ОНЛАЙН-ОБРАЗОВАНИЕ

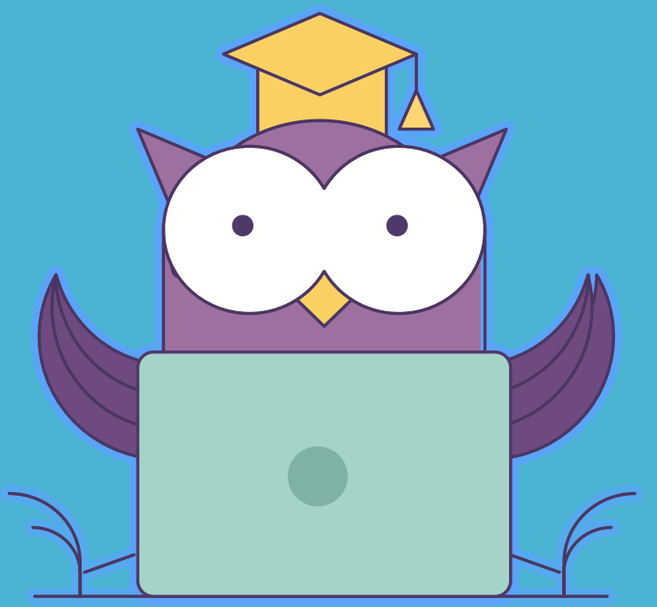
OTUS

Modern JavaScript Frameworks

Возможности JavaScript

Александр Коржиков

Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

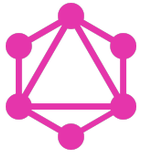
– если есть проблемы со звуком или с видео

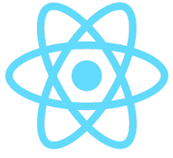

JS


node
JS


npm


express






TS

Темы предыдущего занятия

- Типы данных
- Переменные
- Функции
- Замыкания

Цели занятия

- Вспомнить и применять основные техники языка **JavaScript**
- Попрактиковаться с технологиями **Promise** и **AJAX**

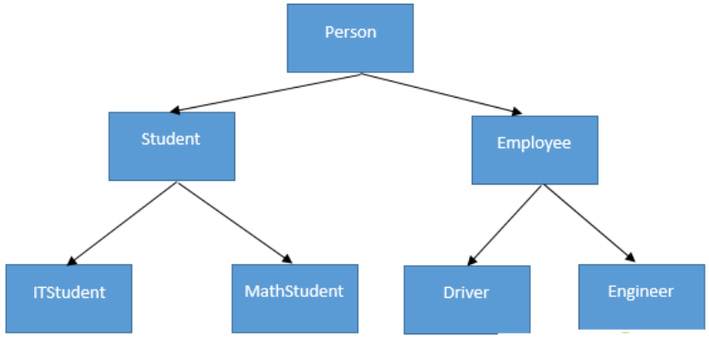
- Наследование
- Promise
- Ajax
- Обзор ES6 features

ECMAScript 6 (2015)

- http://es6-features.org/ - кратко
- http://speakingjs.com/es5/ - Speaking JavaScript
- https://learn.javascript.ru/ajax
- https://learn.javascript.ru/promise

Вопрос

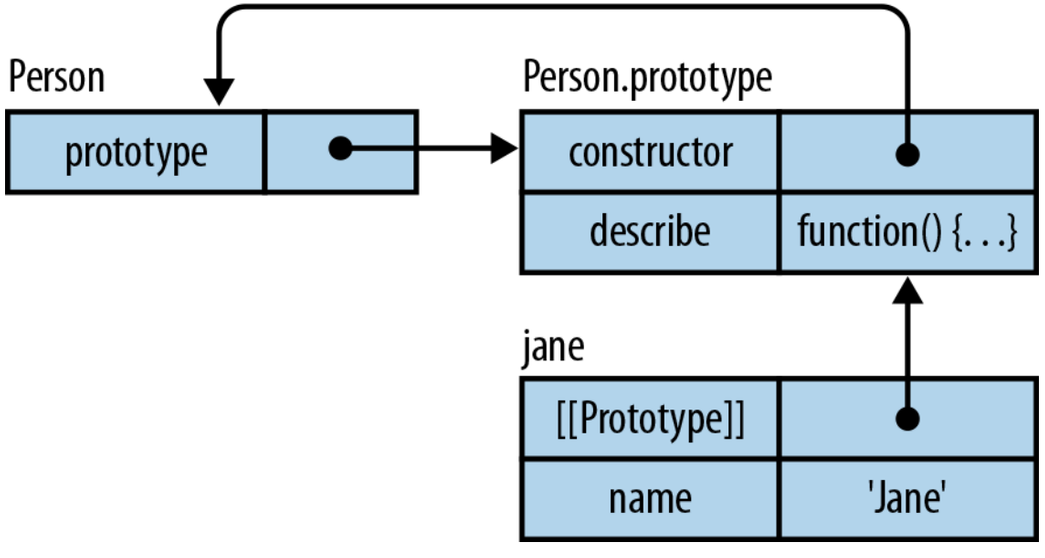
- Какие задачи решает наследование?
  - inheritance, polymorphism, encapsulation, abstraction
- Как реализовать наследование в **JavaScript**?



__proto__ ссылка на объект прототип

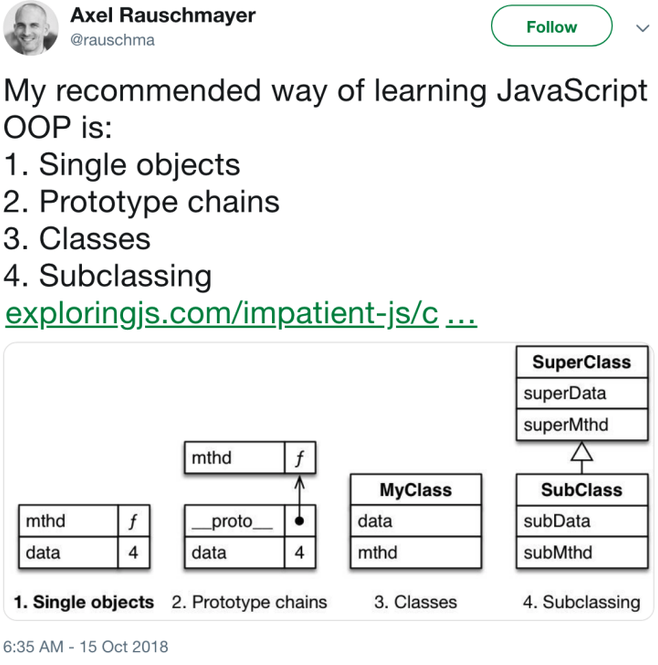
```
var a = { b: 0 }
// not recommended
a.__proto__ = { c : 1 }

// alternative
Object.getPrototypeOf(a) // Reflect.getPrototypeOf()
Object.setPrototypeOf(a, { c : 2 }) // Reflect.setPrototypeOf()
```



Object.create(proto, [propertiesObject]) устанавливает аргумент значением **__proto__**

```
var b = Object.create({ a: 1 })
console.log(b.a === 1) // true
delete b.a
console.log(b.a === 1) // true
console.log(b.__proto__) // { a: 1 }
```



new Fn() устанавливает **__proto__** равным **prototype**

```
function Fn() {}
Fn.prototype = { c: 1 }
var a = new Fn()
a.__proto__ // { c: 1 }
```

- Что будет, если вернуть из конструктора объект?

Реализовать функцию **create** - **polyfill** для **Object.create**

```
var create = function() { /* ... */ }

var b = create({ a: 1 })
b.a === 1 // true
delete b.a
b.a === 1 // ?
```

class - «синтаксический сахар» для задания конструктора и прототипа

```
class Pie {
  constructor(name) {
    this.name = name
  }
}

class Pizza extends Pie {
  bake() {
    super.bake()
  }
}
```

- **class C** - декларация класса
- **[extends P]** - наследование
- **constructor(), methods()** - конструктор и методы класса
- **new C()** - создание инстансов класса
- **get / set** - функции доступа / присваивания
- **static** - статичные свойства и методы
- **super** - обращение к родительскому классу

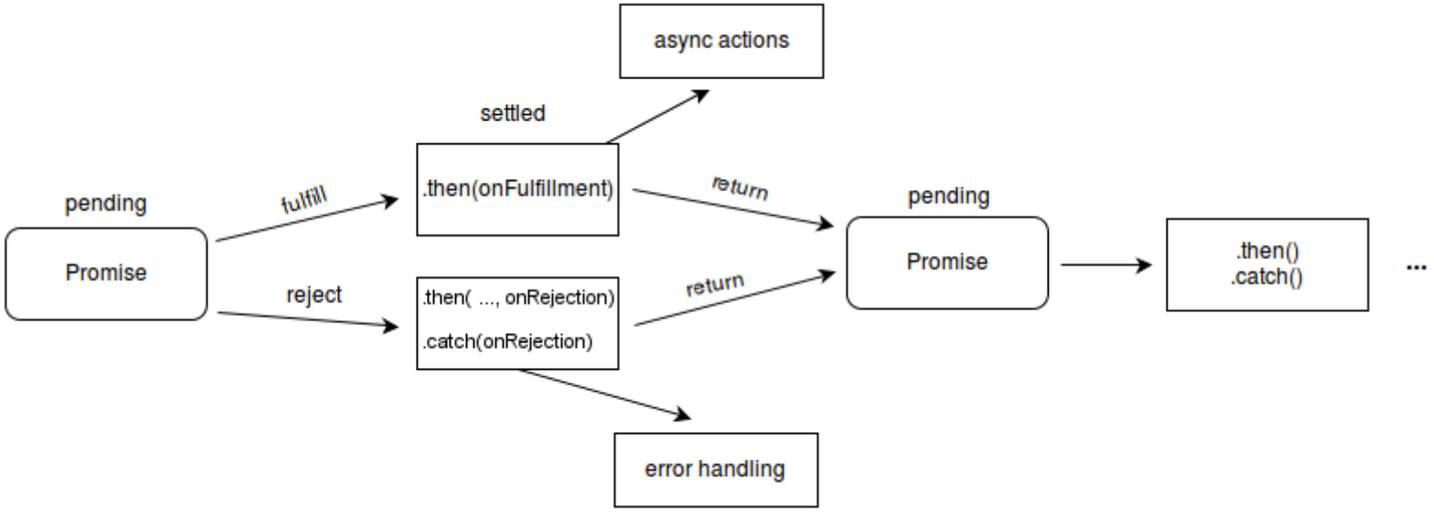
```
class Pie {
  constructor(name) {
    this.name = name
  }
}

class Pizza extends Pie {
  bake() {
    super.bake()
  }
}
```

Inheritance

Q&A

Техника написания асинхронного кода



- **resolve()** - успешное завершение
- **reject()** - выполнено с ошибкой

```
new Promise((resolve, reject) =>
  setTimeout(() => resolve('Hello'), 500)
)

// Alternative
Promise.resolve(value)
Promise.reject(reason)
```

- **then()** - вызвана при **resolve()** с результатом
- **catch()** - при **reject**

```
Promise.resolve(1) // reject()
  .then((a) => { console.log(a) throw 'Oh no!' })
  .catch(reason => { console.error(reason) })

// Alternative
// then(onThen, onCatch)
```

- Chaining...
  - **reject()** следует до **catch()**
  - **reject()** может быть неявно спровоцировано исключением

Что будет выведено в консоль?

```
Promise.resolve(1)
  .then(console.log)
  .then(console.log)
```

А здесь?

```
Promise.reject()
   .catch(() => 2)
   .then(() => { throw 3 })
   .catch(() => 4)
   .then((a) => console.log(a))
```

- **finally()** - выполнится при любом исходе
- **all()** - параллельные promises

```
Promise.resolve()
  .finally(() => 'done')// parallel

Promise.all([promise, promise2])
  .then(() => 'then')
```

- **race()** - как только один из promise-ов завершится

Написать **polyfill** для **Promise.all()**,ожидающий исполнение всех **promise** аргументов и возвращающий все результаты

```
promiseAll([
 promise,
 promise2
])
.then(([res1, res2]) => 'done')
```

«синтаксический сахар» для работы с **Promise**

- **await** внутри **async**
- **return** возвращается **Promise**

```
const promiseFn = () =>
  new Promise(resolve =>
    setTimeout(() => resolve(3.14), 500)
  )

async function asyncFn() {
  const Pi = await promiseFn()
  console.log(Pi)
}

asyncFn()
```

Переписать, используя **async / await**

```
function getResponseSize(url) {
  return fetch(url).then(response => {
    const reader = response.body.getReader()
    let total = 0

    return reader.read().then(function processResult(result) {
      if (result.done) return total

      const value = result.value
      total += value.length
      console.log('Received chunk', value)

      return reader.read().then(processResult)
    })
  })
}
```

- В чем различие между разными подходами?

Что здесь не так?

```
new Promise((resolve, reject) => {
 const result = doSomething()
 if (result) {
  resolve(result)
 }

 doSomethingElse()
 reject(new Error('Empty result'))
})
```

Promise

Q&A

Asynchronous JavaScript And Xml

Отправка запроса и получение ответа на сервер без перезагрузки страницы

Вопрос

Какие существуют способы отправить запрос на сервер?

- **XMLHttpRequest()**
- fetch()
- WebSocket()
- JSONP

```
var req = new XMLHttpRequest()
req.open('GET', '/xhr/test.html', true)
req.onreadystatechange = function () {
  if (req.readyState == 4) {
    if (req.status == 200) {
      alert(req.responseText)
    }
  }
}
req.send(null)
```

- XMLHttpRequest()
- **fetch()**
- WebSocket()
- JSONP

```
fetch('https://api.fixer.io/latest')
  .then((resp) => resp.json())
  .then(({
    rates: {
      RUB
    }
  }) => {
    console.log(`1 EUR = ${RUB} RUB`)
  })
```

- XMLHttpRequest()
- **fetch()**
- WebSocket()
- JSONP

```
fetch(url)
  .then((response) => { /* ... */ })
  .then(/* ... */)

/*
 * When did you join github?
 * https://api.github.com/users/...
 */
```

- Template Literals
- Parameters Handling - defaults, rest, spread
- Modules
- Symbol - iterators, generators
- Data Structures - Set, Map, Typed Arrays
- Proxy & Reflect
- ...

Если остается время

Реализовать функцию **reduce** - **polyfill** для **Array.prototype.reduce**

```
// example
[[0, 1], [2, 3], [4, 5]].reduce((memo, currentValue) => {
  return memo.concat(currentValue)
}, [])

reduce(
 [1, 2, 3, 4], // arguments
 (a, b) => a + b, // action
 0 // initial value
) // 10
```

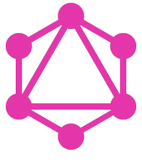
- Разобрали теорию и задачи, возникающие при работе с технологиями AJAX, Promise
- Вспомнили что такое наследование и его реализацию в JavaScript

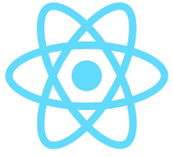

JS


node
JS


npm


express






A


TS

Написать функцию promiseReduce, которая получает на вход **asyncFunctions**, **reduce**, **initialValue**. promiseReduce поочередно вызывает переданные асинхронные функции и выполняет reduce функцию сразу при получении результата до вызова следующей асинхронной функции.

```
function promiseReduce(asyncFunctions, reduce, initialValue) {/* ? */}

const fn1 = () => Promise.resolve(1)
const fn2 = () => new Promise(resolve => {
 setTimeout(() => resolve(2), 1000)
})

promiseReduce([fn1, fn2], (memo, value) => memo * value, 2)
/* => 4 */
```

Спасибо за внимание!

Пожалуйста, пройдите опрос в личном кабинете

- Все ли темы были понятны? (да - нет)
- Легкий материал или нет? (1 просто - 10 сложно)