

Разработка Мобильных Приложений с React Native

Лекции

- Обзор Javascript
- JavaScript, ES6
- React, JSX
- Компоненты, Свойства (Props), Состояние, Стил
- Компоненты, Views, Данные от пользователя (User input)
- Дебаггиг
- Данные
- Навигация
- Экспо компоненты
- Redux
- Производительность
- Публикация, Тестирование

Проекты

- Проект 0
- Проект 1
- Проект 2
- Финальный Проект

Mobile App Development with React Native

Jordan Hayashi

Информация

- Телеграмм
- Email
- Гугл

Лекции

- Короткие перерывы

Лекции

- Короткие перерывы
- Есть вопрос? Перебивайте меня

Лекции

- Короткие перерывы
- Есть вопрос? Перебивайте меня
 - Концепции строятся друг на друге, поэтому важно понимать абсолютно все.

Лекции

- Короткие перерывы
- Есть вопрос? Перебивайте меня
 - Концепции строятся друг на друге, поэтому важно понимать абсолютно все.
 - Если какая-то информация будет лишней, я дам вам знать

Лекция 0: Обзор JavaScript

Jordan Hayashi

JavaScript - интерпретируемый

- У каждого браузера свой JavaScript-движок, который либо интерпр. код, либо использует своего рода ленивую компиляцию

JavaScript - интерпретируемый

- У каждого браузера свой JavaScript-движок, который либо интерпр. код, либо использует своего рода ленивую компиляцию
 - V8: Chrome и Node.js
 - SpiderMonkey: Firefox
 - JavaScriptCore: Safari
 - Chakra: Microsoft Edge/IE

JavaScript - интерпретируемый

- У каждого браузера свой JavaScript-движок, который либо интерпр. код, либо использует своего рода ленивую компиляцию
 - V8: Chrome и Node.js
 - SpiderMonkey: Firefox
 - JavaScriptCore: Safari
 - Chakra: Microsoft Edge/IE
- Каждый из них имплементирует стандарт ECMAScript

Синтаксис

Типы данных

Типы данных

- Динамическая типизация

Типы данных

- Динамическая типизация
- Примитивные типы (нет методов, неизменяемы)
 - undefined
 - null
 - boolean
 - number
 - string
 - (symbol)

Типы данных

- Динамическая типизация **immutable**
- Примитивные типы (нет методов, неизменяемы)
 - undefined
 - null
 - boolean
 - number
 - string
 - (symbol)

Типы данных

- Динамическая типизация **immutable**
- Примитивные типы (нет методов, неизменяемы)
 - undefined
 - null
 - boolean
 - number
 - string
 - (symbol)
- Объекты

Typecasting?

Coercion

Coercion (принуждение)

Coercion (принуждение)

- Явное и неявное принуждение

- `const x = 42;`
- `const explicit = String(x);` `// explicit === "42"`
- `const implicit = x + "";` `// implicit === "42"`

Coercion (принуждение)

- Явное и неявное принуждение

- `const x = 42;`
- `const explicit = String(x);` `// explicit === "42"`
- `const implicit = x + "";` `// implicit === "42"`

- `==` VS. `===`

- `==` принуждает типы
- `===` обязует использовать одинаковые типы

coercion

Truthy vs Falsy

Truthy vs Falsy

(Правдивые и Фальши)

Truthy vs Falsy

Truthy vs Falsy

- Фолзи значения

Truthy vs Falsy

- **Фолзи значения**

- `undefined`
- `null`
- `false`
- `+0`, `-0`, `NaN`
- `""`

Truthy vs Falsy

- **Фолзи значения**

- `undefined`
- `null`
- `false`
- `+0`, `-0`, `NaN`
- `""`

- **Труфи значения**

Truthy vs Falsy

- **Фолзи значения**

- undefined
- null
- false
- +0, -0, NaN
- ""

- **Труфи значения**

- {}

Truthy vs Falsy

- **Фолзи значения**

- undefined
- null
- false
- +0, -0, NaN
- ""

- **Труфи значения**

- {}
- []

Truthy vs Falsy

- **Фолзи значения**

- undefined
- null
- false
- +0, -0, NaN
- ""

- **Труфи значения**

- {}
- []
- **Все остальные**

Objects, Arrays, Functions, Objects

Objects, Arrays, Functions, Objects

- Сколько объектов я указал?

Objects, Arrays, Functions, Objects

- Сколько объектов я указал?
- На самом деле 4.

Objects, Arrays, Functions, Objects

- Сколько объектов я указал?
- На самом деле 4.

- Остальные объекты

Objects, Arrays, Functions, Objects

- Сколько объектов я указал?
- На самом деле 4.
- Остальные объекты
- Прототипное наследование (продолжение следует...)

Primitives vs. Objects

Primitives vs. Objects

- Все примитивы неизменяемы (immutable)

Primitives vs. Objects

- Все примитивы неизменяемы (immutable)
- Объекты изменяемы (mutable) и хранятся только по ссылке

Primitives vs. Objects

- Все примитивы неизменяемы (immutable)
- Объекты изменяемы (mutable) и хранятся только по ссылке
- Копирование по ссылке vs. копирование по значению

Prototypal Inheritance (прототипное наследование)

Prototypal Inheritance (прототипное наследование)

- У непримитивных типов есть кое-какие свойства/методы (у них связь)

Prototypal Inheritance (прототипное наследование)

- У непримитивных типов есть кое-какие свойства/методы (у них связь)
 - `Array.prototype.push()`
 - `String.prototype.toUpperCase()`

Prototypal Inheritance (прототипное наследование)

- У непримитивных типов есть кое-какие свойства/методы (у них связь)
 - `Array.prototype.push()`
 - `String.prototype.toUpperCase()`
- Каждый объект хранит ссылку своего прототипа

Prototypal Inheritance

(прототипное наследование)

- У непримитивных типов есть кое-какие свойства/методы (у них связь)
 - `Array.prototype.push()`
 - `String.prototype.toUpperCase()`
- Каждый объект хранит ссылку своего прототипа
- Чем ближе свойство/метод, тем больше у него приоритет.

Prototypal Inheritance

(прототипное наследование)

- Почти у всех примитивных типов есть свои объект-обертки (object wrappers)
 - String()
 - Number()
 - Boolean()
 - Object()
 - (Symbol())

Prototypal Inheritance

(прототипное наследование)

- JS автоматически «упаковывает» (wrap) примитивные значения, таким образом давая вам доступ к методам

```
42.toString()           // Errors
const x = 42;
x.toString()            // "42"
x.__proto__             // [Number: 0]
x instanceof Number     // false
```

Prototypal Inheritance (прототипное наследование)

- В чем польза ссылок на прототип?

Prototypal Inheritance (прототипное наследование)

- В чем польза ссылок на прототип?
- Какая есть альтернатива?

Prototypal Inheritance (прототипное наследование)

- В чем польза ссылок на прототип?
- Какая есть альтернатива?
- В чем опасность такого метода?

Score (Область видимости)

- **Время существования переменной**
 - Block scope / Блочная область видимости (const, let): с места создания, до следующего символа «}»

Scope (Область видимости)

- **Время существования переменной**
 - Block scope / Блочная область видимости (const, let): с места создания, до следующего символа «}»
 - Lexical scope / Лексическая область видимости (var): с момента создания, до конца их функции

Scope (Область видимости)

- **Время существования переменной**
 - Block scope / Блочная область видимости (const, let): с места создания, до следующего символа «}»
 - Lexical scope / Лексическая область видимости (var): с момента создания, до конца их функции
- **Поднятие (Hoisting)**
 - Поднимаются объявления функций и переменных, но не лексически ограниченные инициализации

Scope (Область видимости)

- **Время существования переменной**
 - Block scope / Блочная область видимости (const, let): с места создания, до следующего символа «}»
 - Lexical scope / Лексическая область видимости (var): с момента создания, до конца их функции
- **Поднятие (Hoisting)**
 - Поднимаются объявления функций и переменных, но не лексически ограниченные инициализации
- **Но как/почему?**

Движок JavaScript'a

- До выполнения кода, движок прочитает весь файл и выдаст ошибку, в случае нахождения синтаксической ошибки
 - Все объявления функций будут сохранены в памяти
 - Пропуск инициализации переменных, но лексически-видимые имена переменных будут объявлены

Глобальный объект

- На самом деле все переменные и функции являются свойствами и методами глобального объекта
 - window - глобальный объект браузера
 - global - глобальный объект Node.js