает:

<script src="file.js">

alert(1); // содержимое игнорируется, так как есть атрибут src

</script>

Нужно выбрать: либо внешний скрипт <script src="…">, либо обычный код внутри тега <script>.

Вышеприведённый пример можно разделить на два скрипта:

<script src="file.js"></script>

<script>

alert(1);

</script>

директивы: "use strict".

## «use strict»

Директива выглядит как строка: "use strict" или 'use strict'. Когда она находится в начале скрипта, весь сценарий работает в «современном» режиме.

Например:

"use strict";

// этот код работает в современном режиме

...

Позже мы изучим функции (способ группировки команд). Забегая вперёд, заметим, что вместо всего скрипта "use strict" можно поставить в начале большинства видов функций. Это позволяет включить строгий режим только в конкретной функции. Но обычно люди используют его для всего файла.

Убедитесь, что «use strict» находится в начале

Проверьте, что "use strict" находится в первой исполняемой строке скрипта, иначе строгий режим может не включиться.

Здесь строгий режим не включён:

alert("some code");

// "use strict" ниже игнорируется - он должен быть в первой строке

"use strict";

// строгий режим не активирован

Над "use strict" могут быть записаны только комментарии.

Нет никакого способа отменить use strict

Нет директивы типа "no use strict", которая возвращала бы движок к старому поведению.

Как только мы входим в строгий режим, отменить это невозможно.

В старых браузерах консоль не учитывает такой use strict, там можно «оборачивать» код в функцию, вот так:

(function() {

'use strict';

// ...ваш код...

})()

## Всегда используйте «use strict»

Нам ещё предстоит рассмотреть различия между строгим режимом и режимом «по умолчанию».

В следующих главах, изучая особенности языка, мы будем отмечать различия между строгим и стандартным режимами. К счастью, их не так много, и они действительно делают нашу жизнь лучше.

На данный момент достаточно иметь общее понимание об этом режиме:

## Асинхронные скрипты: defer/async

Браузер загружает и отображает HTML постепенно. Особенно это заметно при медленном интернет-соединении: браузер не ждёт, пока страница загрузится целиком, а показывает ту часть, которую успел загрузить.

Если браузер видит тег <script>, то он по стандарту обязан сначала выполнить его, а потом показать оставшуюся часть страницы.

Например, в примере ниже – пока все кролики не будут посчитаны – нижний <p> не будет показан:

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<p>Начинаем считать:</p>

<script>

alert( 'Первый кролик!' );

alert( 'Второй кролик!' );

alert( 'Третий кролик!' );

</script>

<p>Кролики посчитаны!</p>

</body>

</html>

Такое поведение называют «синхронным». Как правило, оно вполне нормально, но есть важное следствие.

**Если скрипт – внешний, то пока браузер не выполнит его, он не покажет часть страницы под ним.**

То есть, в таком документе, пока не загрузится и не выполнится big.js, содержимое <body> будет скрыто:

<html>

<head>

<script src="big.js"></script>

</head>

<body>

Этот текст не будет показан, пока браузер не выполнит big.js.

</body>

</html>

И здесь вопрос – действительно ли мы этого хотим? То есть, действительно ли оставшуюся часть страницы нельзя показывать до загрузки скрипта?

Есть ситуации, когда мы не только НЕ хотим такой задержки, но она даже опасна.

Например, если мы подключаем внешний скрипт, который показывает рекламу или вставляет счётчик посещений, а затем идёт наша страница. Конечно, неправильно, что пока счётчик или реклама не подгрузятся – оставшаяся часть страницы не показывается. Счётчик посещений не должен никак задерживать отображение страницы сайта. Реклама тоже не должна тормозить сайт и нарушать его функциональность.

А что, если сервер, с которого загружается внешний скрипт, перегружен? Посетитель в этом случае может ждать очень долго!

Вот пример, с подобным скриптом (стоит искусственная задержка загрузки):

<p>Важная информация не покажется, пока не загрузится скрипт.</p>

<script src="https://js.cx/hello/ads.js?speed=0"></script>

<p>...Важная информация!</p>

Что делать?

Можно поставить все подобные скрипты в конец страницы – это уменьшит проблему, но не избавит от неё полностью, если скриптов несколько. Допустим, в конце страницы 3 скрипта, и первый из них тормозит – получается, другие два его будут ждать – тоже нехорошо.

Кроме того, браузер дойдёт до скриптов, расположенных в конце страницы, они начнут грузиться только тогда, когда вся страница загрузится. А это не всегда правильно. Например, счётчик посещений наиболее точно сработает, если загрузить его пораньше.

Поэтому «расположить скрипты внизу» – не лучший выход.

Кардинально решить эту проблему помогут атрибуты async или defer:

**Атрибут async**

Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении <script async src="..."> браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен – он выполнится.

**Атрибут defer**

Поддерживается всеми браузерами, включая самые старые IE. Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async.

Первое – браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

То есть, в таком коде (с async) первым сработает тот скрипт, который раньше загрузится:

<script src="1.js" async></script>

<script src="2.js" async></script>

А в таком коде (с defer) первым сработает всегда 1.js, а скрипт 2.js, даже если загрузился раньше, будет его ждать.

<script src="1.js" defer></script>

<script src="2.js" defer></script>

Поэтому атрибут defer используют в тех случаях, когда второй скрипт 2.js зависит от первого 1.js, к примеру – использует что-то, описанное первым скриптом.

Второе отличие – скрипт с defer сработает, когда весь HTML-документ будет обработан браузером.

Например, если документ достаточно большой…

<script src="async.js" async></script>

<script src="defer.js" defer></script>

Много много много букв

…То скрипт async.js выполнится, как только загрузится – возможно, до того, как весь документ готов. А defer.js подождёт готовности всего документа.

Это бывает удобно, когда мы в скрипте хотим работать с документом, и должны быть уверены, что он полностью получен.

async вместе с defer

При одновременном указании async и defer в современных браузерах будет использован только async, в IE9- – только defer (не понимает async).

Атрибуты async/defer – только для внешних скриптов

Атрибуты async/defer работают только в том случае, если назначены на внешние скрипты, т.е. имеющие src.

При попытке назначить их на обычные скрипты <script>…</script>, они будут проигнорированы.

Тот же пример с async:

<p>Важная информация теперь не ждёт, пока загрузится скрипт...</p>

<script async src="https://js.cx/hello/ads.js?speed=0"></script>

<p>...Важная информация!</p>

При запуске вы увидите, что вся страница отобразилась тут же, а alert из внешнего скрипта появится позже, когда загрузится скрипт.

Эти атрибуты давно «в ходу»

Большинство современных систем рекламы и счётчиков знают про эти атрибуты и используют их.

Перед вставкой внешнего тега <script> понимающий программист всегда проверит, есть ли у него подобный атрибут. Иначе медленный скрипт может задержать загрузку страницы.

**Итого**

* Для добавления кода JavaScript на страницу используется тег <script>
* Атрибуты type и language необязательны.
* Скрипт во внешнем файле можно вставить с помощью <script src="path/to/script.js"></script>.
* Директива "use strict" переключает движок в «современный» режим, изменяя поведение некоторых встроенных функций. Позже в учебнике мы увидим подробности.
* Строгий режим включается путём размещения "use strict" в начале скрипта или функции. Некоторые функции языка, такие как «классы» и «модули», автоматически включают строгий режим.
* Строгий режим поддерживается всеми современными браузерами.
* Рекомендуется всегда начинать скрипты с "use strict". Все примеры в этом руководстве предполагают строгий режим, если (очень редко) не указано иное
* Специальные атрибуты async и defer используются для того, чтобы пока грузится внешний скрипт – браузер показал остальную (следующую за ним) часть страницы. Без них этого не происходит.
* Разница между async и defer: атрибут defer сохраняет относительную последовательность скриптов, а async – нет. Кроме того, defer всегда ждёт, пока весь HTML-документ будет готов, а async – нет.