24 января 2023 г.

Преобразование объектов в примитивы

Что произойдёт, если сложить два объекта `obj1 + obj2`, вычесть один из другого `obj1 - obj2` или вывести их на экран, воспользовавшись `alert(obj)`?

JavaScript совершенно не позволяет настраивать, как операторы работают с объектами. В отличие от некоторых других языков программирования, таких как Ruby или C++, мы не можем реализовать специальный объектный метод для обработки сложения (или других операторов).

В случае таких операций, объекты автоматически преобразуются в примитивы, затем выполняется сама операция над этими примитивами, и на выходе мы получим примитивное значение.

Это важное ограничение: результатом `obj1 + obj2` (или другой математической операции) не может быть другой объект!

К примеру, мы не можем создавать объекты, представляющие векторы или матрицы (или достижения или может ещё что-то), складывать их и ожидать в качестве результата «суммированный» объект. Такие архитектурные ходы автоматически оказываются «за бортом».

Итак, поскольку мы технически здесь мало что можем сделать, в реальных проектах нет математики с объектами. Если она всё же происходит, то за редким исключением, это из-за ошибок в коде.

В этой главе мы рассмотрим, как объект преобразуется в примитив и как это можно настроить.

У нас есть две цели:

1. Это позволит нам понять, что происходит в случае ошибок в коде, когда такая операция произошла случайно.
2. Есть исключения, когда такие операции возможны и вполне уместны. Например, вычитание или сравнение дат (`Date` объекты). Мы встретимся с ними позже.

Правила преобразования

В главе Преобразование типов мы рассмотрели правила для числовых, строковых и логических преобразований примитивов. Но мы оставили пробел для объектов. Теперь, когда мы уже знаем о методах и символах, пришло время заполнить этот пробел.

1. Не существует преобразования к логическому значению. В логическом контексте все объекты являются `true`, всё просто. Существует лишь их числовое и строковое преобразование.
2. Числовое преобразование происходит, когда мы вычитаем объекты или применяем математические функции. Например, объекты `Date` (которые будут рассмотрены в главе Дата и время) могут быть вычтены, и результатом `date1 - date2` будет разница во времени между двумя датами.
3. Что касается преобразований к строке – оно обычно происходит, когда мы выводим на экран объект при помощи `alert(obj)` и в подобных контекстах.

Мы можем реализовать свои преобразования к строкам и числам, используя специальные объектные методы.

Теперь давайте углубимся в детали. Это единственный путь для того, чтобы разобраться в нюансах этой темы.

Хинты

Как JavaScript решает, какое преобразование применить?

Существует три варианта преобразования типов, которые происходят в различных ситуациях. Они называются «хинтами», как описано в спецификации:

`"string"`

Для преобразования объекта к строке, когда мы выполняем операцию над объектом, которая ожидает строку, например `alert`:

```
// вывод
alert(obj);

// используем объект в качестве ключа
anotherObj[obj] = 123;
```

`"number"`

Для преобразования объекта к числу, в случае математических операций:

```
// явное преобразование
let num = Number(obj);

// математические (не считая бинарного плюса)
let n = +obj; // унарный плюс
let delta = date1 - date2;

// сравнения больше/меньше
let greater = user1 > user2;
```

Большинство встроенных математических функций также включают в себя такое преобразование.

`"default"`

Происходит редко, когда оператор «не уверен», какой тип ожидать.

Например, бинарный плюс `+` может работать как со строками (объединяя их в одну), так и с числами (складывая их). Поэтому, если бинарный плюс получает объект в качестве аргумента, он использует хинт `"default"` для его преобразования.

Также, если объект сравнивается с помощью `==` со строкой, числом или символом, тоже неясно, какое преобразование следует выполнить, поэтому используется хинт `"default"`.

```
// бинарный плюс использует хинт "default"
let total = obj1 + obj2;

// obj == number использует хинт "default"
if (user == 1) { ... };
```

Операторы сравнения больше/меньше, такие как `<` `>`, также могут работать как со строками, так и с числами. Тем не менее, по историческим причинам, они используют хинт `"number"`, а не `"default"`.

Впрочем на практике, всё немного проще.

Все встроенные объекты, за исключением одного (объект `Date`, который мы рассмотрим позже), реализуют `"default"` преобразование тем же способом, что и `"number"`. И нам следует поступать так же.

Чтобы выполнить преобразование, JavaScript пытается найти и вызвать три следующих метода объекта:

1. Вызвать `obj[Symbol.toPrimitive](hint)` – метод с символьным ключом `Symbol.toPrimitive` (системный символ), если такой метод существует,
2. Иначе, если хинт равен `"string"`
   - попробовать вызвать `obj.toString()` или `obj.valueOf()`, смотря какой из них существует.
3. Иначе, если хинт равен `"number"` или `"default"`
   - попробовать вызвать `obj.valueOf()` или `obj.toString()`, смотря какой из них существует.

Symbol.toPrimitive

Давайте начнём с первого метода. Есть встроенный символ с именем `Symbol.toPrimitive`, который следует использовать для обозначения метода преобразования, вот так:

```
obj[Symbol.toPrimitive] = function(hint) {
  // вот код для преобразования этого объекта в примитив
  // он должен вернуть примитивное значение
  // hint = чему-то из "string", "number", "default"
};
```

Если метод `Symbol.toPrimitive` существует, он используется для всех хинтов, и больше никаких методов не требуется.

Например, здесь объект `user` реализует его:

```
let user = {
  name: "John",
  money: 1000,

  [Symbol.toPrimitive](hint) {
    alert(`hint: ${hint}`);
    return hint == "string" ? `{name: "${this.name}"}` : this.money;
  }
};

// демонстрация результатов преобразований:
alert(user); // hint: string -> {name: "John"}
alert(+user); // hint: number -> 1000
alert(user + 500); // hint: default -> 1500
```

Как мы можем видеть из кода, `user` становится либо строкой со своим описанием, либо суммой денег в зависимости от преобразования. Единый метод `user[Symbol.toPrimitive]` обрабатывает все случаи преобразования.

toString/valueOf

Если нет `Symbol.toPrimitive`, тогда JavaScript пытается найти методы `toString` и `valueOf`:

- Для хинта `"string"` : вызвать метод `toString` , а если он не существует или возвращает объект вместо примитивного значения, то `valueOf` (таким образом, `toString` имеет приоритет при строковом преобразовании).
- Для других хинтов: вызвать метод `valueOf` , а если он не существует или возвращает объект вместо примитивного значения, то `toString` (таким образом, `valueOf` имеет приоритет для математических операций).

Методы `toString` и `valueOf` берут своё начало с древних времён. Это не символы (символов тогда ещё не было), а скорее просто «обычные» методы со строковыми именами. Они предоставляют альтернативный «старомодный» способ реализации преобразования.

Эти методы должны возвращать примитивное значение. Если `toString` или `valueOf` возвращает объект, то он игнорируется (так же, как если бы метода не было).

По умолчанию обычный объект имеет следующие методы `toString` и `valueOf` :

- Метод `toString` возвращает строку `"[object Object]"` .
- Метод `valueOf` возвращает сам объект.

Взгляните на пример:

```
let user = {name: "John"};

alert(user); // [object Object]
alert(user.valueOf() === user); // true
```

Таким образом, если мы попытаемся использовать объект в качестве строки, как например в `alert` или вроде того, то по умолчанию мы увидим `[object Object]` .

Значение по умолчанию `valueOf` упоминается здесь только для полноты картины, чтобы избежать какой-либо путаницы. Как вы можете видеть, он возвращает сам объект и поэтому игнорируется. Не спрашивайте меня почему, это по историческим причинам. Так что мы можем предположить, что его не существует.

Давайте применим эти методы для настройки преобразования.

Для примера, используем их в реализации всё того же объекта `user` . Но уже используя комбинацию `toString` и `valueOf` вместо `Symbol.toPrimitive` :

```
let user = {
  name: "John",
  money: 1000,

  // для хинта равного "string"
  toString() {
    return `{name: "${this.name}"}`;
  },

  // для хинта равного "number" или "default"
  valueOf() {
    return this.money;
  }

};

alert(user); // toString -> {name: "John"}
```

```
alert(+user); // valueOf -> 1000
alert(user + 500); // valueOf -> 1500
```

Как видим, получилось то же поведение, что и в предыдущем примере с `Symbol.toPrimitive`.

Довольно часто нам нужно единое «универсальное» место для обработки всех примитивных преобразований. В этом случае мы можем реализовать только `toString`:

```
let user = {
  name: "John",

  toString() {
    return this.name;
  }
};

alert(user); // toString -> John
alert(user + 500); // toString -> John500
```

В отсутствие `Symbol.toPrimitive` и `valueOf`, `toString` обработает все примитивные преобразования.

Преобразование может вернуть любой примитивный тип

Важная вещь, которую следует знать обо всех методах преобразования примитивов, заключается в том, что они не обязательно возвращают подсказанный хинтом примитив.

Нет никакого контроля над тем, вернёт ли `toString` именно строку, или чтобы метод `Symbol.toPrimitive` возвращал именно число для хинта `"number"`.

Единственное обязательное условие: эти методы должны возвращать примитив, а не объект.

> **Историческая справка**
>
> По историческим причинам, если `toString` или `valueOf` вернёт объект, то ошибки не будет, но такое значение будет проигнорировано (как если бы метода вообще не существовало). Это всё потому, что в древние времена в JavaScript не было хорошей концепции «ошибки».
>
> А вот `Symbol.toPrimitive` уже «четче», этот метод *обязан* возвращать примитив, иначе будет ошибка.

Дальнейшие преобразования

Как мы уже знаем, многие операторы и функции выполняют преобразования типов, например, умножение `*` преобразует операнды в числа.

Если мы передаём объект в качестве аргумента, то в вычислениях будут две стадии:

1. Объект преобразуется в примитив (с использованием правил, описанных выше).
2. Если необходимо для дальнейших вычислений, этот примитив преобразуется дальше.

Например:

```
let obj = {
  // toString обрабатывает все преобразования в случае отсутствия других метод
```

```
  toString() {
    return "2";
  }
};

alert(obj * 2); // 4, объект был преобразован к примитиву "2", затем умножение
```

1. Умножение `obj * 2` сначала преобразует объект в примитив (это строка `"2"`).
2. Затем `"2" * 2` становится `2 * 2` (строка преобразуется в число).

А вот, к примеру, бинарный плюс в подобной ситуации соединил бы строки, так как он совсем не брезгует строк:

```
let obj = {
  toString() {
    return "2";
  }
};

alert(obj + 2); // 22 ("2" + 2), преобразование к примитиву вернуло строку =>
```

Итого

Преобразование объекта в примитив вызывается автоматически многими встроенными функциями и операторами, которые ожидают примитив в качестве значения.

Существует всего 3 типа (хинта) для этого:

- `"string"` (для `alert` и других операций, которым нужна строка)
- `"number"` (для математических операций)
- `"default"` (для некоторых других операторов, обычно объекты реализуют его как `"number"`)

Спецификация явно описывает для каждого оператора, какой ему следует использовать хинт.

Алгоритм преобразования таков:

1. Сначала вызывается метод `obj[Symbol.toPrimitive](hint)`, если он существует,
2. В случае, если хинт равен `"string"`
   - происходит попытка вызвать `obj.toString()` и `obj.valueOf()`, смотря что есть.
3. В случае, если хинт равен `"number"` или `"default"`
   - происходит попытка вызвать `obj.valueOf()` и `obj.toString()`, смотря что есть.

Все эти методы должны возвращать примитив (если определены).

На практике часто бывает достаточно реализовать только `obj.toString()` в качестве универсального метода для преобразований к строке, который должен возвращать удобочитаемое представление объекта для целей логирования или отладки.

Предыдущий урок | Следующий урок

Поделиться

Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>` , для нескольких строк кода — тег `<pre>` , если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen…)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

Сергей Кирилюк
13 часов назад

Если с функциями хоть понимаешь как они работают, то тут..

0 0 Ответить • Поделиться ›

JuranTouran
18 часов назад

Так и не понял, подскажите, что такое хинт?

0 0 Ответить • Поделиться ›

KingNut Academy
день назад

Разбор задач, тестов и всего что поможет при прохождении собеседования на должность frontend developer (html, css, js, ts, react). Заходи к нам в ТГ @interview_masters

0 0 Ответить • Поделиться ›

Джун на фронте
день назад

Приглашаю в **блог JS-прогресса** !

В **тг Джун на фронте** я врываюсь в веб и Учусь делать динамичные сайты, без банальных конструкторов 🤮

0 0 Ответить • Поделиться ›

Saionara Scarsgard
20 дней назад

Решите простенькое задание (которого так не хватает некоторым урокам), ответ найдёте за спойлером:

```
"use_strict";

let clients = {
name: "Evgenii"
```

```
[Symbol.toPrimitive](hint) {
// alert(`hint: ${hint}`);
return hint == "string" ? `name: "${this.name}"` : this.spentMoney;
},

toString() {
```

показать больше

2 0 Ответить • Поделиться ›

Neysel
месяц назад

Еле как понятна только секция с итогом. Хорошо что есть комментарии

0 0 Ответить • Поделиться ›

Андрей Сукач
3 месяца назад

почему когда [null]==0)//true ?

0 0 Ответить • Поделиться ›

Absolem the blue caterpillar ↗ Андрей Сукач
2 месяца назад

потому что [null] преобразуется в 0

0 0 Ответить • Поделиться ›

Sergey ↗ Андрей Сукач
3 месяца назад edited

В данном случае [null] это массив, который эквивалентен пустому массиву [ ], а пустой массив = 0, следовательно [null] == 0 и это истина.
З.Ы. прототипом [null] является Array(0) если ты понимаешь о чем я))

4 0 Ответить • Поделиться ›

Андрей Сукач ↗ Sergey
месяц назад

Кстати да, прототип [null] является Array(0)).Спасибо большое за объяснение.)

0 0 Ответить • Поделиться ›

Absolem the blue caterpillar ↗ Sergey
2 месяца назад

Чет глупость какая-то

0 0 Ответить • Поделиться ›

Alexander
5 месяцев назад

Я правильно понял, что Symbol.toPrimitive, valueOf() и toString() надо ручками запихнуть в свои объекты для их работы? Единственное, что не понятно и прямо не сказано.

4 0 Ответить • Поделиться ›

Inferno → Alexander
4 месяца назад

Всё написано в статье, читайте внимательнее.
"По умолчанию обычный объект имеет следующие методы toString и valueOf:
Метод toString возвращает строку "[object Object]".
Метод valueOf возвращает сам объект."

4 1 Ответить • Поделиться ›

Хоц → Alexander
4 месяца назад edited

Немного вас уточню, valueOf() и toString() по умолчанию работают, valueOf() возвращает сам обьект, а toString() возвращает строку "[object Object]", соответственно если вы хотите что бы они возвращали что-то другое, нужно это прописать. Это было написано в разделе "toString/valueOf".

8 0 Ответить • Поделиться ›

Никита Сумаки → Alexander
4 месяца назад

да

0 1 Ответить • Поделиться ›

Antoinette
5 месяцев назад

Nice.

0 0 Ответить • Поделиться ›

John
5 месяцев назад edited

Статья написана максимально сложно, чтобы было вообще ничего никому не понятно. В общем простыми словами можно объяснить следующее. У нас есть объекты типа obj = { name: "Alex", money:1000 }, так вот приведение к примитивам valueOf() и toString() - в данных примерах действуют следующим образом. Когда мы вызываем alert - объект приводится к строке или числу, вызывается функция(как свойство объекта) toString() или valueOf() в зависимости это строка или число. Далее, строка нам выводит name: Alex, а число выводит 1000. В принципе для console.log можно записать следующим образом

```
let user = {
name: "Alex",
money: 1000,
```

```
// для хинта равного "string"
toString() {
return `{name: "${this.name}"}`;
},

// для хинта равного "number" или "default"
```

показать больше

17 4 Ответить • Поделиться ›

BlackBird → John
4 месяца назад

спасибо! статья правда совсем непонятная.

5 2 Ответить • Поделиться ›

Aptypsy
5 месяцев назад

@**SexMashina** - святой человек. Проскрольте его профиль для объяснений. Всю духоту для таких довнов как я разжевал

14 0 Ответить • Поделиться ›

Павел Грушак → Aptypsy
5 месяцев назад

спасибо за наводку, почитал, поржал, усвоил, толково объяснил)

0 0 Ответить • Поделиться ›

Загрузить ещё комментарии