

Ти Джей Краудер

Новые возможности  
**JavaScript**<sup>®</sup>

Как написать чистый код  
по всем правилам современного языка

УДК 004.43  
ББК 32.973.26-018.1  
К78

T.J. Crowder

JAVASCRIPT: THE NEW TOYS

Copyright © 2020 by Thomas Scott «T.J.» Crowder  
All Rights Reserved. This translation published under license  
with the original publisher John Wiley & Sons, Inc.

**Краудер, Ти Джей.**

К78 Новые возможности JavaScript : как написать чистый код по всем правилам современного языка / Ти Джей Краудер : [перевод с английского М. А. Райтман]. — Москва : Эксмо, 2023. — 640 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-04-159515-9

Перед вами сборник правил написания кода на современном языке JavaScript. На наглядных примерах автор объясняет, как работают последние версии JS, какие приемы в нем можно использовать, чтобы сделать код коротким и чистым, а каких ошибок лучше избегать, чтобы не было багов.

УДК 004.43  
ББК 32.973.26-018.1

ISBN 978-5-04-159515-9

© Райтман М. А., перевод на русский язык, 2023  
© Оформление. ООО «Издательство «Эксмо», 2023

---

<b>ГЛАВА 4. КЛАССЫ</b>	<b>97</b>
Что такое класс?	98
Представление нового синтаксиса класса	98
Добавление конструктора	100
Добавление свойств экземпляра	102
Добавление метода прототипа	102
Добавление статического метода	104
Добавление свойства-аксессуара	105
Вычисляемые имена методов	107
Сравнение с устаревшим синтаксисом	107
Создание подклассов	110
Ключевое слово <code>super</code>	113
Написание конструкторов подклассов	114
Наследование свойств и методов прототипа суперкласса и доступ к ним	115
Наследование статических методов	119
Ключевое слово <code>super</code> в статических методах	120
Методы, возвращающие новые экземпляры	121
Создание подклассов для встроенных компонентов	126
Где доступен <code>super</code>	127
Отказ от <code>Object.prototype</code>	131
Синтаксис <code>new.target</code>	131
Объявления классов в сравнении с выражениями классов	135
Объявления классов	135
Выражения классов	136
Еще не все	137
От старых привычек к новым	137
Использование класса при создании функций конструктора	137
<b>ГЛАВА 5. ОБЪЕКТЫ</b>	<b>139</b>
Вычисляемые имена свойств	139
Стенография свойств	140
Получение и настройка прототипа объекта	141
Метод <code>Object.setPrototypeOf</code>	141
Свойство <code>__proto__</code> в браузерах	142
Буквальное указание имени свойства <code>__proto__</code> в браузерах	143
Синтаксис метода и применение <code>super</code> вне классов	143
Тип данных <code>Symbol</code>	146
Почему же символы?	147
Создание и использование символов	148
Символы не для конфиденциальности	149
Глобальные символы	150

Хорошо известные символы	154
Новые функции объектов	155
Метод <code>Object.assign</code>	155
Метод <code>Object.is</code>	156
Метод <code>Object.values</code>	157
Метод <code>Object.entries</code>	157
Метод <code>Object.fromEntries</code>	158
Функция <code>Object.getOwnPropertySymbols</code>	158
Метод <code>Object.getOwnPropertyDescriptors</code>	158
Метод <code>Symbol.toPrimitive</code>	159
Порядок свойств	161
Синтаксис расширения свойств	163
От старых привычек к новым	164
Использовать вычисляемый синтаксис при создании свойств с динамическими именами	164
Используйте сокращенный синтаксис при инициализации свойства из переменной с тем же именем	165
Используйте метод <code>Object.assign</code> вместо пользовательских функций «extend» или явного копирования всех свойств	165
Используйте синтаксис расширения при создании нового объекта на основе свойств существующего объекта	165
Используйте символ, чтобы избежать коллизии имен	165
Используйте методы <code>Object.getPrototypeOf/setPrototypeOf</code> вместо свойства <code>__proto__</code>	166
Используйте синтаксис метода для методов	166

## ГЛАВА 6. ВОЗМОЖНОСТИ ИТЕРАЦИИ: ИТЕРИРУЕМЫЕ ОБЪЕКТЫ, ИТЕРАТОРЫ, ЦИКЛЫ FOR-OF, ИТЕРАТИВНЫЕ РАСШИРЕНИЯ, ГЕНЕРАТОРЫ

Итерируемые объекты, итераторы, циклы for-of, итерируемое расширение	167
Итераторы и итерируемые объекты	168
Цикл for-of: Неявное использование итератора	168
Явное использование итератора	170
Остановка итерации на ранней стадии	171
Прототип итератора объекта	172
Сделать что-либо итерируемым объектом	175
Итерируемые итераторы	179
Синтаксис итеративного расширения	181
Итераторы, цикл for-of и DOM	182
Функции-генераторы	184
Базовая функция-генератор, просто производящая значения	185
Использование функций-генераторов для создания итераторов	186
Функции-генераторы в качестве методов	188

Использование генератора напрямую	189
Потребление значений генераторами	189
Использование оператора <code>return</code> в функции-генераторе	193
Приоритет оператора <code>yield</code>	194
Методы <code>return</code> и <code>throw</code> : Завершение работы генератора	195
Остановка генератора или итеративного объекта: <code>yield<sup>o</sup></code>	197
От старых привычек к новым	202
Используйте конструкции с итеративными элементами	202
Используйте возможности итеративных коллекций DOM	203
Используйте интерфейсы итераторов и итеративных объектов	203
Используйте синтаксис итеративного расширения в большинстве мест, где вы применяли <code>Function.prototype.apply</code>	203
Используйте генераторы	204
<b>ГЛАВА 7. ДЕСТРУКТУРИЗАЦИЯ</b>	<b>205</b>
Краткий обзор	205
Базовая деструктуризация объекта	206
Базовая (и итеративная) деструктуризация массива	209
Значения по умолчанию	211
Синтаксис <code>Rest</code> в шаблонах деструктуризации	213
Использование отличающихся имен	214
Вычисляемые имена свойств	216
Вложенная деструктуризация	216
Деструктуризация параметров	217
Деструктуризация в циклах	220
От старых привычек к новым	221
Используйте деструктуризацию при получении только некоторых свойств от объекта	221
Используйте деструктуризацию для объектов <code>options</code>	222
<b>ГЛАВА 8. ОБЪЕКТЫ PROMISE</b>	<b>223</b>
Почему же промисы?	223
Основы промисов	224
Краткий обзор	224
Пример	226
Промисы и элементы <code>thenable</code>	228
Использование существующего промиса	229
Метод <code>then</code>	229
Связывание промисов в цепочки	230
Сравнение с обратными вызовами	234
Метод <code>catch</code>	235

Метод <code>finally</code>	237
Метод <code>throw</code> в обработчиках <code>then</code> , <code>catch</code> и <code>finally</code>	241
Метод <code>then</code> с двумя аргументами	243
Добавление обработчиков к уже выполненным промисам	245
Создание промисов	246
Конструктор <code>Promise</code>	247
Метод <code>Promise.resolve</code>	250
Метод <code>Promise.reject</code>	251
Другие служебные методы промисов	252
Метод <code>Promise.all</code>	252
Метод <code>Promise.race</code>	254
Метод <code>Promise.allSettled</code>	254
Метод <code>Promise.any</code>	255
Шаблоны промисов	255
Обрабатывать ошибки или возвращать промис	255
Серии промисов	256
Параллельные промисы	258
Антишаблоны промисов	259
Излишнее выражение <code>new Promise(...)</code>	259
Отсутствие обработки ошибок (или неправильная обработка)	259
Оставление ошибок незамеченными при преобразовании API обратного вызова	260
Неявное преобразование отклонения в успешное выполнение	261
Попытка использовать результаты вне цепочки	262
Использование обработчиков бездействия	262
Неправильное разветвление цепочки	263
Подклассы промисов	264
От старых привычек к новым	265
Используйте промисы вместо успешных/неудачных обратных вызовов	265
<b>ГЛАВА 9. АСИНХРОННЫЕ ФУНКЦИИ, ИТЕРАТОРЫ И ГЕНЕРАТОРЫ</b>	<b>267</b>
Асинхронные функции	267
Создание промисов асинхронными функциями	270
Оператор <code>await</code> использует промисы	271
Стандартная логика становится асинхронной при использовании <code>await</code>	272
Отклонения — это исключения, исключения — это отклонения; выполнение — это результаты, возвращаемые значения — это разрешения	273
Параллельные операции в асинхронных функциях	276
Нет необходимости возвращать <code>await</code>	277
Ловушка Pitfall: Использование асинхронной функции в неожиданном месте	278
Асинхронные итераторы, итерируемые и генераторы	279
Асинхронные итераторы	279

Асинхронные генераторы	283
Выражение <code>for-await-of</code>	285
От старых привычек к новым	286
Используйте асинхронные функции и <code>await</code> вместо явных промисов и <code>then/catch</code>	286
<b>ГЛАВА 10. ШАБЛОНЫ, ПОМЕЧЕННЫЕ ФУНКЦИИ И НОВЫЕ ВОЗМОЖНОСТИ СТРОК</b>	<b>287</b>
Шаблонные литералы	287
Базовая функциональность (Непомеченные шаблонные литералы)	288
Помеченные шаблонные функции (Помеченные шаблонные литералы)	290
Метод <code>String.raw</code>	295
Повторное использование шаблонных литералов	297
Шаблонные литералы и автоматическая вставка точки с запятой	297
Улучшенная поддержка Юникода	297
Юникод, а что такое строка JavaScript?	298
Экранирующая последовательность кодовой точки	300
Метод <code>String.fromCodePoint</code>	300
Метод <code>String.prototype.codePointAt</code>	300
Метод <code>String.prototype.normalize</code>	301
Итерация	303
Новые строковые методы	304
Метод <code>String.prototype.repeat</code>	304
Методы <code>String.prototype.startsWith</code> и <code>String.prototype.endsWith</code>	305
Метод <code>String.prototype.includes</code>	306
Методы <code>String.prototype.padStart</code> и <code>String.prototype.padEnd</code>	306
Методы <code>String.prototype.trimStart</code> и <code>String.prototype.trimEnd</code>	307
Обновления методов <code>match</code> , <code>split</code> , <code>search</code> и <code>replace</code>	307
От старых привычек к новым	309
Используйте шаблонные литералы вместо конкатенации строк (где это уместно)	309
Используйте помеченные функции и шаблонные литералы для DSL вместо пользовательских механизмов заполнения	310
Используйте строковые итераторы	310
<b>ГЛАВА 11. МАССИВЫ</b>	<b>311</b>
Новые методы массивов	311
Метод <code>Array.of</code>	311
Метод <code>Array.from</code>	312
Метод <code>Array.prototype.keys</code>	315
Метод <code>Array.prototype.values</code>	316
Метод <code>Array.prototype.entries</code>	317

Метод <code>Array.prototype.copyWithIn</code>	318
Метод <code>Array.prototype.find</code>	321
Метод <code>Array.prototype.findIndex</code>	322
Метод <code>Array.prototype.fill</code>	322
<i>Общая ловушка Pitfall: Использование объекта в качестве значения заполнения</i>	323
Метод <code>Array.prototype.includes</code>	324
Метод <code>Array.prototype.flat</code>	324
Метод <code>Array.prototype.flatMap</code>	326
Итерация, расширение, деструктуризация	326
Стабильная сортировка массива	326
Типизированные массивы	327
Краткий обзор	327
Основное использование	330
<i>Подробнее о преобразовании значений</i>	331
Объект <code>ArrayBuffer</code> : Хранилище для типизированных массивов	333
Порядковый номер (Порядок байтов)	336
Вид <code>DataView</code> : Необработанный доступ к буферу	337
Совместное использование <code>ArrayBuffer</code> массивами	339
<i>Совместное использование без перекрытия</i>	339
<i>Совместное использование с перекрытием</i>	340
Подклассы типизированных массивов	341
Методы типизированного массива	341
<i>Стандартные методы массива</i>	341
Метод <code>%TypedArray%.prototype.set</code>	343
Метод <code>%TypedArray%.prototype.subarray</code>	343
От старых привычек к новым	344
Используйте <code>find</code> и <code>findIndex</code> для поиска в массивах вместо циклов (где это уместно)	344
Используйте для заполнения массивов <code>Array.fill</code> , а не циклы	344
Используйте <code>readAsArrayBuffer</code> вместо <code>readAsBinaryString</code>	345
<b>ГЛАВА 12. КАРТЫ И МНОЖЕСТВА</b>	<b>347</b>
Коллекции Map или карты	347
Основные операции с картой	348
Равенство ключей	350
Создание карт из итерируемых	351
Итерация содержимого карты	352
Создание подклассов для карты	354
Производительность	355
Множества	355
Основные операции с множеством	356
Создание множеств из итерируемых	357



Итерация содержимого множества	357
Создание подклассов для множества	359
Производительность	359
<b>Слабые карты (WeakMap)</b>	<b>360</b>
Слабые карты не итерируемые	360
Варианты использования и примеры	360
<i>Вариант использования: Закрытая информация</i>	361
<i>Вариант использования: Хранение информации для объектов, находящихся вне вашего контроля</i>	362
Значения, ссылающиеся на ключ	364
<b>Слабые множества (WeakSet)</b>	<b>369</b>
Вариант использования: Отслеживание	370
Вариант использования: Маркировка	371
<b>От старых привычек к новым</b>	<b>372</b>
Используйте карты вместо объектов для карт общего назначения	372
Используйте множества вместо объектов для множеств	372
Используйте слабые карты для хранения личных данных вместо публичных свойств	373
<b>ГЛАВА 13. МОДУЛИ</b>	<b>375</b>
<hr/>	
Введение в модули	375
Основы модулей	376
Спецификатор модуля	378
Базовый именованный экспорт	379
Экспорт по умолчанию	381
Использование модулей в браузерах	383
<i>Скрипты модуля не задерживают синтаксический анализ</i>	384
<i>Атрибут <code>nomodule</code></i>	384
<i>Спецификаторы модулей в Интернете</i>	385
Использование модулей в Node.js	386
<i>Спецификаторы модулей в Node.js</i>	388
<i>Node.js добавляет дополнительные возможности модулей</i>	389
Переименование экспорта	389
Повторный экспорт экспорта из другого модуля	390
Переименование импорта	391
Импорт объекта пространства имен модуля	392
Экспорт объекта пространства имен другого модуля	393
Импорт модуля только из-за побочных эффектов	394
Импорт и экспорт записей	394
Импорт записей	394
Экспорт записей	395
Импорт в режиме реального времени и доступности только для чтения	397
Экземпляры модуля зависят от базы realm	400

Как загружаются модули	400
Получение и синтаксический анализ	402
Создание экземпляра	405
Выполнение	406
Обзор временной мертвой зоны (TDZ)	406
Циклические зависимости и TDZ	407
Обзор синтаксиса импорта/экспорта	408
Разновидности экспорта	408
Разновидности импорта	410
Динамический импорт	411
Динамический импорт модуля	411
Пример динамического модуля	413
Динамический импорт в немодульных скриптах	416
Встряхивание дерева	418
Бандлинг (Объединение)	420
Метаданные импорта	420
Модули воркеров	421
Загрузка веб-воркера в качестве модуля	421
Загрузка воркера Node.js в качестве модуля	422
Воркер находится в собственной базе realm	422
От старых привычек к новым	423
<i>Используйте модули вместо псевдопространств имен</i>	423
<i>Используйте модули вместо обертыивания кода в функции области видимости</i>	424
<i>Используйте модули, чтобы избежать создания мегалитических файлов кода</i>	424
<i>Конвертируйте CJS, AMD и другие модули в ESM</i>	424
<i>Не изобретайте велосипед, используйте хорошо обслуживаемый бандлер</i>	424
<b>ГЛАВА 14. РЕФЛЕКСИЯ — ОБЪЕКТЫ REFLECT И PROXY</b>	<b>425</b>
Объект Reflect	425
Метод <code>Reflect.apply</code>	427
Метод <code>Reflect.construct</code>	427
Метод <code>Reflect.ownKeys</code>	429
Методы <code>Reflect.get</code> и <code>Reflect.set</code>	429
Другие функции Reflect	431
Объект Proxy	431
Пример: Регистрирующий прокси	435
Ловушки прокси	442
<i>Общие возможности</i>	442
<i>Ловушка <code>apply</code></i>	443
<i>Ловушка <code>construct</code></i>	443
<i>Ловушка <code>defineProperty</code></i>	443
<i>Ловушка <code>deleteProperty</code></i>	445
<i>Ловушка <code>get</code></i>	446

<i>Ловушка</i> <code>getOwnPropertyDescriptor</code>	447
<i>Ловушка</i> <code>getPrototypeOf</code>	448
<i>Ловушка</i> <code>has</code>	449
<i>Ловушка</i> <code>isExtensible</code>	449
<i>Ловушка</i> <code>ownKeys</code>	449
<i>Ловушка</i> <code>preventExtensions</code>	450
<i>Ловушка</i> <code>set</code>	451
<i>Ловушка</i> <code>setPrototypeOf</code>	451
Пример: Скрытие свойств	452
Отключаемые прокси	456
От старых привычек к новым	456
Используйте прокси, а не полагайтесь на потребляющий код, чтобы не изменять объекты API	457
Используйте прокси для отделения кода реализации от инструментального кода	457

---

## ГЛАВА 15. ОБНОВЛЕНИЯ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ 459

Свойство <code>flags</code>	459
Новые флаги	460
Липкий флаг ( <code>y</code> )	460
Флаг Юникода ( <code>u</code> )	461
Флаг «все точки» ( <code>s</code> )	461
Именованные группы захвата	462
Основная функциональность	462
Обратные ссылки	466
Заменяющие токены	467
Утверждения ретроспективной проверки	467
Позитивная ретроспективная проверка	468
Негативная ретроспективная проверка	469
Жадность проявляется в принципе справа налево в ретроспективных проверках	469
Ссылки и нумерация групп захвата	470
Функциональные возможности Юникода	471
Экранирование кодовой точки	471
Экранирование свойства Юникода	472
От старых привычек к новым	476
Используйте липкий флаг ( <code>y</code> ) вместо создания подстрок и использования « <code>^</code> » при синтаксическом анализе	476
Используйте флаг «все точки» ( <code>s</code> ) вместо использования обходных путей для сопоставления всех символов (включая разрывы строк).	477
Используйте именованные группы захвата вместо анонимных	477
Используйте ретроспективные проверки вместо различных обходных путей	478
Используйте экранирование кодовой точки вместо суррогатных пар в регулярных выражениях	478
Используйте шаблоны Юникода вместо обходных путей	478

<b>ГЛАВА 16. СОВМЕСТНО ИСПОЛЬЗУЕМАЯ ПАМЯТЬ</b>	<b>479</b>
Введение	479
Здесь водятся драконы!	480
Поддержка браузера	481
Основы совместно используемой памяти	482
Критические секции, блокировки и условные переменные	483
Создание совместно используемой памяти	484
Совместно используется память, но не объекты	489
Условия гонки, вышедшие из строя хранилища, устаревшие значения, значения с разрывами, тиринг и многое другое	490
Объект <code>Atoms</code>	492
Низкоуровневые возможности объекта <code>Atoms</code>	495
Использование <code>Atoms</code> для приостановки и возобновления потоков	497
Пример совместно используемой памяти	498
Здесь водятся драконы! (Снова)	519
От старых привычек к новым	525
Используйте совместно используемые блоки вместо многократного обмена большими блоками данных	525
Используйте <code>Atoms.wait</code> и <code>Atoms.notify</code> вместо разделения заданий воркеров для поддержки цикла событий (при необходимости)	525
<b>ГЛАВА 17. РАЗЛИЧНЫЕ АСПЕКТЫ</b>	<b>527</b>
Тип данных <code>BigInt</code>	527
Создание значения типа <code>BigInt</code>	529
Явное и неявное преобразование	530
Производительность	531
Массивы <code>BigInt64Array</code> и <code>BigUint64Array</code>	531
Служебные функции	531
Новые целочисленные литералы	532
Двоичные целочисленные литералы	532
Восьмеричные целочисленные литералы, попытка № 2	533
Новые математические методы	534
Общие математические функции	534
Поддержка низкоуровневых математических функций	535
Оператор возведения в степень ( <sup>**</sup> )	535
Изменения в <code>Date.prototype.toString</code>	537
Изменения в <code>Function.prototype.toString</code>	538
Дополнения конструктора <code>Number</code>	538
«Безопасные» целые числа	538
Константы <code>Number.MAX_SAFE_INTEGER</code> и <code>Number.MIN_SAFE_INTEGER</code>	539
Метод <code>Number.isSafeInteger</code>	540

Метод <code>Number.isInteger</code>	540
Методы <code>Number.isFinite</code> и <code>Number.isNaN</code>	540
Методы <code>Number.parseInt</code> и <code>Number.parseFloat</code>	541
Свойство <code>Number.EPSILON</code>	541
Символ <code>Symbol.isConcatSpreadable</code>	541
Различные хитрости синтаксиса	542
Оператор нулевого слияния	543
Опциональная цепочка	543
Необязательные привязки <code>catch</code>	546
Разрывы строк Юникода в JSON	546
Правильно сформированный JSON из метода <code>JSON.stringify</code>	546
Различные стандартные библиотеки/глобальные дополнения	547
Метод <code>Symbol.hasInstance</code>	547
Свойство <code>Symbol.unscopables</code>	547
Объект <code>globalThis</code>	549
Свойство <code>description</code> символа	549
Метод <code>String.prototype.matchAll</code>	550
Приложение Б: Возможности, доступные только для браузера	550
HTML-подобные комментарии	551
Хитрости регулярного выражения	552
Расширение управляющего символа экранирования ( <code>\cX</code> )	552
Допуск недопустимых последовательностей	553
Метод <code>RegExp.prototype.compile</code>	553
Дополнительные встроенные свойства	553
Дополнительные свойства объекта	554
Дополнительные строковые методы	555
Различные фрагменты свободного или неясного синтаксиса	555
Когда же <code>document.all</code> есть... или нет?	557
Оптимизация хвостового вызова	558
От старых привычек к новым	561
Используйте двоичные литералы	561
Используйте новые математические функции вместо различных математических обходных путей	562
Используйте оператор нулевого слияния для значений по умолчанию	562
Используйте опциональную цепочку вместо проверок <code>&amp;&amp;</code>	562
Уберите привязку ошибки ( <code>e</code> ) из <code>"catch (e)"</code> , если она не используется	562
Используйте оператор возведения в степень ( <code>°°</code> ) вместо метода <code>Math.pow</code>	563
<b>ГЛАВА 18. ГРЯДУЩИЕ ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КЛАССА</b>	<b>565</b>
Публичные и приватные поля класса, методы и акцессоры	565
Определения публичного поля (свойства)	566

Приватные поля	572
Приватные методы и акцессоры экземпляра	579
Приватные методы	579
Приватные акцессоры	584
Публичные статические поля, приватные статические поля и приватные статические методы	585
Публичные статические поля	585
Приватные статические поля	586
Приватные статические методы	586
От старых привычек к новым	587
Используйте определения свойств вместо создания свойств в конструкторе (где это уместно)	587
Используйте приватные поля вместо префиксов (где это уместно)	588
Используйте приватные методы вместо функций вне класса для приватных операций	588
<b>ГЛАВА 19. ВЗГЛЯД В БУДУЩЕЕ...</b>	<b>591</b>
Оператор <code>await</code> верхнего уровня	592
Обзор и примеры использования	592
Пример	594
Обработка ошибок	599
Слабые ссылки и обратные вызовы очистки	600
Слабые ссылки	601
Обратные вызовы очистки	604
Индексы соответствия <code>RegExp</code>	609
Метод <code>String.prototype.replaceAll</code>	611
Выражение <code>Atoms.asyncWait</code>	612
Различные хитрости синтаксиса	613
Числовые разделители	613
Поддержка <code>Hashbang</code>	614
Осуждаемые устаревшие возможности <code>RegExp</code>	614
Спасибо, что прочитали!	615
<b>ПРИЛОЖЕНИЕ. ФАНТАСТИЧЕСКИЕ ВОЗМОЖНОСТИ И ГДЕ ОНИ ОБИТАЮТ</b>	<b>617</b>
Функциональные возможности в алфавитном порядке	617
Новые положения	623
Новый синтаксис, ключевые слова, операторы, циклы и тому подобное	624
Новые литеральные формы	626
Дополнения и изменения стандартной библиотеки	626
Прочее	629
<b>АЛФАВИТНЫЙ УКАЗАТЕЛЬ</b>	<b>631</b>