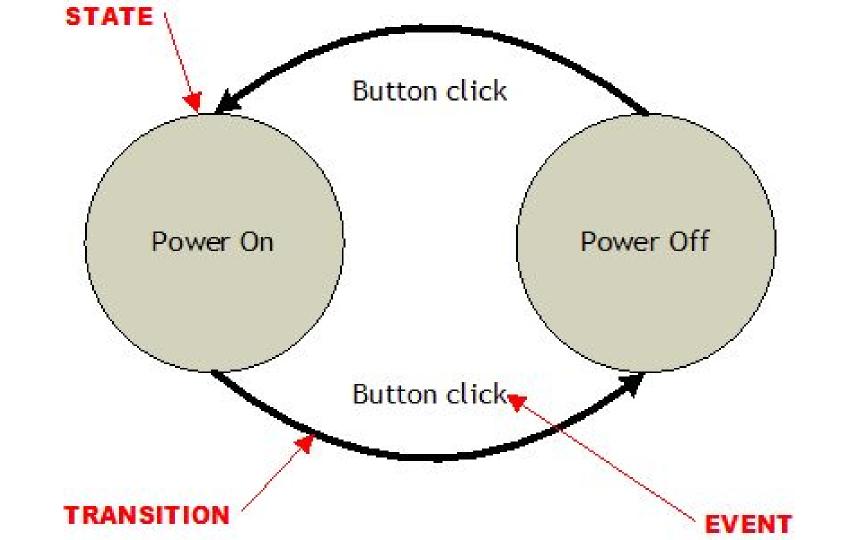
UI

State Machine

О состояниях и переходах между ними



Реальная жизнь

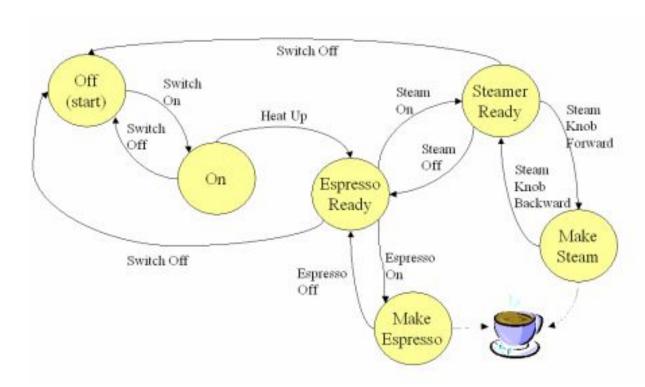
Вединговые аппараты

Бытовая электроника

Машины

Человек

Все что может меняться со временем

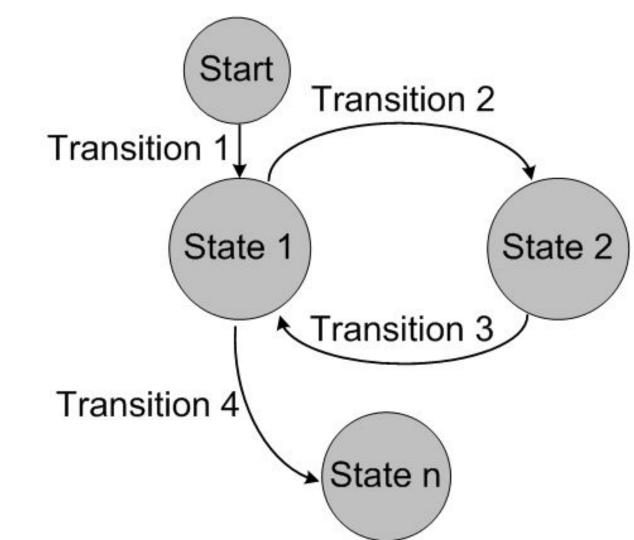


Программирование

Все что может меняться со временем

Состояние счета (оплачен/просрочен)

Страница (опубликована/скрыта)



При чем тут UI?



News Headline

something

visual -

apcoming



Remain factor volpulate turpts. Quidque arca ante, carsos in, orisino quis, exercis al, justo filaves el pede.

Poesere allacus vites turpis consequet semper. Mauris sed nulla qua nis

Interdum tempor. Pollentocque condimentam folio a sem. Sod a lorem ut est Broddant consectation. Cres ac esim vel dial vestilisation socioliti.

Disensational settlement community recities as a feliamona strain pode music, verbouwer ver, retrum et, inconsert vu, eren, notembre tempor. Proin diam segue, semper visez, verius et, vivens et, fisite, Anlamicague carnello delor vel filore; Donoc tempos quem-quis neque. Donoc all amet seine. Sed que etit. Phasellus sactor enem eget nem. Denec ut pures. Visiensia posisieris, ante eu tempor dictams, felt nebb hacifes sem, es auctor moter nella non besen. Sesponalisse potenti. Prasport allejand, nepper protium-compie matris, gisiem sagar disposition setti, qui comi segor disposition setti, qui presium mel fectus as rinagria, britagin accumisas.

Sed a losem un est timodunt consecutavo: Quisque arou area, cursus in, armane que, vivera ut, parto Mallam inmenanto generále arci.

consectatues adipacing oils 's

Note factor

tincidiant consectations Walterfacilisis

Propert oliquet, reque

groum surgue dignissim anta, ac pretium nici lectus

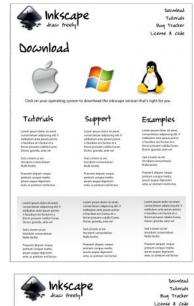
Promient aliquet, requi

Lower speam dolar or amer, umaschstare adpiscing elst it estiluture ame course prins in fraudkus encilikate et u fraum porums cubilla Curae. Donec greeds, ante vel

Sed a lovem ut exi

Promot alignet requ

pretium conque matte, ignam augue digmostre ante, as pretium red last as











pretion conque matrix.



Processed allegant, evenue

prefium compare marks, grown eagur dignocem ante, as perfium not lectur

Promote aliquet, nego

pertium conque martis, spoure augue dignissim ante, ac pretium niel lectus

Plantent elliquet, nequ

profium conque metto, ipeum supur dignissim unde, ex predium noti lactus



Donec opyrids, ando yell

Praesent aliquet, neque

pedian coopernate, years augue digresses ante, ai pertian not lecha

Nulls facilità.

Promient aliquet, reque

antis, ac protium not lectur

Sed a lovers ut-est

Nufls facilist.

tincidant consecreture Nafia facilisi.

$29 = 2\Pi$

э - экран, п - переход

59 = ?

Вспоминаем комбинаторику;)

20 = 5*(5-1)

Для 10 экранов это 90

```
// Switch this view into `"editing"` mode, displaying the input field.
edit: function() {
    $(this.el).addClass("editing");
    this.input.focus();
},

// Close the `"editing"` mode, saving changes to the todo.
close: function() {
```

Backbone View - Ручное управление DOM

\$(this.el).removeClass("editing");

},

this.model.save({content: this.input.val()});

А что с обычными сайтами? (клиент-

серверная модель)

Х Экранов = Х Переходов

В реальности переходы остались, но на уровне данных

А если попробовать реализовать на

клиенте?

Принцип работы: полная перерисовка всегда



Встречайте Act

https://github.com/mokevnin/act

В этом месте живой пример для тех

кто слушает доклад в живую

Facts

- it works!
- ~ 60 sloc
- innerHTML
- State is Structure
- Render is Pure
- One-Way Data Flow

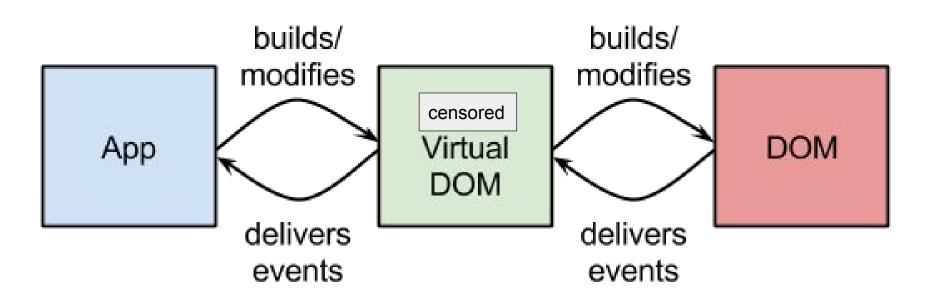
```
class ToDo extends Act.Component {
 getInitialState() {
    return {currentText: "", tasks: this.ops.tasks || []};
  handleAddTask(e) {
    e.preventDefault();
    const tasks = this.state.tasks.slice();
    tasks.unshift(this.state.currentText);
    this.setState({currentText: "", tasks: tasks});
  render() {
    const locals = {
      tasks: this.state.tasks,
      currentText: this.state.currentText
   };
    return templateFn(locals);
```

В чем подвох?

- innerHTML медленно
- HTML не расширяемо. Приложение один компонент.
- HTML много проблем события/фокусы.

Что делать?

Virtual Dom



Virtual Dom

- Diff алгоритм может быть сведен к O(n) вместо O(n^3)
- JS адски быстрый, генерировать {} каждый раз легко
- В реальном DOM делаются только минимально необходимые изменения
- Никто не может делать изменения напрямую в реальный DOM, контролируемый виртуальным

На практике метод render должен

возвращать структуру {} в терминах VDOM

Рабочий вариант, но глубина убивает понимание

```
function render(data) {
    return new VNode('div', {
        className: "greeting"
        new VText("Hello " + String(data.name))
    ]);
```

A если эмулировать html?

Неплохо. Но это ведь јѕ код...

```
class Toolbar extends Component<{}, {}, {}> {
  render() {
    return (
      <div className="toolbar">
        <div className="btn-group" role="group">
          <div className={this.getStatusClasses()}>
            <span className={this.getStatusInnerClasses()} />
          </div>
        </div>
      </div>
```

Пока не придумали ничего лучше, но на практике это работает нормально

Что получили?

- Своя система событий (исправляющая проблемы браузерной)
- Имена аттрибутов соответствуют спецификациям dom
- Возможность создавать новые компоненты. Композируемость
- Серверная генерация содержимого
- А если рендерить не в html? И это тоже



Hello, World!

```
var HelloMessage = React.createClass({
    render: function() {
        return <div>Hello {this.props.name}</div>;
    }
});

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

Reactjs

- Just View
- One-Way Data Flow
- Virtual Dom / Independent rendering
- Flux/React Native/Babel/Immutable.js

Reactjs

- Линейный рост сложности. Практически отсутсвует случайная сложность.
- Предсказуемое поведение. Простая отладка.
- Легко сопровождать. Есть один понятный и простой способ создавать архитектуру приложения. Сложно писать по другому.
- Раньше были проблемы с анимациями, сейчас не знаю. https://github.com/chenglou/react-motion

Спасибо! twitter.com/mokevnin