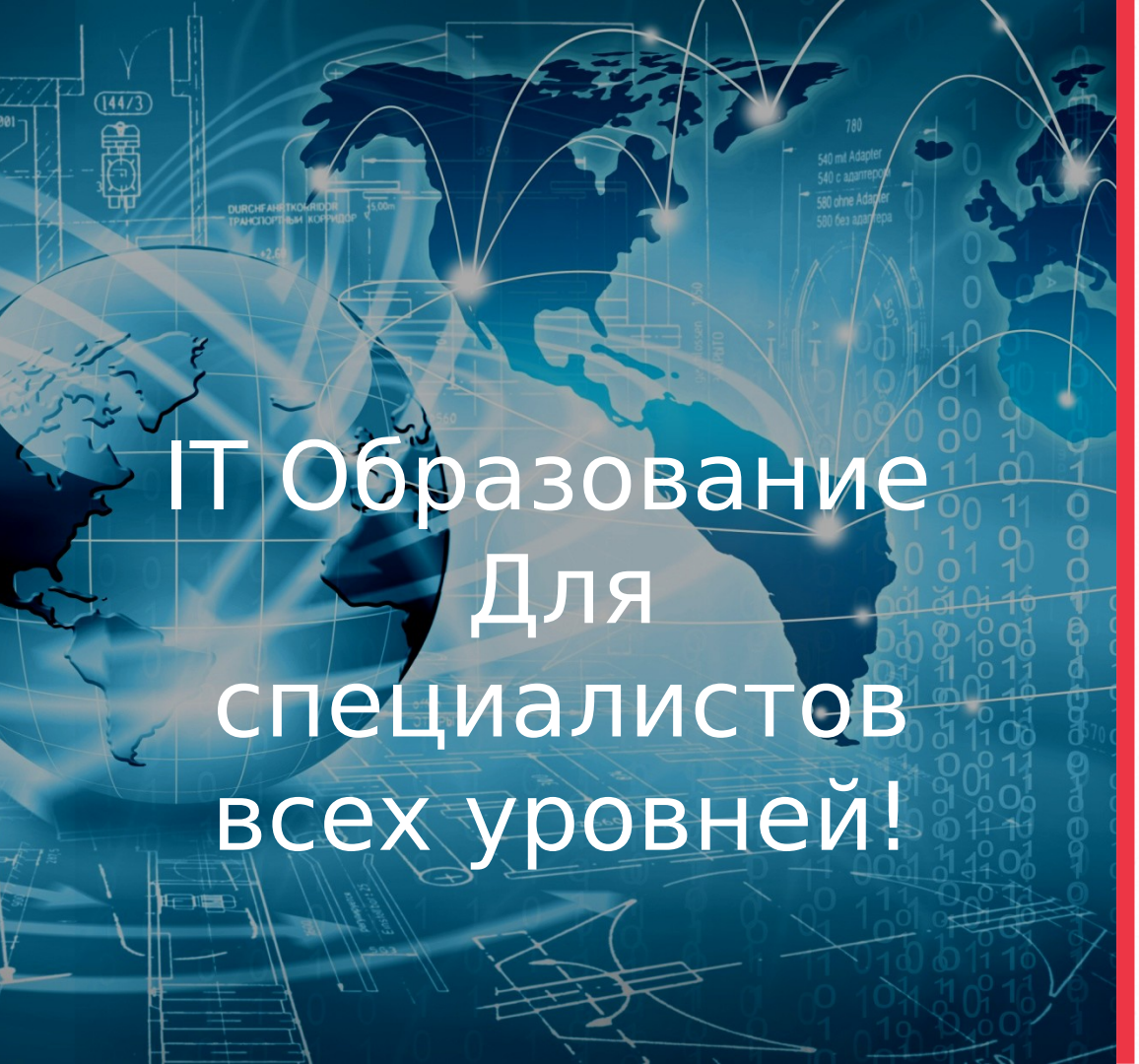


IT Образование Для специалистов всех уровней!

Education
Academy

WWW.ITEDUCATE.COM.UA

JavaScript Pro

Try Catch

Testing

Перехват ошибок

Перехват ошибок

0.1 + 0.2 === 0.3 → false

Конструкция try..catch состоит из двух основных блоков: try, и затем catch

```
try {

  // код ...

} catch (err) {

  // обработка ошибки

}
```

Работает она так:
Выполняется код внутри блока try.
Если в нём ошибок нет, то блок catch(err) игнорируется, то есть выполнение доходит до конца try и потом прыгает через catch.
Если в нём возникнет ошибка, то выполнение try на ней прерывается, и управление прыгает в начало блока catch(err).
При этом переменная err (можно выбрать и другое название) будет содержать объект ошибки с подробной информацией о произошедшем.

Таким образом, при ошибке в try скрипт не «падает», и мы получаем возможность обработать ошибку внутри catch.

Try&Catch

try..catch подразумевает, что код синтаксически верен. Если грубо нарушена структура кода, например не закрыта фигурная скобка или где-то стоит лишняя запятая, то никакой try..catch здесь не поможет. Такие ошибки называютсясинтаксическими, интерпретатор не может понять такой код.

try..catch работает только в синхронном коде

Ошибку, которая произойдёт в коде, запланированном «на будущее», например, в setTimeout, try..catch не поймает

Объект ошибки

name

Тип ошибки. Например, при обращении к несуществующей переменной: "ReferenceError".

message

Текстовое сообщение о деталях ошибки.

stack

Везде, кроме IE8-, есть также свойство stack, которое содержит строку с информацией о последовательности вызовов, которая привела к ошибке.

Оператор throw

Оператор throw генерирует ошибку. Синтаксис: throw <объект ошибки>.

Технически, в качестве объекта ошибки можно передать что угодно, это может быть даже не объект, а число или строка, но всё же лучше, чтобы это был объект, желательно — совместимый со стандартным, то есть чтобы у него были как минимум свойства name и message.

В качестве конструктора ошибок можно использовать встроенный конструктор: **new Error(message)** или любой другой.

В JavaScript встроен ряд конструкторов для стандартных ошибок: **SyntaxError, ReferenceError, RangeError** и некоторые другие.

Проброс исключения

Ошибку, о которой catch не знает, он не должен обрабатывать - в catch(e) мы анализируем объект ошибки, и если он нам не подходит, то делаем throw e.

При этом ошибка «выпадает» из try..catch наружу. Далее она может быть поймана либо внешним блоком try..catch (если есть), либо «повалит» скрипт.

Finally

```
try {
   .. пробуем выполнить код ..
} catch(e) {
   .. перехватываем исключение ..
} finally {
   .. выполняем всегда ..
}
```

Секция finally не обязательна, но если она есть, то она выполняется всегда:

после блока try, если ошибок не было,

после catch, если они были.

Finally

```
try {
   .. пробуем выполнить код ..
} catch(e) {
   .. перехватываем исключение ..
} finally {
   .. выполняем всегда ..
}
```

Секция finally не обязательна, но если она есть, то она выполняется всегда:

после блока try, если ошибок не было,

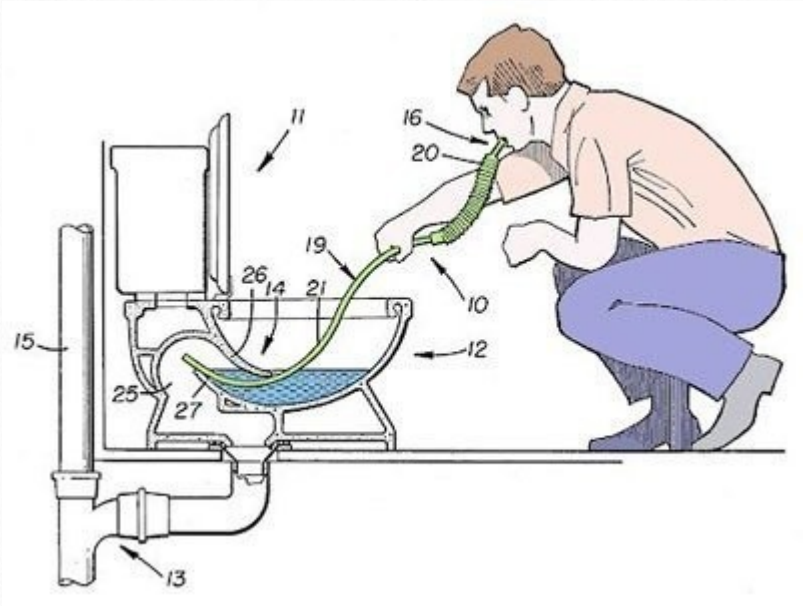
после catch, если они были.

Покры́тие ко́да

Покры́тие ко́да — мера, используемая при тестировании программного обеспечения. Она показывает процент, насколько исходный код программы был протестирован.

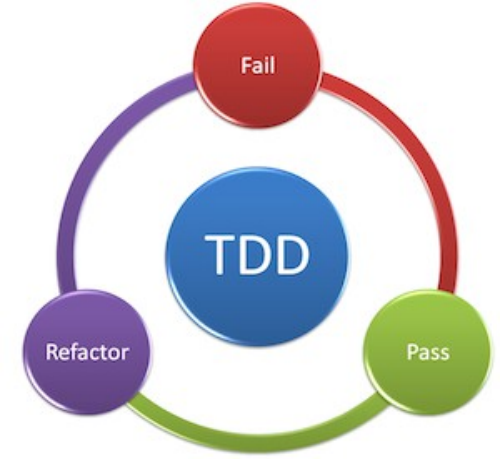
Техника покрытия кода была одной из первых методик, изобретённых для систематического тестирования программного обеспечения. Первое упоминание покрытия кода в публикациях появилось в 1963 году[1].

Покры́тие ко́да

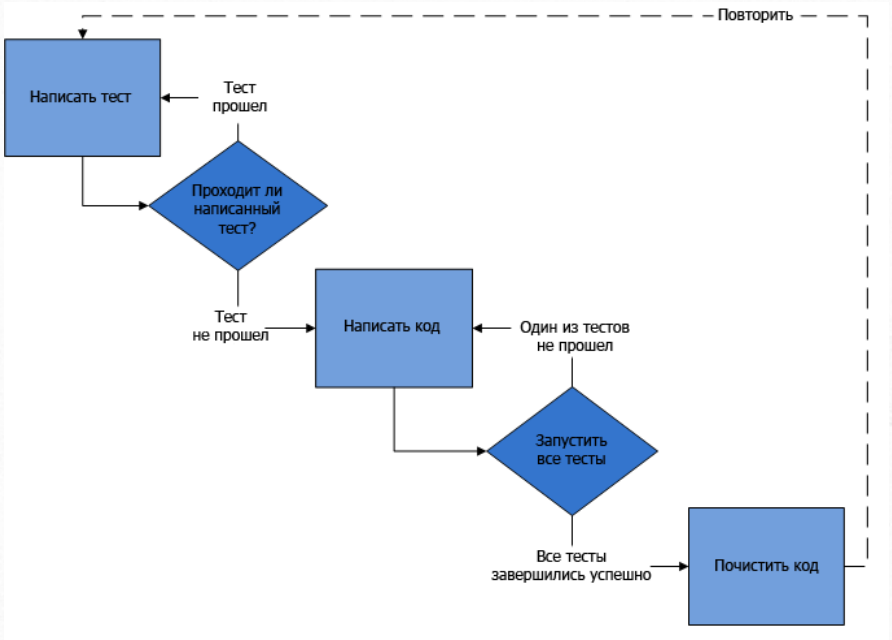


TDD

техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам. Кент Бек, считающийся изобретателем этой техники, утверждал в 2003 году, что разработка через тестирование поощряет простой дизайн и внушает уверенность (англ. inspires confidence)[1].

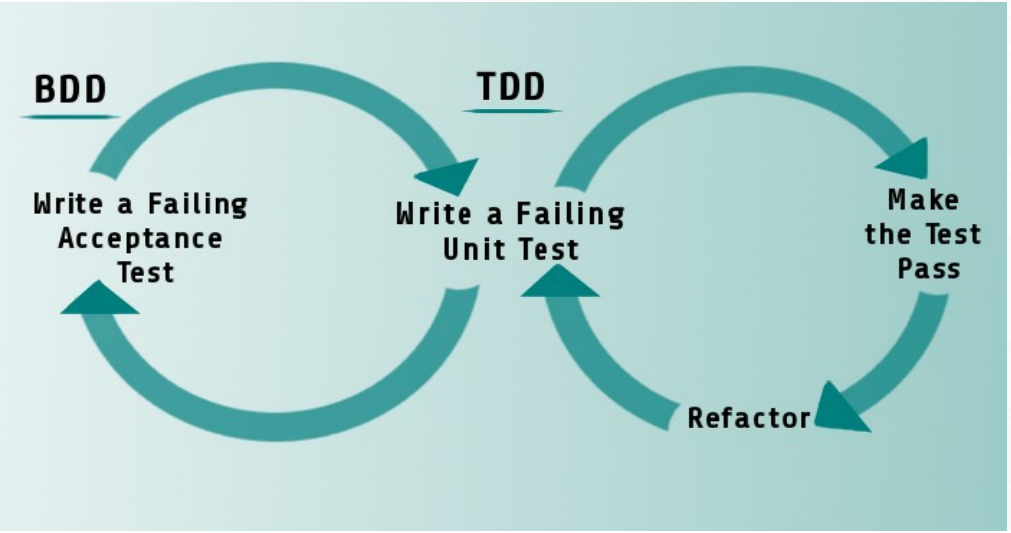


TDD


Повторить
Написать тест
Тест прошел
Проходит ли написанный тест?
Тест не прошел
Написать код
Один из тестов не прошел
Запустить все тесты
Все тесты завершились успешно
Почистить код

BDD

техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам. Кент Бек, считающийся изобретателем этой техники, утверждал в 2003 году, что разработка через тестирование поощряет простой дизайн и внушает уверенность (англ. inspires confidence)[1].



BDD and TDD

Feature: get cash from an ATM

Background:
Given the ATM has 1000
And the user John is authenticated
And the user's account has 5000
Scenario: success

When the user asks the ATM for 500
Then the ATM will have 500
And the user's account will have 4500
And the ATM will provide 500 in cash
Scenario: not enough money in the ATM
When the user asks the ATM for 1500
Then the ATM will have 1000
And the user's account will have 5000
And the ATM will notify the user it does not have enough cash