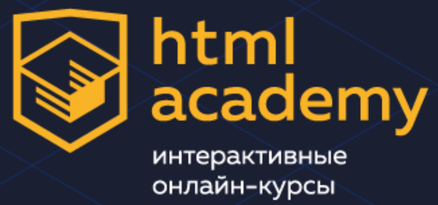

Раздел 0×09

Узкие места JS

План действий на сегодня

План действий на сегодня

– узкие места JS

План действий на сегодня

– узкие места JS

  – WAT

План действий на сегодня

- узкие места JS
  - WAT
  - обязательная точка с запятой

План действий на сегодня

- узкие места JS
  - WAT
  - обязательная точка с запятой
  - подвешивание переменных *(hoisting)*

План действий на сегодня

- узкие места JS
  - WAT
  - обязательная точка с запятой
  - подвешивание переменных *(hoisting)*
  - способы объявления функций и их отличия

Узкие места JS

что вас могут спросить на собеседовании

Узкие места JS

изучайте особенности языка, со стороны может казаться что это баг или тупое поведение

WAT 🐥

https://www.destroyallsoftware.com/talks/wat

https://github.com/ufocoder/javascript.anomaly

WAT

- `[] + []`
- `[] + {}`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + [] === ""`
- `[] + {}`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + [] === ""`
  Плюс говорит о строковой операции, а для строковых операций вызывается `toString`, который переводит массив в пустую строку

- `[] + {}`

- `{} + []`

- `{} + {}`

- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {} === "[object Object]"`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {} === "[object Object]"`
  тот же принцип, что и в первом примере — плюс переводит объекты в строку, поэтому на обоих операторах вызывается `toString`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {}`
- `{} + [] === 0`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {}`
- `{} + [] === 0`
  пустые фигурные скобки трактуются как пустой блок кода, а не как объект, поэтому их можно отбросить. Остается `+[]`: приведение через унарный плюс массива в число. Пустой массив приводится в число как `0`
- `{} + {}`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {}`
- `{} + []`
- `{} + {} === "[object Object][object Object]"`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {}`
- `{} + []`
- `{} + {} === "[object Object][object Object]"`
  одни браузеры считают это сложением двух объектов, в то время как другие — опускают пустой блок кода, оставляя объект, переведенный в число унарным плюсом, что, логично, дает `NaN`
- `Array(3).join("wat" - 1)`

WAT

- `[] + []`
- `[] + {}`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1) === "NaNNaN"`

WAT

- `[] + []`
- `[] + {}`
- `{} + []`
- `{} + {}`
- `Array(3).join("wat" - 1) === "NaNNaN"`
  `join` возвращает строку из элементов массива с разделителем, переданным параметром.
  `"wat" - 1` даст `NaN`, который переводится в строку как `NaN`

```
> '5' - 3
< 2         // слабая типизация + приведение типов = головная боль
> '5' + 3
< '53'      // потому что все мы любим постоянство
> '5' - '4'
< 1         // строка - строка = число. Что?
> '5' + + '5'
< '55'      // окей, допустим
> 'foo' + + 'foo'
< 'fooNaN'  // чудесно!
> '5' + - '2'
< '5-2'     // хорошо
> '5' + - + - - + - - + + - + - + - + - - - '-2'
< '52'      // будем считать, что так и должно быть
> var x = 3;
> '5' + x - x
< 50        // логично
> '5' - x + x
< 5         // к чёрту математику
```

Оператор сравнения

— `0 >= null === true`

— `0 > null || 0 == null === false`

Let `r` be the result of performing abstract relational comparison `lval < rval`. If `r` is `true` or `undefined`, return `false`. Otherwise, return `true`

— `!(0 < null) === !(false) === true`

Превратим массив строк в массив чисел

— `[1, 2, 3].map(parseInt)`

Превратим массив строк в массив чисел

```
— [1, 2, 3].map(parseInt)
  [1, NaN, NaN]
```

Превратим массив строк в массив чисел

```
— [1, 2, 3].map(parseInt)
  [1, NaN, NaN]

— [1, 2, 3].map(function (it, i, arr) {
    parseInt(it, i, arr);
  });
```

Превратим массив строк в массив чисел

```
— [1, 2, 3].map(parseInt)
  [1, NaN, NaN]

— [1, 2, 3].map(function (it, i, arr) {
    parseInt(it, i, arr);
  });
```

Превратим массив строк в массив чисел

```
— [1, 2, 3].map(parseInt)
  [1, NaN, NaN]

— [1, 2, 3].map(function (it, i, arr) {
    parseInt(it, i, arr);
  });
```

Превратим массив строк в массив чисел

- ```
  [1, 2, 3].map(parseInt)
  [1, NaN, NaN]
  ```

```
parseInt(1, 0, [1, 2, 3])
> 1
```

- ```
  [1, 2, 3].map(function (it, i, arr) {
    parseInt(it, i, arr);
  });
  ```

Превратим массив строк в массив чисел

- ```
  [1, 2, 3].map(parseInt)
  [1, NaN, NaN]
  ```

```
parseInt(1, 0, [1, 2, 3])
> 1
```

- ```
  [1, 2, 3].map(function (it, i, arr) {
    parseInt(it, i, arr);
  });
  ```

```
parseInt(2, 1, [1, 2, 3])
> NaN
```

Превратим массив строк в массив чисел

```
[1, 2, 3].map(parseInt)
[1, NaN, NaN]
```

```
[1, 2, 3].map(function (it, i, arr) {
  parseInt(it, i, arr);
});
```

```
parseInt(1, 0, [1, 2, 3])
> 1

parseInt(2, 1, [1, 2, 3])
> NaN

parseInt(3, 2, [1, 2, 3])
> NaN
```

Почему WAT

https://habrahabr.ru/post/137188/

Не стоит искать магию там, где её нет! 🎩🔫

ничего не работает просто так. Всему есть объяснение, поэтому нужно разбираться в основах, чтобы понимать как именно это работает

Еще немного о сравнениях

Интерактивная демонстрация

Необязательные точки с запятой и скобки

Обязательная точка с запятой

точка с запятой в JS необязательна и её можно
не писать совсем. Однако при этом важно понимать,
как JS разбирает такой код

Что выведет в консоль этот код?

```
var a = 1
var b = 1
a
++
b
console.log(a, b)
```

Что выведет в консоль этот код?

```
var a = 1
var b = 1
a
++
b
console.log(a, b) // 1, 2
```

При отсутствии точки с запятой,

JS ставит её тогда, когда он считает, что выражение законченно

Обязательная точка с запятой

```
// Файл module.js
(function () {
  // Модуль module.js
})();
```

Обязательная точка с запятой

```
// Файл one.js
(function () {
  // Модуль one.js
})()

// Файл two.js
(function () {
  // Модуль two.js
})()
```

Обязательная точка с запятой

```
(function () {
  // Модуль one.js
})()(function () {
  // Модуль two.js
})()
```

Обязательная точка с запятой

– Браузер склеит это все в 4 последовательных вызова

– В лучшем случае мы получим ошибку

– В худшем это как-то отработает и мы получим непонятный баг

Обязательная точка с запятой

```
// Файл one.js
;(function () {
  // Модуль one.js
})()

// Файл two.js
;(function () {
  // Модуль two.js
})()
```

Обязательные фигурные скобки

в выражениях *for*, *if*, *while*, *else*, *function*

фигурные скобки *(блоки кода)* могут быть опущены,

если блок состоит только из одной строчки

Что выведет в консоль этот код?

```
var isIvan = false;

var name = 'Пётр';
var surname = 'Петров';
if (isIvan)
  name = 'Иван';
  surname = 'Иванович';

console.log(name, surname);
```

Что выведет в консоль этот код?

```
var isIvan = false;

var name = 'Пётр';
var surname = 'Петров';
if (isIvan)
  name = 'Иван';
  surname = 'Иванович';

console.log(name, surname); // Пётр Иванович
```

Избыточность

правильный код может содержать конструкции, которые не имеют значения или смысла и могут быть безболезненно упрощены

Техника *failfast*

```
var binarySearch = function (array, value) {
  if (!array || array.length === 0) { return -1; }
  if (typeof value !== 'number') { return -1; }

  // ...

  return find(0, array.length);
};
```

failfast

```
var binarySearch = function (array, value) {
  if (!array || array.length === 0) { return -1; }
  if (typeof value !== 'number') { return -1; }

  var find = function (left, right) {
    if (left === right) { return -1; }

    var pivot = Math.floor((right + left) / 2);
    var item = array[pivot];

    if (item === value) { return pivot; }
    if (item > value) { return find(left, pivot); }
    if (item < value) { return find(pivot + 1, right); }
  };

  return find(0, array.length);
};
```

https://ru.wikipedia.org/wiki/Двоичный_поиск

Без *failfast*

```
var binarySearch = function (array, value) {
  if (!array || array.length === 0) {
    return -1;
  } else if (typeof value !== 'number') {
    return -1;
  } else {
    var find = function (left, right) {
      if (left === right) {
        return -1;
      } else {
        var pivot = Math.floor((right + left) / 2);
        var item = array[pivot];
        if (item === value) {
          return pivot;
        } else if (item > value) {
          return find(left, pivot);
        } else if (item < value) {
          return find(pivot + 1, right);
        }
      }
    };
    return find(0, array.length);
  }
}
```

Избыточные проверки

```
var isPositiveNumber = function (myNumber) {
  if (typeof myNumber === 'undefined') {
    throw new Error('Parameter is not defined');
  }

  var myNumber = parseInt(myNumber);

  return myNumber > 0;
};

isPositiveNumber(15);
isPositiveNumber(-30);
```

Избыточные проверки

```
var isPositiveNumber = function (myNumber) {
  if (typeof myNumber === 'undefined') {
    throw new Error('Parameter is not defined');
  }

  var myNumber = parseInt(myNumber);

  return myNumber > 0;
};

isPositiveNumber(15);
isPositiveNumber(-30);
```

«Подвешивание» переменных

интерактивное дело

Прокачивайтесь

– Решайте задачки:
  https://www.htmlacademy.ru — некие интерактивные курсы
  https://www.codewars.com — соревнования
  https://www.enki.com — упражнения

– Не верьте на слово, проверяйте —
  консоль ваш друг, работает в любой вкладке браузера с 2010 года

– Заведите **pet project** —
  небольшой домашний проект, который решает какую-то вашу небольшую задачу:
  раскрашивает страницу, расставляет буквы «ё», считает расходы и т.д.

– Ходите на интервью, приходите к нам наставниками

http://learnjswith.me/javascript-fizzbuzz


KIITOS.