«Использование консоли разработчика JavaScript»

Оглавление

Введение

Иногда полезно иметь возможность запускать команды JavaScript без создания веб-страницы и последующего включения в нее кода сценариев или блоков <script>. В подобных ситуациях можно воспользоваться консолью JavaScript браузера Chrome (рис. 1).

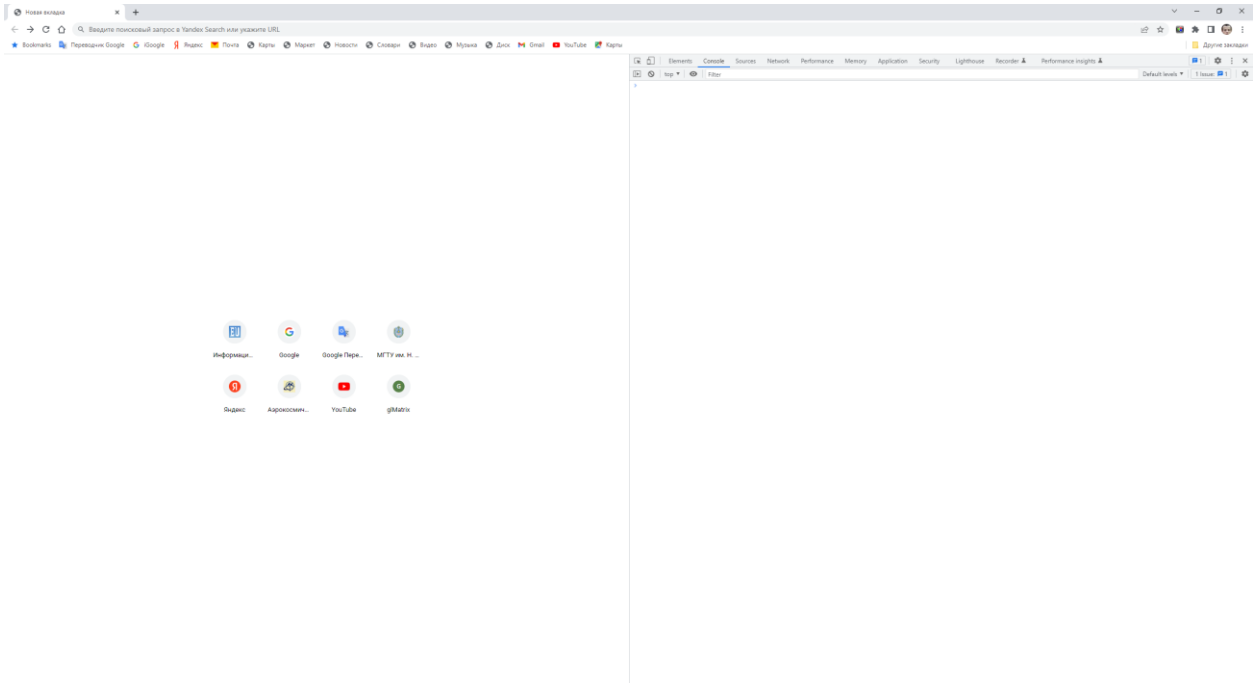


Рис. 1. Консоль JavaScript браузера Chrome

Чтобы получить доступ к консоли JavaScript, откройте меню Chrome, расположенное в правом верхнем углу окна браузера. Ему соответствует пиктограмма в виде трех точек, расположенных вертикально. Щелкните на ней и выберите в раскрывшемся меню пункт Дополнительные инструменты, а в следующем меню — пункт Инструменты разработчика.

Для открытия окна консоли JavaScript существует и более быстрый способ, требующий простого нажатия комбинации клавиш <Alt+Command+J> (Mac) или F12 (Windows).

Также можно щелкнуть правой кнопкой мыши по элементу на веб-странице и выбрать «Просмотреть код» во всплывающем меню. Далее можно переключиться на вкладку Консоль.

Консоль JavaScript является, пожалуй, наилучшим другом разработчика на JavaScript. Она не только позволяет быстро и просто тестировать и выполнять JavaScript-код, но и сообщает, в каких местах кода содержатся ошибки, и предлагает средства, облегчающие обнаружение и разрешение проблем, связанных с работой кода. В консоли указывается имя файла с ошибкой и номер строки. Если щелкнуть мышью по имени файла, то можно открыть его и увидеть выделенную строку с ошибкой.

Сразу же после открытия консоли можно начать ввод команд, которые будут выполняться после нажатия клавиши <Enter>. Чтобы попрактиковаться в этом, откройте консоль JavaScript и последовательно вводите приведенные ниже команды, нажимая клавишу <Enter> после ввода каждой из них.

```
1080/33
```

```
40+2
```

```
40*34
```

```
100%3
```

Стрелки, направленные влево и вправо показывают, какие строки ввели вы, а какие принадлежат интерпретатору.

Ввод функций.

Чтобы проверить работу функции, выполните следующие действия.

1. Откройте консоль JavaScript в браузере Chrome.
2. Введите код функции в консоли.

Можете ввести весь текст в одной строке или же нажимать клавиши <Shift+Enter> для создания нескольких строк, прежде чем выполнить код.

```
function addNumbers(numbersToAdd) {
      let sum = 0;
      for (let oneNumber=0; oneNumber<numbersToAdd.length; oneNumber++)
      {
            sum = sum + numbersToAdd[oneNumber];
      }
      return sum;
}
```

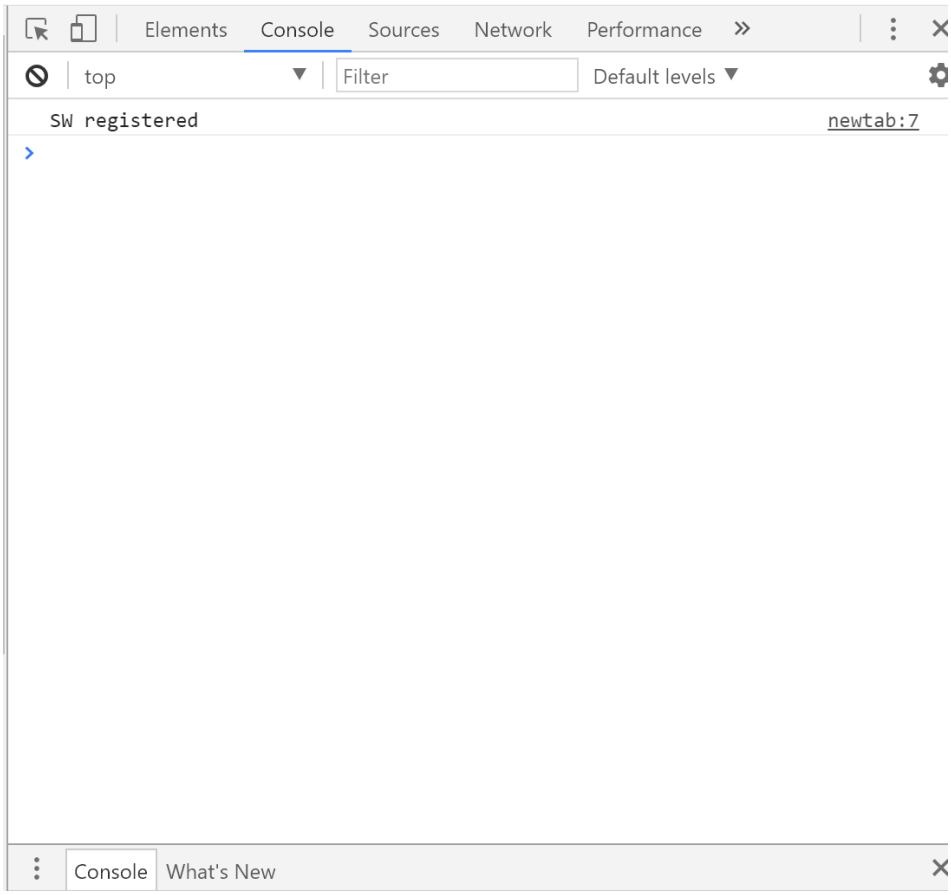
3. После ввода завершающей фигурной скобки нажмите клавишу <Enter>.

В консоли должен отобразиться текст `undefined`.

4. Введите следующие команды и нажмите клавиши <Enter>:

```
const myNumbers = [2,4,2,7];
addNumbers(myNumbers);
```

В Chrome для очистки консоли используется значок с перечеркнутым кружком (см. левый верхний угол):



Он сообщает интерпретатору, что ему больше не нужно помнить переменные, которые были созданы.

Отслеживание ошибок

Самый неприятный момент наступает тогда, когда вы пытаетесь посмотреть в браузере страницу, содержащую код JavaScript… и ничего не происходит. Большинство браузеров настроено на тихое игнорирование ошибок JavaScript, поэтому зачастую вы даже не увидите никаких сообщений, если не откроете консоль JavaScript. Именно в консоли будет содержаться описание ошибки, а также номер строки, в которой она находится.

Консоль поможет «отловить» основные опечатки, например, пропуски закрывающих знаков пунктуации или неправильно введенные имена команд JavaScript.

Рассмотрим некоторые распространенные ошибки и сообщения о них в браузере Chrome:

- **Пропущенный символ пунктуации**. Программы JavaScript включают множество пар символов, например, открывающих и закрывающих круглых или квадратных скобок. Если вы введете `alert('привет';` забыв поставить закрывающую круглую скобку, то, вероятно увидите сообщение `Uncaught SyntaxError: missing ) after argument list`, которое означает, что браузер ожидал обнаружить другие символы. В данном случае программе встретилась точка с запятой вместо закрывающей круглой скобки.
- **Отсутствие кавычек**. Кавычки используются для задания строковых констант в JavaScript. Если в коде была забыта открывающая или закрывающая кавычка, или вместо одинарной была использована двойная кавычка `alert(‘привет”);`. В любом случае, будет выведено сообщение `Uncaught SyntaxError: Invalid or unexpected token.`

- **Опечатки в командах**. Если вы неправильно запишите команду в JavaScript, например, `aler('привет');` то получите сообщение об ошибке, из которого узнаете, что неправильно записанная команда не определена, например `Uncaught ReferenceError: aler is not defined`. Если ошибка была допущена в названии метода существующего класса, то ошибка будет такой `Uncaught TypeError: Object [object Object] has no method 'название'`.

Точки останова

Вы можете приостановить выполнение сценария в любом месте, *используя точки останова (breakpoints)*, и проверить, какие значения имеют переменные на этом этапе.

Chrome

1. Перейдите на вкладку `Sources (Исходный код)`.
2. Выберите в левой части панели сценарий, с которым вы хотите работать. Справа появится его код.
3. Найдите номер строки, где вы хотите сделать паузу, и щелкните мышью слева от номера строки. Нельзя добавить точку останова в строку, содержащую только комментарий. Убрать точку останова можно щелкнув по ней.
4. Во время выполнения сценарий остановится на этой строке. В разделе `Call Stack` показываются функции, которые были выполнены до остановки. Чтобы увидеть значение любой переменной на момент выполнения сценария, достаточно установить на ее имя указатель мыши или добавить выражение в окно `Watch` (Наблюдение), щелкнув по кнопке `Add watch expression… (Добавить наблюдение…)` в виде символа «+». Просто щелкните по ней, и появится текстовое поле. Ведите имя переменной, которую вы хотите отследить, или инструкцию JavaScript, которую желаете выполнить. В разделе `Scope` (Область видимости) перечислены все переменные, принадлежащие «области видимости» текущей позиции.

Пошаговое выполнение кода

Если вы установили несколько точек останова, вы можете последовательно пройтись по ним, чтобы отследить изменение значений и появление проблем.

1. Значок паузы показывается, пока интерпретатор не доберется до точки останова. Когда такое происходит, он меняется на значок проигрывания. Это позволяет возобновить выполнение кода.
2. Переход к следующей строке кода и пошаговое выполнение (с остановкой на каждом шаге).
   Если отладчик сталкивается с функцией, он переходит на следующую строку за ней (как будто переступает ее), не заходя внутрь ее определения.
   При желании вы можете заставить отладчик войти внутрь функции, чтобы увидеть, что в ней происходит.
3. Заход внутрь вызовов. Интерпретатор перейдет к первой строке заданной функции.
4. Выход из функции, в которую вы зашли ранее. Остальной код функции будет выполнен, когда отладчик перейдет к внешнему коду.

Условные контрольные точки

Вы можете сделать так, чтобы контрольная точка срабатывала только при выполнении определенного условия. В условии допускается использовать существующие переменные.

Chrome

1. Щелкните правой кнопкой мыши на номере строки кода.
2. Выберите пункт меню `Add Conditional Breakpoint` (Добавить условную контрольную точку).
3. Введите условие во всплывающем окне.
4. Когда вы запустите сценарий, он остановится на этой строке только в том случае, если условие будет истинным.

Ключевое слово `debugger`

Контрольную точку в коде можно создать с помощью одного только ключевого слова `debugger`. Интерпретатор автоматически останавливается на нем, когда открыты инструменты разработки.

Крайне важно не забыть удалить такие инструкции перед развертыванием кода в реальных условиях. В противном случае, если у пользователя будут открыты инструменты разработки, ваш сценарий может остановиться.

Если у вас есть тестовый сервер, вы можете поместить отладочный код внутрь условной инструкции, которая проверяет, в какой среде происходит выполнение сценария (тогда отладочный код будет работать только на определенном сервере).

Коды ошибок WebGL

Коды ошибок WebGL	Описание
`gl.NO_ERROR`	Ошибок нет (с момента последнего вызова `gl.getError()`).
`gl.INVALID_ENUM`	Аргумент типа `GLenum` имеет недопустимое значение. Например, ошибка возникнет, если вызвать `gl.drawArrays()` и передать значение первого аргумента равное `gl.COLOR_BUFFER_BIT`.
`gl.INVALID_VALUE`	Целочисленный аргумент не принадлежит заданному диапазону значений. Например, ошибка возникнет, если вызвать `gl.clear()` и передать ему значение `gl.POINTS`.
`gl.INVALID_OPERATION`	Ошибка возникает в случае, когда вызывается метод без перевода WebGL state в нужное состояние, т.е. не были вызваны некоторые обязательные предварительные функции перед вызовом основной функции. Например, ошибка возникнет, если вызвать `gl.bufferData()` без предварительного вызова функции `gl.bindbuffer()`.
`gl.OUT_OF_MEMORY`	Не хватает памяти для выполнения соответствующего метода.