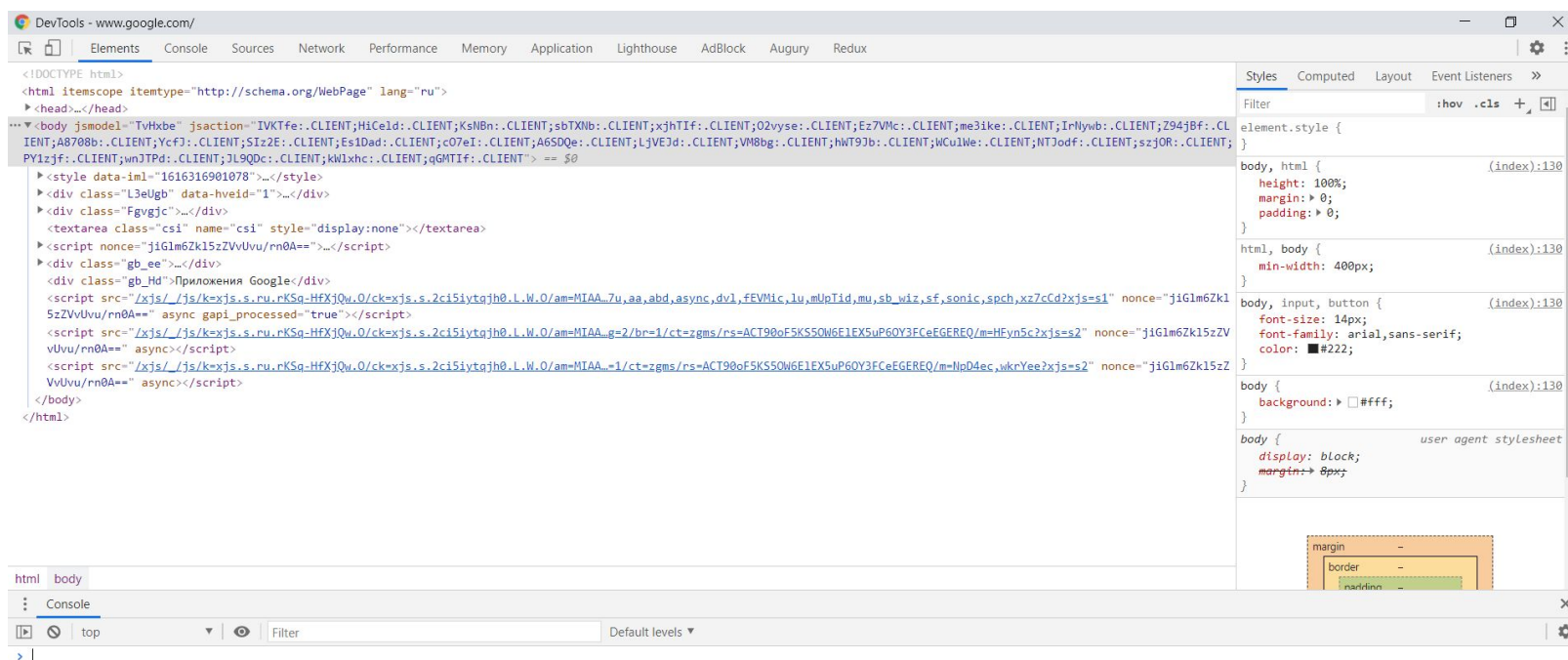


JavaScript

Dev tools

Отладка кода в dev tools

Все современные браузеры поддерживают «инструменты разработчика». Исправление ошибок с их помощью намного проще и быстрее.



Что можно делать с помощью dev tools?

1. Искать и устранять дефекты вёрстки «на лету»
2. Разбирать иерархию наследования каскадных стилей
3. Отлаживать JS и изменять его в режиме реального времени
4. Динамически проверять тестовые решения прямо в консоли
5. Исследовать загружаемые ресурсы
6. Просматривать отображение приложения в различных разрешениях
7. Анализировать проблемы безопасности
8. Анализировать утечки памяти

Elements

The screenshot shows the Chrome DevTools interface for the URL `yandex.ru/`. The **Elements** panel on the left displays the DOM tree of a search bar. The selected element is a `form` with class `search2`, containing various mini-suggest components and a search input wrapper. The **Styles** panel on the right shows the default browser styles for the selected element, including `display: block`, `margin-right: 16px`, `border-radius: 6px 0 0 6px`, and `background-color: #ffdb4d`. Below the styles panel, the **Console** panel shows a list of 11 non-unique items, each with a warning icon and the message "Non unique item: , no title, replaced to _N".

```
<div class="mv mw_sX mw_TJ">
  <aqwf>
    <ftgr>
      <dbgrsae>
        <div class="aY EN Aa_Dj_S1">
          <div class="home-arrow i-bem home-arrow_js_initied" data-bem="{"home-arrow":{}}">
            <div class="home-arrow__search-wrapper">
              <div class="home-logo home-arrow__logo"></div>
              <div class="home-arrow__search">
                <form class="search2 mini-suggest mini-suggest_search_yes mini-suggest_personal_yes mini-suggest_theme_flat mini-suggest_tab-change_yes mini-suggest_request_xhr mini-suggest_separate-popup_yes mini-suggest_autofocus_yes mini-suggest_re-request_yes mini-suggest_direct_yes mini-suggest_ally_yes i-bem search2_empty_yes search2_js_initied" action="https://yandex.ru/search/" role="search" aria-label="Поиск в интернете" data-bem="{"search2":{"n1":true}}" data-mini-bem="{"mini-suggest":{"url":"https://yandex.ru/suggest/suggest-ya.cgi?abtestids=323029&no_ml=1&a=0&apply_cm2_boosting_to_ranking_weights=1&cm2_boost_integration_coefficient=0&srv=morda_ru_desktop&wiz=TrWth&u1=ru&fact=1&v=4&icon=1&lr=47&hl=1&bmjson=0&history=1&html=1&platform=desktop&rich_nav=1&show_experiment=222&show_experiment=224&verified_nav=1&rich_phone=1&safeclick=1&skip_clickdaemon_host=1&yu=53372181607274690&maybe_ads=1&mt_wizard=1","deleteUrl":"https://yandex.ru/suggest/delete-text?srv=morda_ru_desktop&text_to_delete=","counter":{"service":"morda_ru_desktop","url":"https://yandex.ru/clck/jclck","timeout":300,"params":{"dtype":"stred","pid":"0","cid":"2873","lr":"47"},"yandexuid":"53372181607274690","reqIdInCookie":null,"src":true},"clickHost":"https://yandex.ru/clck","navSource":"/","sameSiteSupport":1,"ally":{"nav":"внешняя ссылка"},"direct":{"label":"поиск","counter_delay":5000,"adb":true},"requestTimeout":200,"requestUrl":"https://yandex.ru/suggest/ad-rerequest-ya.cgi?abtestids=323029&no_ml=1&a=0&apply_cm2_boosting_to_ranking_weights=1&cm2_boost_integration_coefficient=0&srv=morda_ru_desktop&wiz=TrWth&u1=ru&fact=1&v=4&icon=1&lr=47&hl=1&bmjson=0&history=1&html=1&platform=desktop&rich_nav=1&show_experiment=222&show_experiment=224&verified_nav=1&rich_phone=1&safeclick=1&skip_clickdaemon_host=1&yu=53372181607274690&maybe_ads=1&mt_wizard=1","requestMaxLatency":0}}"> == $0
              <div class="search2__wrapper">
                ...
            </div>
          </div>
        </div>
      </ftgr>
    </aqwf>
  </div class="mv mw_sX mw_TJ">
```

... form.search2.mini-suggest.mini-suggest_search_yes.mini-suggest_personal_yes.mini-suggest_theme_flat.mini-suggest_tab-change_yes.mini-suggest_request_xhr.mini-suggest_separate-popup_yes.mini-suggest_autof ...

Console

Filter

Default levels

15 hidden

- Non unique item: , no title, replaced to _2
- Non unique item: , no title, replaced to _3
- Non unique item: , no title, replaced to _4
- Non unique item: , no title, replaced to _5
- Non unique item: , no title, replaced to _6
- Non unique item: , no title, replaced to _7
- Non unique item: , no title, replaced to _8
- Non unique item: , no title, replaced to _9
- Non unique item: , no title, replaced to _10
- Non unique item: , no title, replaced to _11

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

1fKTfmx_nPfwH5nY7h8K_LHv3z5_Ruf_Spv4w:59

Elements

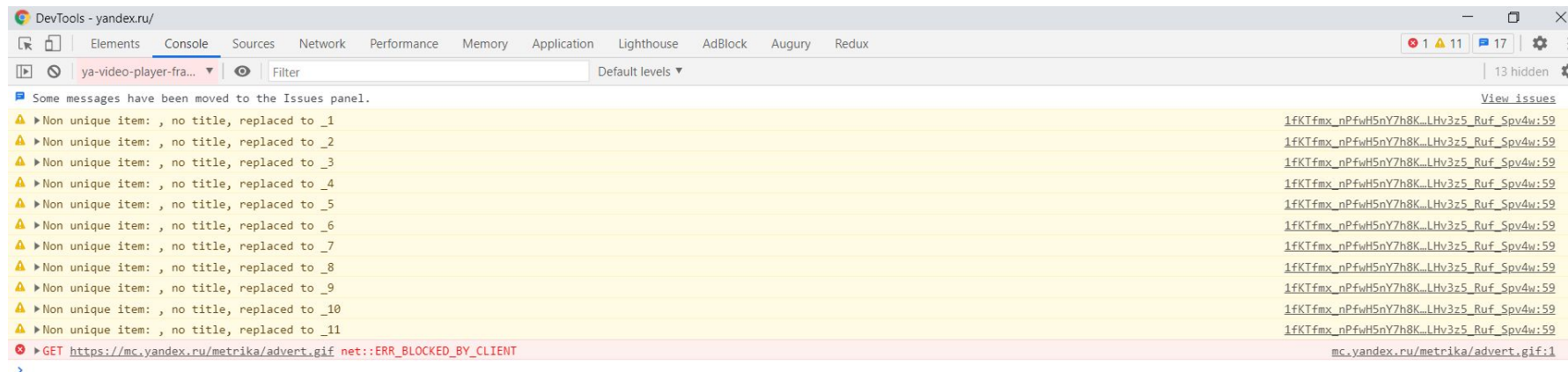
Главный раздел, где находится html разметка документа. Позволяет:

- Инспектировать html
- Находить и исправлять верстку
- Добавлять или удалять атрибуты
- Искать нужные элементы

Справа – вся необходимая информация по CSS и взаимодействию с элементами.

- Добавлять и удалять стили
- Исследовать иерархию наследования CSS
- Искать в каком компоненте был добавлен текущий стиль
- В подтабе Event Listeners можно исследовать зарегистрированные подписки на элемент.

Console



Console

Console

Незаменимый инструмент в отладке кода. Позволяет онлайн выполнять куски кода. Находясь на точке остановки позволяет обращаться к переменным в текущем окружении.

Консоль открывается клавишей **Esc** или переходом в раздел **Console**.

Самая часто используемая команда: **console.log()** позволяет выводить в консоль переменные/текст/объекты и тп.

Есть ещё одна полезная команда **console.dir()**. Она позволяет выводить объект с его структурой, прототипом и тд.

Console

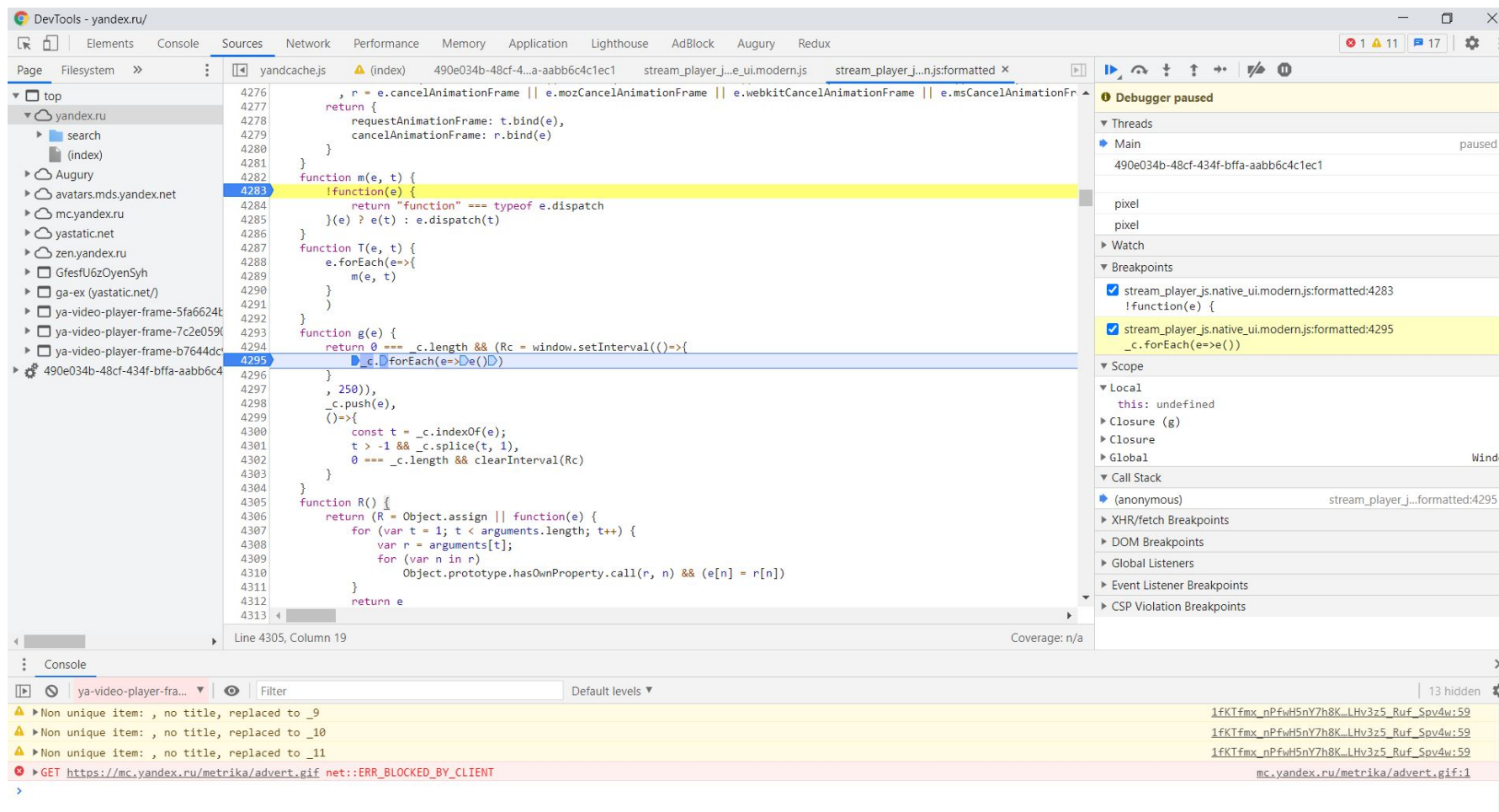
- **console.warn()** – помечает сообщение жёлтым цветом и значком warning
- **console.error()** – помечает сообщение красным цветом и значком ошибки
- **console.group()** – группирует некие связанные данные
- **console.table()** – выводит данные в консоль в виде таблицы
- **console.time()** – позволяет замерять время выполнения операций
- **console.trace()** – выводит в консоль результаты трассировки стека и позволяет судить о том, что произошло в определённом месте программы

Ошибки

Ошибки при выполнении скрипта выводятся в консоль автоматически. Добавить в консоль ошибку можно и вручную, с помощью **console.error()**. Навигироваться на код, где произошла ошибка можно по ссылке, расположенной справа от ошибки.

```
Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
Uncaught Error: Invocation of form runtime.connect(null, ) doesn't match def
  at Object.normalizeArgumentsAndValidate (VM26_extensions::schemaUtils:11)
  at Object.<anonymous> (VM21_extensions::binding:363)
  at Object.<anonymous> (VM20_extensions::runtime:56)
  at Object.handleRequest (VM21_extensions::binding:64)
  at Object.<anonymous> (VM21_extensions::binding:374)
  at Object.value [as sendMessage] (polyfill.js:95)
  at checkCollapse (include.preload.js:201)
  at HTMLDocument.document.addEventListener.event (include.preload.js:602)
Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
Failed to load resource: net::ERR_BLOCKED_BY_CLIENT
Failed to load resource: the server responded with a status of 404 ()
undefined || false
false
7 Uncaught Error: Invocation of form runtime.connect(null, ) doesn't match d
  at Object.normalizeArgumentsAndValidate (VM26_extensions::schemaUtils:11)
  at Object.<anonymous> (VM21_extensions::binding:363)
  at Object.<anonymous> (VM20_extensions::runtime:56)
  at Object.handleRequest (VM21_extensions::binding:64)
  at Object.<anonymous> (VM21_extensions::binding:374)
  at Object.value [as sendMessage] (polyfill.js:108)
  at HTMLDocument.document.addEventListener.event (composer.postload.js:4
```

Sources



Sources

Главный раздел – исходный код позволяет:

- Инспектировать и отлаживать код
- Изменять отдельные куски кода «на лету»

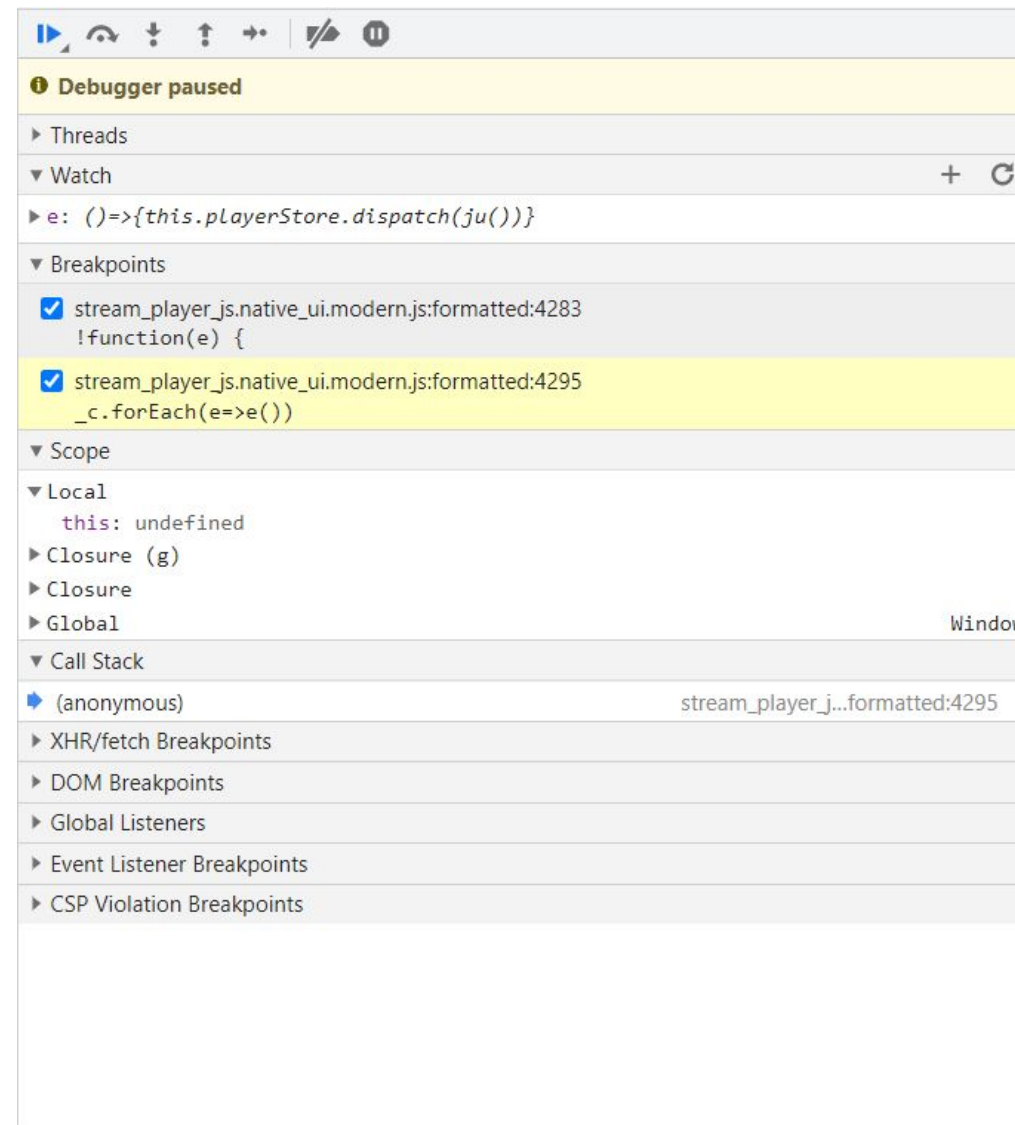
Иерархия исходных файлов (слева, обычно скрыт)

Информационная зона (Включает в себя элементы контроля) позволяет:





- Останавливать/продолжать выполнение приложения
- Контролировать направление, а именно зайти внутрь, пропустить и тп
- Исследовать переменные в текущем окружении
- Исследовать стек вызова методов
- Следить за значениями конкретных переменных
- Следить за точками остановок

Раздел отладки

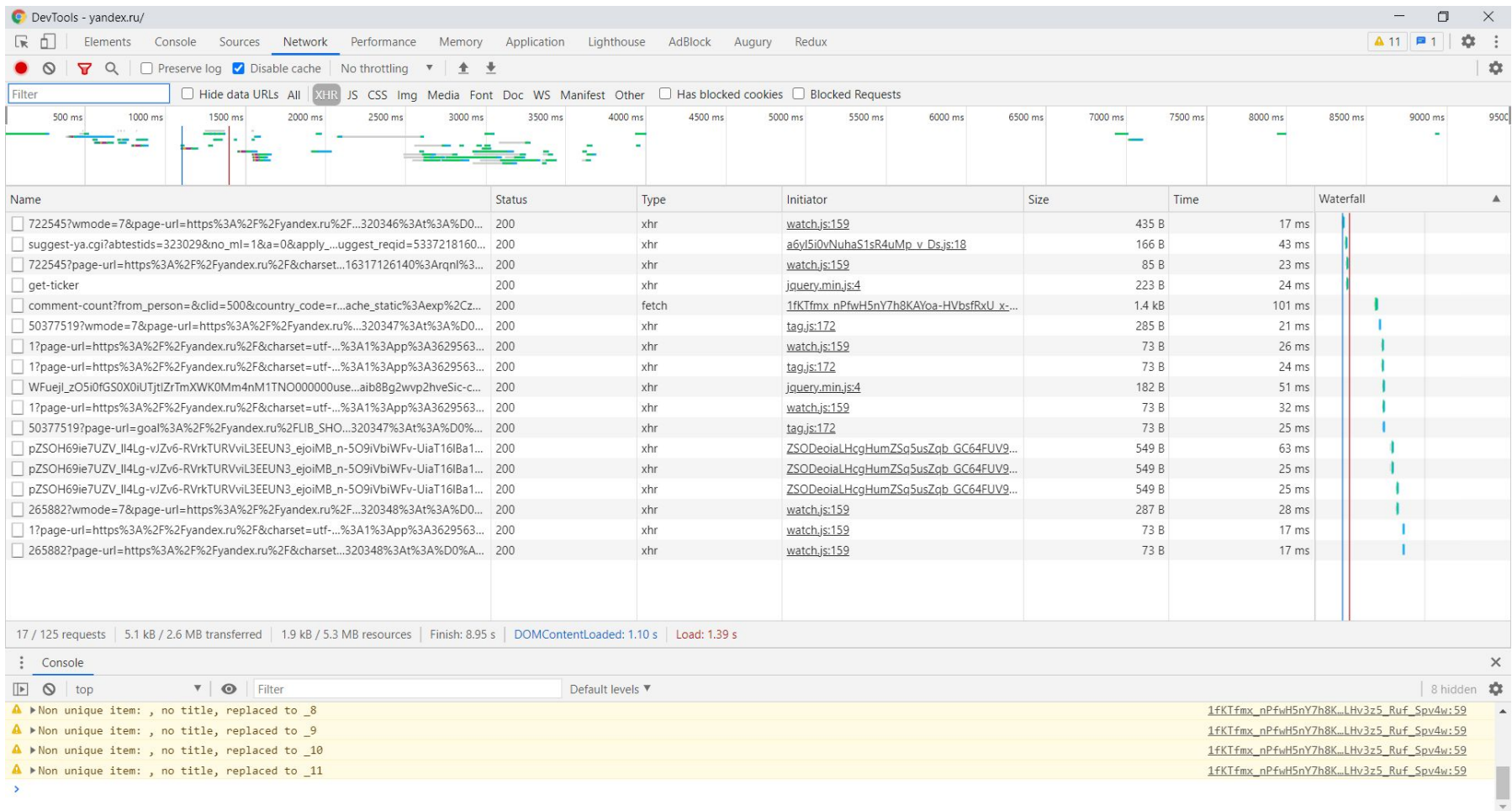
- **Watch** – показывает текущее значение выражений, добавлять выражение нужно вручную
- **Breakpoints** – точки остановок
- **Scope** – переменные окружения
 - Local – локальные переменные текущего окружения, `this` – ссылка на текущий объект
 - Global – глобальные переменные и функции
- **Call stack** – стек вызовов, все вложенные вызовы, которые привели к текущему месту кода



Управление отладкой

-  **Продолжить выполнение (F8)** – продолжение выполнения скрипта, до новых точек остановки
-  **Сделать шаг не заходя внутрь функции (F10)** – выполнение команды, находящейся на текущей строке, если в ней находится вызов функции, не переходит в её реализацию
-  **Сделать шаг с заходом в функцию (F11)** – аналогично предыдущему, только с заходом в функцию
-  **Сделать шаг до выхода из текущей функции (Shift + F11)** – выполняет скрипт до выхода из текущей функции
-  **Отключить все точки остановок (ctrl + F8)** – отключает все точки остановок
 - **Отключить/включить автоматическую остановку при ошибке** – включает
-  режим остановки на ошибках

Network

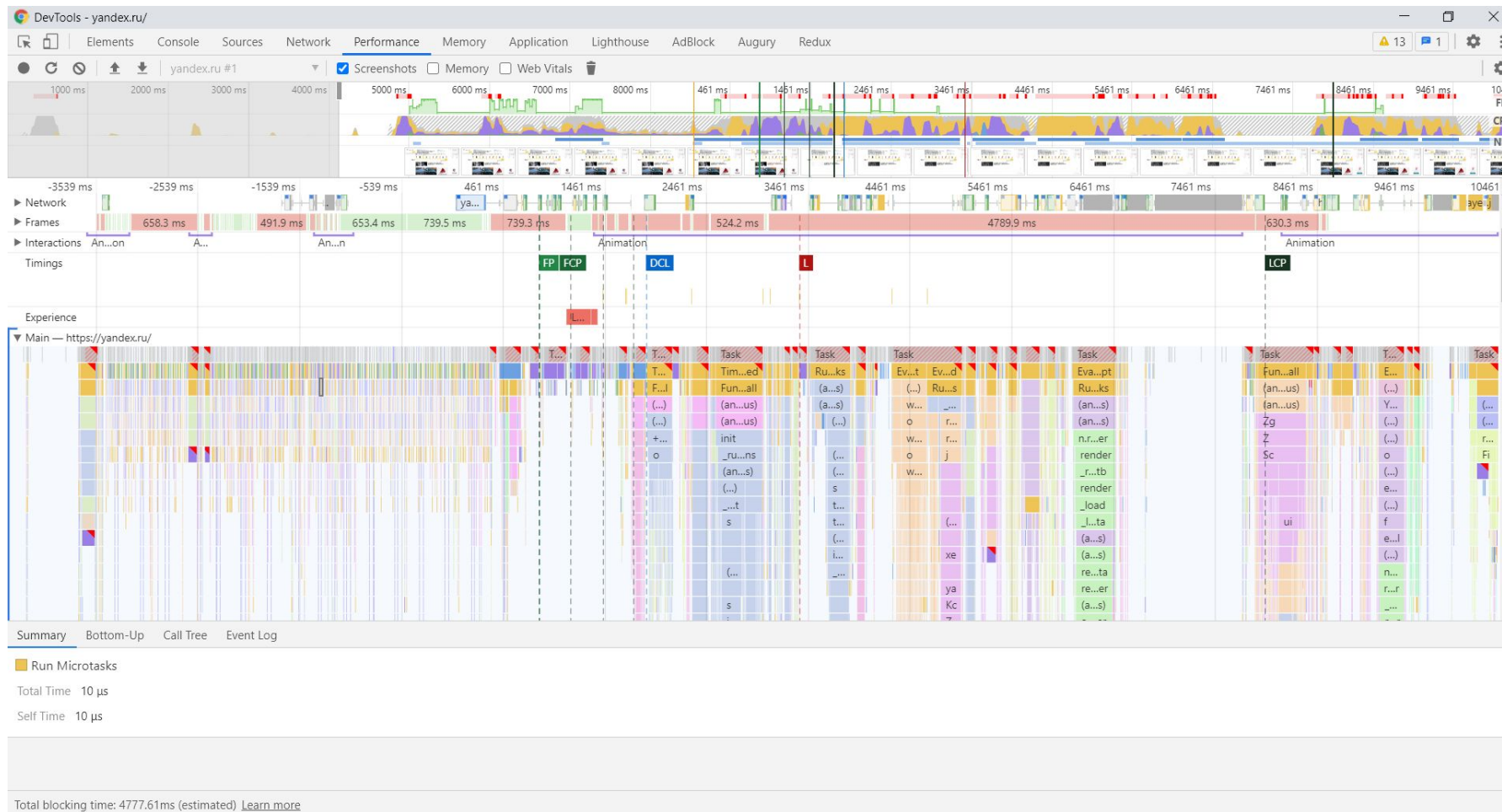


Network

С помощью вкладки Network можно выяснить, сколько времени заняла загрузка страницы, какие ресурсы подключились к странице и многое другое.

Помимо этого можно включить имитацию медленного интернет-соединения (ограничить пропускную способность), отключить кеши или дополнительно отфильтровать ресурсы.

Perfomance



Perfomance

Панель отображает таймлайн использования сети, выполнения JS и загрузки памяти. После первоначального построения будут доступны различные подробные данные о выполнении кода и всем жизненном цикле страницы.

Этот инструмент применяется для улучшения и анализа производительности работы приложения.

Особенности:

- Возможно сделать запись для проведения анализа каждого события
- Возможно просмотреть FPS, загрузку CPU и сетевые запросы.
- Щёлкнув по событию на диаграмме появятся детали
- Можно изменять масштаб таймлайна

Application

The screenshot shows the Chrome DevTools Application panel for the URL `yandex.ru/`. The left sidebar contains a tree view of application components: Manifest, Service Workers, Storage, Local Storage, Session Storage (selected), IndexedDB, Web SQL, Cookies, Cache, Background Services, and Frames.

The main pane displays a table of storage items for the selected `https://yandex.ru` session storage. The table has two columns: Key and Value.

Key	Value
<code>_ym22227335_reqNum</code>	<code>22</code>
<code>yandexJSPlayerApiSavedSingleVideoSessionWatchedTimeSinceAdUtcTimestamp</code>	<code>1616317136.936</code>
<code>videoplayer-ad-session-id-last-service-name</code>	<code>"other"</code>
<code>_ym722545_reqNum</code>	<code>8</code>
<code>_ym_synced</code>	<code>{ "SYNCED_ADM": 26938618 }</code>
<code>_ym58984852_isid</code>	<code>166051162204</code>
<code>videoplayer-ad-session-id-last-adsid</code>	<code>"e521f5e41394f17b00ed47740fc965da582609806e1dxWEBx6503x1616317130"</code>
<code>_ym265882_reqNum</code>	<code>4</code>
<code>_ym22227335_lastHit</code>	<code>1616188079351</code>
<code>_ym3_reqNum</code>	<code>3</code>
<code>_ym50377519_isid</code>	<code>1294999270493</code>
<code>videoplayer-bandwidth-estimate</code>	<code>{ "bandwidthEstimate": 4620627, "time": 1616317878455 }</code>
<code>_ym_uid</code>	<code>"1607350998714854638"</code>
<code>test</code>	<code>1</code>
<code>yu</code>	<code>v1.0_53372181607274690:1616320345</code>
<code>_ym_wasSynced</code>	<code>{ "time": 1616317122702, "params": { "eu": 0 }, "bkParams": {} }</code>

Below the storage table, the 'Background Services' section shows a list of services with a '1' in the first column and '22' in the second column.

The bottom of the panel shows the Console tab with several log messages indicating non-unique items being replaced to indices 8, 9, 10, and 11. Each message is followed by a long alphanumeric string.

Application

Данная вкладка нужна для инспектирования и очистки всех ресурсов, таких как session storage, local storage, куков, кеша приложения, шрифтов и тд.

Ключевыми возможностями являются:

- Быстрая очистка хранилищ и кеша
- Инспектирование и управление хранилищами
- Инспектирование и удаление файлов cookie

Clean code.
Code style.

Преимущества и признаки чистого кода

- Программист должен легко читать код
- Быстрое понимание кода => быстрое решение задач
 - Добавление новых возможностей
 - Поиск и устранение проблем

Признаки хорошего кода:

- Простой, логичный, понятный
- Чистый и структурированный
- Краткий и лаконичный
- Стилистически единый

Плохой код и откуда он берётся

Признаки плохого кода:

- Непонятный, заумный и сложный
- Запутанный, визуально нагруженный
- Длинный и повторяющийся
- Беспорядочный

Откуда берётся такой код?

- Наследие и сторонний код
- Отсутствие стандартов кодирования
- Различная квалификация сотрудников
- Низкая культура
- Нехватка времени

Правила именования

- Осмысленные имена
 - `getSomethingSuperNeededForDrawGraph()`
 - `getFilteredRows()`

- Magic numbers

`let totalHours = 8*5*4 // плохой пример`

```
const WORKING_HOURS = 8;
```

```
const WORKING_DAYS = 5;
```

```
const WEEK_IN_MONTH = 4;
```

```
let totalHours = WORKING_HOURS * WORKING_DAYS * WEEK_IN_MONTH
```


Правила именования

- Одно слово для каждой концепции
 - fetch/get/retrieve
 - read/load
 - write/upload

- Имена классов и объектов

Должны представлять собой существительные и их комбинации.

- Customer
- Account

- Имена методов

Представляют собой глаголы или глагольные словосочетания.

- getName/setName

Правила именования

- Именование переменных и членов класса

Плохие примеры:

- flag
- _array
- newArray

Хорошие примеры:

- isBlocked
- sortedArray
- currentPosition

Итоги и советы по именованию

Хорошие имена позволяют быстрее ориентироваться в коде и понимать что происходит вокруг.

Хорошее имя:

- Передает назначение
- Имеет разумную длину
- Легко читается
- Быстро вспоминается и ищется
- Не дезинформирует

Функции

- **Длина** – должна быть оптимальной (зачастую это та, которая позволяет разместить функцию на одном экране без пролистывания. Примерно 25-50 строк)
- **Оптимальное количество аргументов** – аргументов не должно быть слишком много (0 – отлично, 1-2 – хорошо, 3 и более уже стоит задуматься)
- **Функциональность** – функция должна выполнять одну операцию. Ничего лишнего.

Комментарии

Комментарии это плохо. Почему? Потому что:

- Они могут быть неактуальными.
- Могут быть избыточными
- Могут запутывать и дезинформировать

Самое плохое в комментариях это закоментированный код!

Комментарии

Исключениями могут быть:

- Комментарии, которые поясняют зачем что-то делается.
- Пояснения в нетривиальных случаях.
- Документация

Рефакторинг

Это изменение внутренней структуры программы без изменений ее видимого поведения с целью облегчить понимание и удешевить модификацию.

М. Фаулер

Цели рефакторинга:

- Достичь лучшего понимания и читаемости кода
- Упростить добавление нового кода
- Улучшить дизайн существующего кода

Процесс рефакторинга:

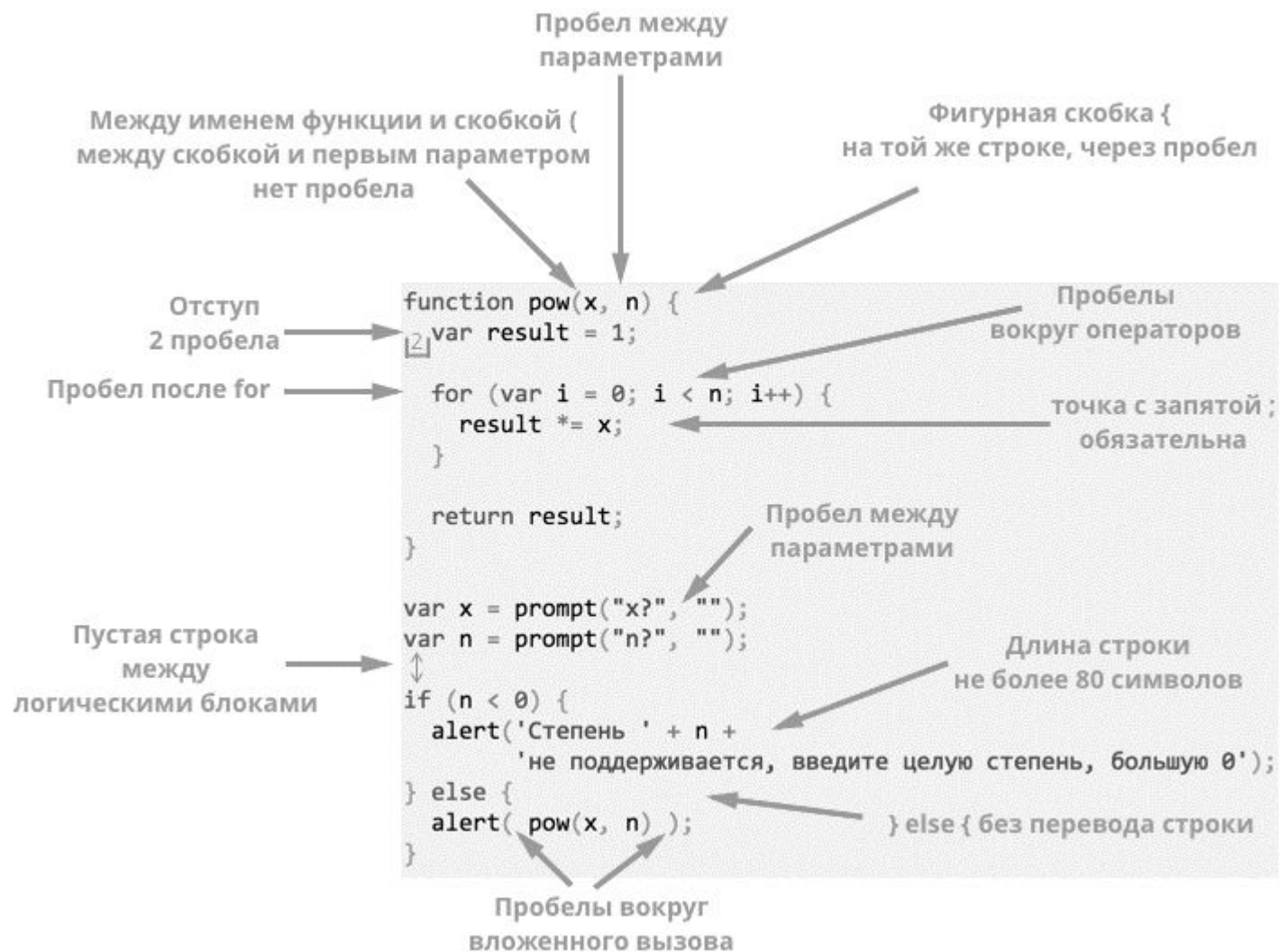
- Удаление дублирования
- Упрощение сложной логики
- Прояснение непонятного кода

Ключевые моменты

- Некачественный код – смерть проекта
- Качество кода – ответственность разработчика
- Чистка кода – непрерывная активность

Code style

```
if (n < 0) {  
    alert('Степень ' + n + ' не поддерживается');  
}  
  
show("Строки" +  
    "    выровнены" +  
    "    строго" +  
    "    одна под другой");  
  
function pow(x, n) {  
    var result = 1;  
    //      <--  
    for (var i = 0; i < n; i++) {  
        result *= x;  
    }  
    //      <--  
    return result;  
}
```



Автопроверка code style

Самые известные автоматические средства для проверки стиля кода:

- **JSLint** – средство проверки синтаксиса JS
- **JSHint** – форк от JSLint, управляемый сообществом (более конфигурируемый)
- **TSLint** – средство проверки и форматирования для typescript
- **ESLint** – средство проверки и форматирования синтаксиса JS

ESLint

19 февраля 2019 года вышел официальный пост о прекращении разработки **TSLint**. В нём компания-разработчик советует переходить на **ESLint**.

ESLint обладает гораздо большим количеством описанных правил, покрывает всё, что может быть во фронте. В нём есть правила, которые позволяют проверять код на уровне блоков и выявлять дублирование кода, сложность восприятия и тп. Есть множество плагинов, которые, например, проверяют регулярные выражения.

Перехват и обработка ошибок

Перехват ошибок. Try catch

Это способы «правильно» обрабатывать исключительные ситуации.

Правильность заключается в обёртке кода, где предположительно может быть ошибка, специальной конструкцией **try...catch**.

```
try {  
    // потенциально опасный код  
} catch (error) { // обработка ошибки }
```

Принцип работы:

1. Выполняется код внутри блока **try**
2. Если ошибок не произошло, блок **catch** игнорируется
3. Если произошла ошибка – выполняется блок **catch**

Перехват ошибок

Цель такой конструкции заключается в необходимости оградить скрипт от возможных ошибок. Другими словами, при возникновении ошибки выполнение скрипта не должно прерываться.

Ошибки могут возникать как автоматически:

- При неправильном использовании переменных
- При обращении к несуществующему полю
- При использовании несуществующих аргументов и тд.

Так могут и создаваться вручную, с помощью ключевого слова **throw**.

Перехват ошибок. Throw

```
Try {  
    let err = new Error("can't create");  
    err.message = "Can't resolve";  
    err.name = "ERROR";  
    throw err;  
} catch (error) {  
    console.dir(error);  
}
```

Аргументом `error` метода **catch** будет являться то, что было «проброшено» после **throw**. В данном случае это будет объект `Error`.

Перехват ошибок. Объект Error

В случае возникновения ошибки JS генерирует объект, содержащий детали этой ошибки. Для всех встроенных ошибок такой объект имеет несколько основных свойств:

- **name** – имя ошибки
- **message** – сообщение о деталях ошибки
- **stack** – текущий стек вызова. Строка, содержащая информацию о последовательности вложенных вызовов, которые привели к ошибке

Перехват ошибок. Условный блок catch

Условным блоком **catch** называется структура **try...catch** с использованием **if...else**, например:

```
try {  
    // код  
} catch (error) {  
    if (error instanceof TypeError) {  
        // обработка исключения TypeError  
    } else {  
        // обработка остальных исключений  
    }  
}
```

Перехват ошибок. Секция **finally**

Конструкция **try...catch** может содержать ещё одну секцию – **finally**. Эта секция не обязательна, но если она есть, то она выполняется всегда. Не зависимо от того была ошибка или нет. Секция **finally** используется для того, чтобы завершить начатые операции при любом варианте развития событий.

```
try {  
    // пробуем выполнить код  
} catch (error) {  
    // перехватываем исключение  
} finally {  
    // выполняем всегда  
}
```

Перехват ошибок. Секция finally

```
function func() {  
  try {  
    return 1;  
  } catch (error) {  
    // перехватываем исключение  
  } finally {  
    console.log("finally")  
  }  
}  
console.log(func())
```

В данном примере присутствует return, но finally отработает раньше.

Перехват ошибок. `Window.onerror`

В случае, когда ошибка произошла вне блока `try...catch` или выпала наружу, во внешний код – скрипт упадёт. Чтобы этого избежать можно воспользоваться специальным свойством `window.onerror`. Если в него записать функцию, то она выполнится и получит в аргументах сообщение ошибки, текущий URL и номер строки, откуда выпала ошибка.

```
window.onerror = (message, url, lineNumber) => {  
  console.log("message")  
}
```

Q&A

Thank You



