[Исключения: throw/catch/finally](https://javascript.ru/basic/syntax-switch-for#isklyucheniya:-throw-catch-finally)

Работа с исключениями в javascript организована в типичном для языка стиле вседозволенности.

Абсолютно любой объект можно бросить в виде исключения:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | ... | |

|  |  |  |
| --- | --- | --- |
| 3 | throw {message: "Ого!"} | |
| 4 | .. |

|  |  |
| --- | --- |
| 5 | } catch (e) { |
| 6 | alert("Ага, попался!") | |

|  |  |
| --- | --- |
| 7 | } |

Обычно бросают, все же, не простые объекты, а потомки встроенного класса Error:

|  |  |
| --- | --- |
| throw new Error("connection down, server timeout") | |
| // или сделать наследника Error: |

|  |
| --- |
| throw new ConnectionError("server timeout") |

[Встроенный класс Error](https://javascript.ru/basic/syntax-switch-for#vstroennyy-klass-error)

Встроенный класс Error удобен для унификации проверки "a instanceof Error" и интероперабельности с другими библиотеками, так как ничего полезного в нем часто нет.

В Firefox у объекта Error есть полный бэктрейс в свойстве stack и еще масса полезных свойств:

|  |  |
| --- | --- |
| // так можно быстро узнать стек без отладчика | |
| alert( (new Error()).stack ) |

В IE тоже есть некоторые полезные свойства, а в Opera/Safari - на момент написания только message:

|  |
| --- |
| var e = new Error("hi") |
| alert(e.message)  // => "hi" | |

Поэтому, например, у меня созданы потомки (синтаксис фреймворка dojo):

|  |  |
| --- | --- |
| 01 | // базовый класс для ошибок, со стеком (Firefox) |
| 02 | // сообщением message и дополнительной информацией extra | |

|  |  |
| --- | --- |
| 03 | dojo.declare( |
| 04 | "dojo.Error", | |

|  |  |
| --- | --- |
| 05 | Error, |
| 06 | function(message, extra) { | |

|  |  |  |
| --- | --- | --- |
| 07 | this.message = message; | |
| 08 | this.extra = extra; |

|  |  |  |
| --- | --- | --- |
| 09 | this.stack = (new Error()).stack; | |
| 10 | } |

|  |  |
| --- | --- |
| 11 | ) |
| 12 | // Потомок, имя ошибки хранится в name | |

|  |  |  |
| --- | --- | --- |
| 13 | // формат: dojo.declare(Имя, Родитель, Конструктор) | |
| 14 | dojo.declare( |

|  |  |  |
| --- | --- | --- |
| 15 | "dojo.CommunicationError", | |
| 16 | dojo.Error, |

|  |  |
| --- | --- |
| 17 | function() { |
| 18 | this.name="CommunicationError"; | |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 | ) |

|  |  |
| --- | --- |
| 21 |  |
| 22 | // еще потомок для примера | |

|  |  |
| --- | --- |
| 23 | dojo.declare( |
| 24 | "dojo.LockedError", | |

|  |  |
| --- | --- |
| 25 | dojo.Error, |
| 26 | function() { | |

|  |  |  |
| --- | --- | --- |
| 27 | this.name="LockedError"; | |
| 28 | } |

|  |  |
| --- | --- |
| 29 | ) |
| 30 | ... | |

[Ловля конкретного исключения](https://javascript.ru/basic/syntax-switch-for#lovlya-konkretnogo-isklyucheniya)

При перехвате исключений редко нужно перехватывать все подряд. Обычно - надо перехватить определенный класс исключений. Стандартный оператор catch такого не умеет, поэтому полный код обработки будет выглядить так:

|  |  |
| --- | --- |
| 01 | try { |
| 02 | … код ... | |

|  |  |
| --- | --- |
| 03 | } catch(e) { |
| 04 | // ловим нужное исключение | |

|  |  |  |
| --- | --- | --- |
| 05 | if (e instanceof ConnectionError) { | |
| 06 | // обрабатываем его |

|  |  |  |
| --- | --- | --- |
| 07 | … reconnect … | |
| 08 | } else { |

|  |  |  |
| --- | --- | --- |
| 09 | // пробрасываем незнакомое исключение дальше | |
| 10 | throw e |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  | |

|  |  |
| --- | --- |
| 13 | } finally { |
| 14 | // блок finally выполняется всегда, | |

|  |  |  |
| --- | --- | --- |
| 15 | // вне зависимости - было исключение или нет | |
| 16 | … notifyUser() .. |

|  |  |
| --- | --- |
| 17 | } |

В этом примере также присутствует блок finally, взятый в javascript из java. В стандартной схеме try..catch..finally, код из блока finally выполняется либо после try, если эксепшна нет, либо после catch, даже если эксепшн выпал наружу.

Короче говоря, код в finally выполнится при любом результате работы try/catch, и туда удобно ставить всякие очистки, уведомления о конце процесса и т.п.

# Свои ошибки, наследование от Error

Когда мы работаем с внешними данными, возможны самые разные ошибки.

Если приложение сложное, то ошибки естественным образом укладываются в иерархию, разобраться в которой помогает instanceof.

## [Свой объект ошибки](https://learn.javascript.ru/oop-errors" \l "%D1%81%D0%B2%D0%BE%D0%B9-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82-%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8)

Для примера создадим функцию readUser(json), которая будет разбирать JSON с данными посетителя. Мы его получаем с сервера – может, нашего, а может – чужого, в общем – желательно проверить на ошибки. А может, это даже и не JSON, а какие-то другие данные – не важно, для наглядности поработаем с JSON.

Пример json на входе в функцию: { "name": "Вася", "age": 30 }.

В процессе работы readUser возможны различные ошибки. Одна – очевидно, SyntaxError – если передан некорректный JSON.

Но могут быть и другие, например PropertyError – эта ошибка будет возникать, если в прочитанном объекте нет свойства name или age.

Реализуем класс PropertyError:

function PropertyError(property) {

Error.call(this, property) ;

this.name = "PropertyError";

this.property = property;

this.message = "Ошибка в свойстве " + property;

if (Error.captureStackTrace) {

Error.captureStackTrace(this, PropertyError);

} else {

this.stack = (new Error()).stack;

}

}

PropertyError.prototype = Object.create(Error.prototype);

В этом коде вы можете видеть ряд важных деталей, важных именно для ошибок:

**name – имя ошибки.**

Должно совпадать с именем функции.

**message – сообщение об ошибке.**

Несмотря на то, что PropertyError наследует от Error (последняя строка), конструктор у неё немного другой. Он принимает не сообщение об ошибке, а название свойства property, ну а сообщение генерируется из него.

В результате в объекте ошибки есть как стандартное свойство message, так и более точное property.

Это частая практика – добавлять в объект ошибки свойства, которых нет в базовых объектах Error, более подробно описывающие ситуацию для данного класса ошибок.

**stack – стек вызовов, которые в итоге привели к ошибке.**

У встроенных объектов Error это свойство есть автоматически, вот к примеру:

function f() {

alert( new Error().stack );

}

f(); // выведет список вложенных вызовов, с номерами строк, где они были сделаны

Если же объект ошибки делаем мы, то «по умолчанию» такого свойства у него не будет. Нам нужно как-то самим узнавать последовательность вложенных вызовов на текущий момент. Однако удобного способа сделать это в JavaScript нет, поэтому мы поступаем хитро и копируем его из нового объекта new Error, который генерируем тут же.

В V8 (Chrome, Opera, Node.JS) есть нестандартное расширение [Error.captureStackTrace](https://github.com/v8/v8/wiki/Stack-Trace-API#stack-trace-collection-for-custom-exceptions), которое позволяет получить стек.

Это делает строка из кода выше:

Error.captureStackTrace(this, PropertyError);

Такой вызов записывает в объект this (текущий объект ошибки) стек вызовов, ну а второй аргумент – вообще не обязателен, но если есть, то говорит, что при генерации стека нужно на этой функции остановиться. В результате в стеке будет информация о цепочке вложенных вызовов вплоть до вызова PropertyError.

То есть, будет последовательность вызовов до генерации ошибки, но не включая код самого конструктора ошибки, который, как правило, не интересен. Такое поведение максимально соответствует встроенным ошибкам JavaScript.

**Конструктор родителя здесь не обязателен**

Обычно, когда мы наследуем, то вызываем конструктор родителя. В данном случае вызов выглядит как Error.call(this, message).

Строго говоря, этот вызов здесь не обязателен. Встроенный конструктор Error ничего полезного не делает, даже свойство this.message (не говоря уже о name и stack) не назначает. Единственный возможный смысл его вызова – он ставит специальное внутреннее свойство [[ErrorData]], которое выводится в toString и позволяет увидеть, что это ошибка. Поэтому по стандарту вызывать конструктор Error при наследовании в таких случаях рекомендовано.

## [instanceof + try…catch = ♡](https://learn.javascript.ru/oop-errors" \l "instanceof-try-catch)

Давайте теперь используем наш новый класс для readUser:

// Объявление

function PropertyError(property) {

this.name = "PropertyError";

this.property = property;

this.message = "Ошибка в свойстве " + property;

if (Error.captureStackTrace) {

Error.captureStackTrace(this, PropertyError);

} else {

this.stack = (new Error()).stack;

}

}

PropertyError.prototype = Object.create(Error.prototype);

// Генерация ошибки

function readUser(data) {

var user = JSON.parse(data);

if (!user.age) {

throw new PropertyError("age");

}

if (!user.name) {

throw new PropertyError("name");

}

return user;

}

// Запуск и try..catch

try {

var user = readUser('{ "age": 25 }');

} catch (err) {

if (err instanceof PropertyError) {

if (err.property == 'name') {

// если в данном месте кода возможны анонимы, то всё нормально

alert( "Здравствуйте, Аноним!" );

} else {

alert( err.message ); // Ошибка в свойстве ...

}

} else if (err instanceof SyntaxError) {

alert( "Ошибка в синтаксисе данных: " + err.message );

} else {

throw err; // неизвестная ошибка, не знаю что с ней делать

}

}

Всё работает – и наша ошибка PropertyError и встроенная SyntaxError корректно генерируются, перехватываются, обрабатываются.

Обратим внимание на проверку типа ошибки в try..catch. Оператор instanceof проверяет класс с учётом наследования. Это значит, что если мы в дальнейшем решим создать новый тип ошибки, наследующий от PropertyError, то проверка err instanceof PropertyError для класса-наследника тоже будет работать. Код получился расширяемым, это очень важно.

## [Дальнейшее наследование](https://learn.javascript.ru/oop-errors" \l "%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5%D0%B9%D1%88%D0%B5%D0%B5-%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

PropertyError – это просто общего вида ошибка в свойстве. Создадим ошибку PropertyRequiredError, которая означает, что свойства нет.

Это подвид PropertyError, так что унаследуем от неё. Общий вид конструктора-наследника – стандартный:

function PropertyRequiredError(property) {

// вызываем конструктор родителя и передаём текущие аргументы

PropertyError.apply(this, arguments);

...

}

Достаточно ли в наследнике просто вызвать конструктор родителя? Увы, нет.

Если так поступить, то свойство this.name будет некорректным, да и Error.captureStackTrace тоже получит неправильную функцию вторым параметром.

Можно ли как-то поправить конструктор родителя, чтобы от него было проще наследовать?

Для этого нужно убрать из него упоминания о конкретном классе PropertyError, чтобы сделать код универсальным. Частично – это возможно. Как мы помним, существует свойство constructor, которое есть в prototype по умолчанию, и которое мы можем намеренно сохранить при наследовании.

Исправим родителя PropertyError для более удобного наследования от него:

function PropertyError(property) {

this.name = "PropertyError";

this.property = property;

this.message = "Ошибка в свойстве " + property;

if (Error.captureStackTrace) {

Error.captureStackTrace(this, this.constructor); // (\*)

} else {

this.stack = (new Error()).stack;

}

}

PropertyError.prototype = Object.create(Error.prototype);

PropertyError.prototype.constructor = PropertyError;

В строке (\*) вместо ссылки на PropertyError используем constructor чтобы получить именно конструктор для текущего объекта. В наследнике там будет PropertyRequiredError, как и задумано.

Мы убрали одну жёсткую привязку к PropertyError, но со второй (this.name), увы, сложности. Оно должно содержать имя ошибки, то есть, имя её функции-конструктора. Его можно получить через this.name = this.constructor.name, но в IE11- это работать не будет.

Если поддерживать IE11-, то тут уж придётся в наследнике его записывать вручную.

Полный код для наследника:

function PropertyRequiredError(property) {

PropertyError.apply(this, arguments);

this.name = 'PropertyRequiredError';

this.message = 'Отсутствует свойство ' + property;

}

PropertyRequiredError.prototype = Object.create(PropertyError.prototype);

PropertyRequiredError.prototype.constructor = PropertyRequiredError;

var err = new PropertyRequiredError("age");

// пройдёт проверку

alert( err instanceof PropertyError ); // true

Здесь заодно и message в наследнике было перезаписано на более точное. Если хочется избежать записи и перезаписи, то можно оформить его в виде геттера через Object.defineProperty.

## [Итого](https://learn.javascript.ru/oop-errors" \l "%D0%B8%D1%82%D0%BE%D0%B3%D0%BE)

* Чтобы наследовать от ошибок Error, нужно самостоятельно позаботиться о name, message и stack.
* Благодаря тому, что instanceof поддерживает наследование, удобно организуются проверки на нужный тип. В иерархию ошибок можно в любой момент добавить новые классы, с понятным кодом и предсказуемым поведением.

Чтобы создавать наследники от Error было проще, можно создать класс CustomError, записать в него универсальный код, наподобие PropertyError и далее наследовать уже от него:

// общего вида "наша" ошибка

function CustomError(message) {

this.name = "CustomError";

this.message = message;

if (Error.captureStackTrace) {

Error.captureStackTrace(this, this.constructor);

} else {

this.stack = (new Error()).stack;

}

}

CustomError.prototype = Object.create(Error.prototype);

CustomError.prototype.constructor = CustomError;

// наследник

function PropertyError(property) {

CustomError.call(this, "Ошибка в свойстве " + property)

this.name = "PropertyError";

this.property = property;

}

PropertyError.prototype = Object.create(CustomError.prototype);

PropertyError.prototype.constructor = PropertyError;

// и ещё уровень

function PropertyRequiredError(property) {

PropertyError.call(this, property);

this.name = 'PropertyRequiredError';

this.message = 'Отсутствует свойство ' + property;

}

PropertyRequiredError.prototype = Object.create(PropertyError.prototype);

PropertyRequiredError.prototype.constructor = PropertyRequiredError;

// использование

var err = new PropertyRequiredError("age");

// пройдёт проверку

alert( err instanceof PropertyRequiredError ); // true

alert( err instanceof PropertyError ); // true

alert( err instanceof CustomError ); // true

alert( err instanceof Error ); // true