**Обработка ошибок**

## try catch finally

Конструкция try..catch...finally позволяет обрабатывать ошибки во время исполнения кода. Она позволяет запустить код и перехватить ошибки, которые могут в нём возникнуть. Синтаксис:

**try** {

// исполняем код

} **catch(err**) {

// если случилась ошибка, прыгаем сюда

// err - это объект ошибки

} **finally** {

// выполняется всегда после try/catch

}

Секций **catch** или **finally** может не быть, то есть более короткие конструкции **try..catch** и **try..finally** также корректны.

Блок finally срабатывает при любом выходе из try..catch, в том числе и return.

function func() {

try {

return 1;

} catch (e) {

/\* ... \*/

} finally {

alert( 'finally' );

}

}

alert( func() ); // сначала срабатывает alert из finally, а затем 1

Чтобы try..catch работал, код должен быть **выполнимым**. Другими словами, это должен быть синтаксически корректный JavaScript-код. Ошибки, которые возникают во время фазы чтения, называются ошибками парсинга. Их нельзя обработать (изнутри этого кода), потому что движок не понимает код. Таким образом, try..catch может обрабатывать только ошибки, которые возникают в корректном коде. Такие ошибки называют «ошибками во время выполнения», а иногда «исключениями».

Исключение, которое произойдёт в коде, запланированном «на будущее», например в setTimeout, try..catch не поймает:

try {

**setTimeout**(function() {

noSuchVariable; // скрипт упадёт тут

}, 1000);

} catch (e) {

alert( "не сработает" );

}

Чтобы поймать исключение внутри запланированной функции, try..catch должен находиться внутри самой этой функции

## Объект ошибки

В JavaScript есть множество встроенных конструкторов для стандартных ошибок: **Error**, **SyntaxError**, **ReferenceError**, **TypeError** и другие. Объекты для всех встроенных ошибок содержат следующие свойства:

* **message** – понятное человеку сообщение.
* **name** – строка с именем ошибки (**имя конструктора ошибки**).
* **stack** (нестандартное, но хорошо поддерживается) – стек на момент ошибки.

// "Псевдокод" встроенного класса Error, определённого самим JavaScript

class Error {

constructor(message) {

this.message = message;

this.name = "Error"; // (разные имена для разных встроенных классов ошибок)

this.stack = <стек вызовов>; // нестандартное свойство, но обычно поддерживается

}

Если объект ошибки не нужен, мы можем пропустить его, используя catch { вместо catch(err)}.

## Генерация собственных ошибок. Проброс исключений

Мы можем также генерировать собственные ошибки, используя оператор throw. Аргументом throw может быть что угодно, но обычно это объект ошибки, наследуемый от встроенного класса Error.

*Проброс исключения* – это очень важный приём обработки ошибок: блок catch обычно ожидает и знает, как обработать определённый тип ошибок, поэтому он должен пробрасывать дальше ошибки, о которых он не знает.

let json = '{ "age": 30 }'; // данные неполны

try {

let user = JSON.parse(json); // <-- выполнится без ошибок

if (!user.name) {

throw new SyntaxError("Данные неполны: нет имени"); // (\*)

}

alert( user.name );

} catch(e) {

alert( "JSON Error: " + e.message ); // JSON Error: Данные неполны: нет имени

}

По своей природе catch получает ***все*** свои ошибки из try. Он может получать неожиданные ошибки. К счастью, мы можем выяснить, какую ошибку мы получили, например, по её свойству **name.** Есть простое правило: Блок catch должен обрабатывать только те ошибки, которые ему известны, и «пробрасывать» все остальные.

Техника «проброс исключения» выглядит так:

* Блок catch получает все ошибки.
* В блоке catch(err) {...} мы анализируем объект ошибки err.
* Если мы не знаем как её обработать, тогда делаем throw err.

Не пойманные ошибки могут быть пойманы с помощью **ещё одного уровня** try..catch

**«глобальный» обработчик ошибок**

Даже если у нас нет try..catch, большинство сред позволяют настроить «глобальный» обработчик ошибок, чтобы ловить ошибки, которые «выпадают наружу». В браузере это window.onerror.

<script>

window.onerror = function(**message**, **url**, **line**, **col**, **error**) {

alert(`${message} в ${line}:${col} на ${url}`);

};

function readData() {

badFunc(); // Ой, что-то пошло не так!

}

readData(); //Uncaught ReferenceError: badFunc is not defined в 6:3 на https://127.0.0.1:5501/1.js

</script>

Роль глобального обработчика window.onerror обычно заключается не в восстановлении выполнения скрипта – это скорее всего невозможно в случае программной ошибки, а в отправке сообщения об ошибке разработчикам.

Существуют также веб-сервисы, которые предоставляют логирование ошибок для таких случаев, такие как [https://errorception.com](https://errorception.com/) или [http://www.muscula.com](http://www.muscula.com/).

Пользовательские ошибки, расширение Error

Когда что-то разрабатываем, то нам часто необходимы собственные классы ошибок для разных вещей, которые могут пойти не так в наших задачах. Наши ошибки должны поддерживать базовые свойства, такие как message, name и, желательно, stack. Но также они могут иметь свои собственные свойства. JavaScript позволяет вызывать throw с любыми аргументами, то есть технически наши классы ошибок не нуждаются в наследовании от Error. Но если использовать наследование, то появляется возможность идентификации объектов ошибок посредством obj instanceof Error. Так что лучше **применять наследование**.

class ValidationError extends Error {

constructor(message) {

super(message);

this.name = "ValidationError";

}

}

class **PropertyRequiredError extends ValidationError** { //для отсутствующих свойств

constructor(property) {

super("Нет свойства: " + property);

this.name = "PropertyRequiredError";

this.property = property;

}

}

// Применение

function readUser(json) {

let user = JSON.parse(json);

if (!user.age) {

throw new PropertyRequiredError("age");

}

if (!user.name) {

throw new PropertyRequiredError("name");

}

return user;

}

// Рабочий пример с try..catch

try {

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

} catch (err) {

if (err instanceof ValidationError) {

alert("Неверные данные: " + err.message); // Неверные данные: Нет свойства: name

alert(err.name); // PropertyRequiredError

alert(err.property); // name

} else if (err instanceof SyntaxError) {

alert("Ошибка синтаксиса JSON: " + err.message);

} else {

throw err; // неизвестная ошибка, повторно выбросит исключение

}