

Добавить архитектуры AsyncDom

AsyncDom

PiterJS #21



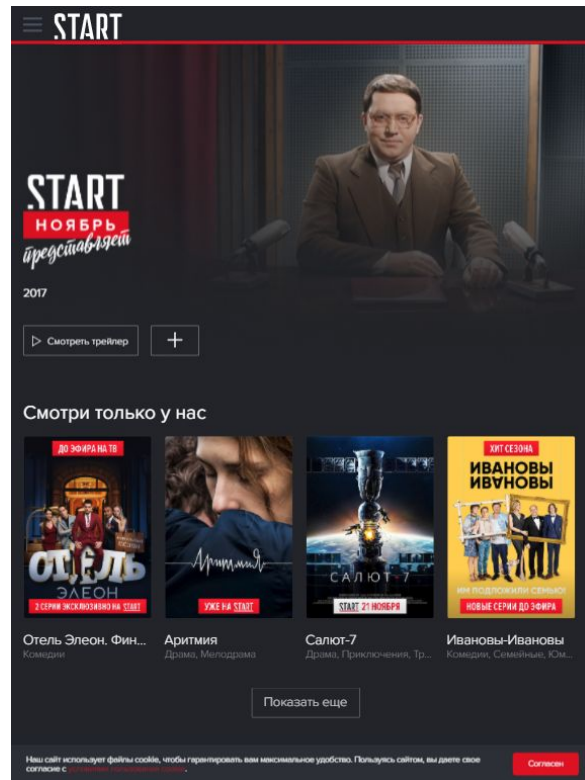
Async Dom

start.ru

- онлайн сервис видеостриминга

Альтернативы?

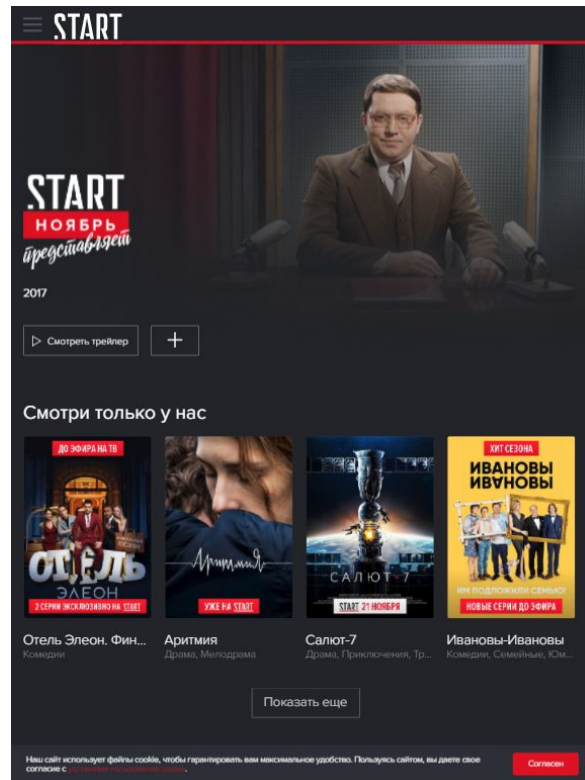
- okko
- 1.5 года на запуск
- 200+ команда



Async Dom

Проблема - нужен перформанс

Подготовка к запуску start.ru



Async Dom

Подготовка к запуску start.ru

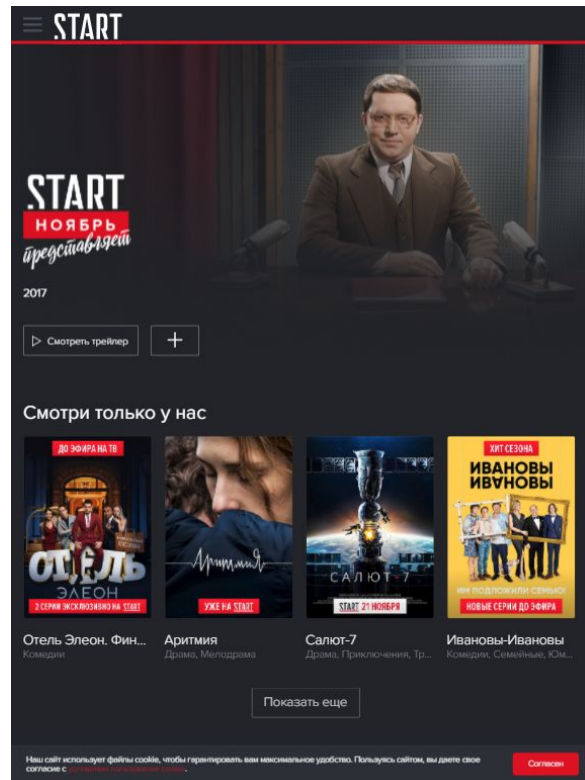
- релиз за 6 месяцев

Команда

- 10 человек

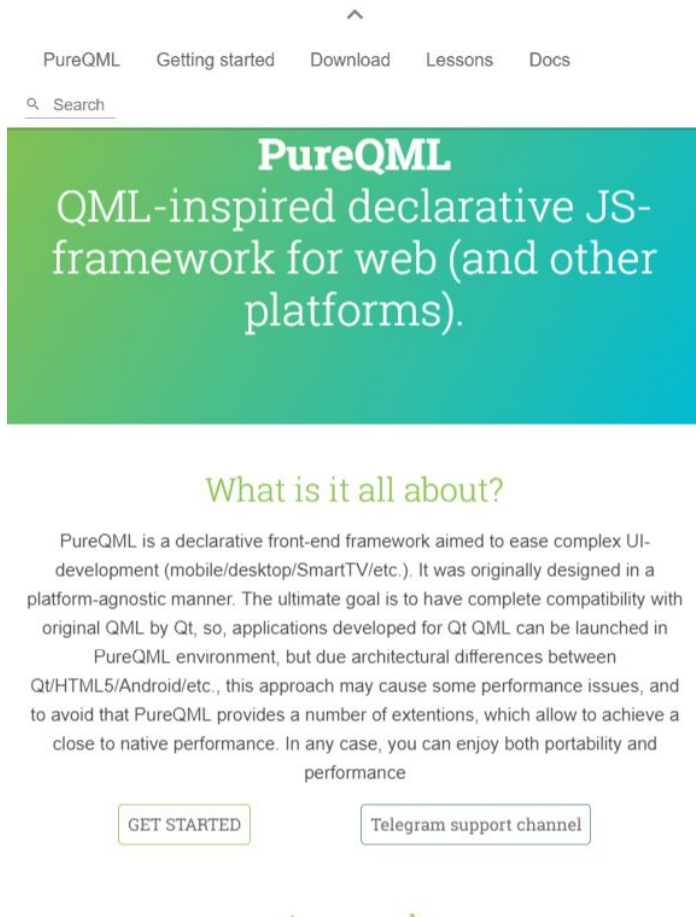
Платформы:

- web
- smartTV



Async Dom

Фреймворк



Async Dom

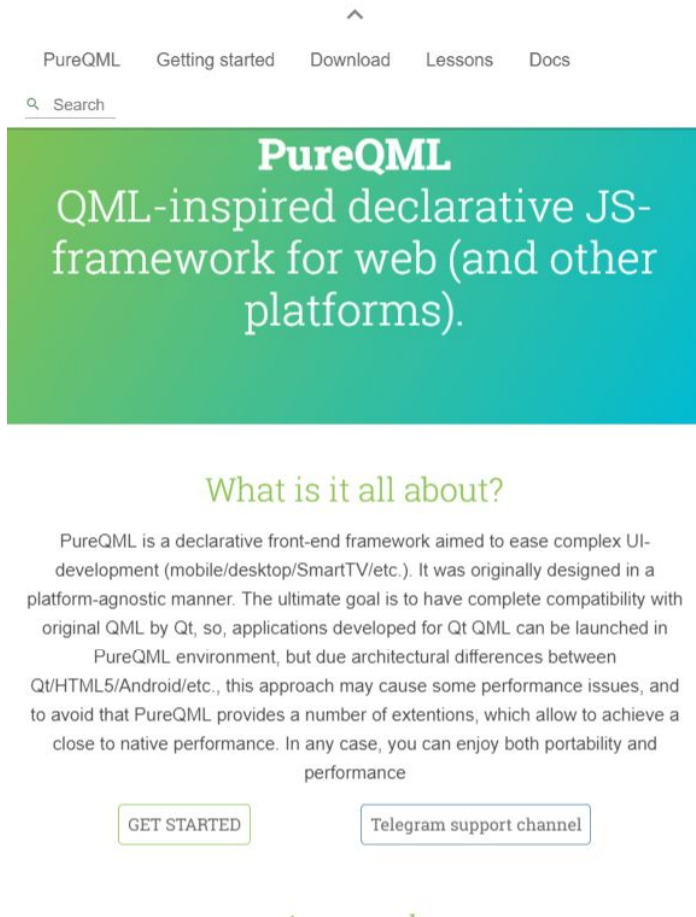
создатели:

Бывшие C++ разработчики UI

для приёмников TriColorTV

looks like:

- QML & JS



Async Dom

Плюсы PureQml

- Заточен под работу на слабых устройствах
- Кастомный run-loop
- Не завязан на DOM
- Декларативен
- Небольшой размер бандла (~120кб)
- Вызывает Layout только после всех модификаций бизнес-логики
- Автофокус

Async Dom

Минусы

- Синтаксис - смесь QML и JS
- ES 5

```
Row {
    width: 300;
    height: 150;
    spacing: 10;

    WebItem {
        id: leftRect;
        width: 100;
        height: 100;
        anchors.verticalCenter: parent.verticalCenter;
        color: "green";
        focus: true;
        border.width: activeFocus ? 4 : 0;
        border.color: "red";

        Text {
            anchors.centerIn: parent;
            color: "white";
            text: "Click me";
            visible: !parent.activeFocus;
        }

        onClicked: { this.setFocus() }
    }

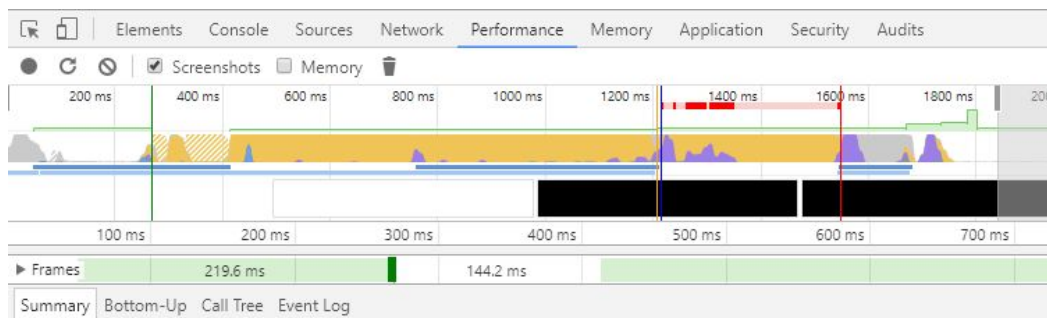
    WebItem {
        id: rightRect;
        width: 100;
        height: 100;
        anchors.verticalCenter: parent.verticalCenter;
        focus: true;
        color: "green";
        border.width: activeFocus ? 4 : 0;
        border.color: "red";

        Text {
            anchors.centerIn: parent;
            color: "white";
            text: "Click me";
            visible: !parent.activeFocus;
        }

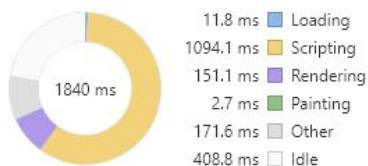
        onClicked: { this.setFocus() }
    }
}
```

Async Dom

Дебажим



Range: 0 – 1.84 s



1.1 ms	0.1 %	775.4 ms	75.5 %	▶ Evaluate Script
46.7 ms	4.6 %	733.3 ms	71.4 %	▶ (anonymous)
1.7 ms	0.2 %	615.5 ms	60.0 %	▶ ContextPrototype.start
21.3 ms	2.1 %	504.8 ms	49.2 %	▶ UiAppPrototype.__setup
2.8 ms	0.3 %	366.1 ms	35.7 %	▶ safeCallImpl
35.4 ms	3.5 %	333.7 ms	32.5 %	▶ set
24.3 ms	2.4 %	308.2 ms	30.0 %	▶ ObjectPrototype.update
5.6 ms	0.5 %	157.8 ms	15.4 %	▶ AuthorizationPagePrototype.__setup
5.7 ms	0.6 %	146.4 ms	14.3 %	▶ EventEmitterPrototype.emitWithArgs
3.5 ms	0.3 %	130.8 ms	12.7 %	▶ StartTvDigitInputPrototype.__setup
8.5 ms	0.8 %	114.6 ms	11.2 %	▶ SettingsPagePrototype.__setup
5.2 ms	0.5 %	109.3 ms	10.6 %	▶ Event
5.8 ms	0.6 %	109.1 ms	10.6 %	▶ UiAppPrototype.__create
2.6 ms	0.3 %	102.4 ms	10.0 %	▶ Function Call
11.5 ms	1.1 %	97.6 ms	9.5 %	▶ ContextPrototype.run
7.4 ms	0.7 %	97.4 ms	9.5 %	▶ ProductContentPrototype.__setup
0.4 ms	0.0 %	86.1 ms	8.4 %	▶ _globals.html5.html.exports.setAnimation
2.5 ms	0.2 %	85.7 ms	8.3 %	▶ setTransition
85.3 ms	8.3 %	85.4 ms	8.3 %	▶ Layout
3.7 ms	0.4 %	84.7 ms	8.3 %	▶ AccountPagePrototype.__setup
9.4 ms	0.9 %	80.6 ms	7.9 %	▶ _globals.html5.html.ElementPrototype.forceLayout
77.8 ms	7.6 %	77.8 ms	7.6 %	▶ Compile Script
0.8 ms	0.1 %	69.3 ms	6.8 %	▶ DigitGridPrototype.__setup
4.8 ms	0.5 %	66.7 ms	6.5 %	▶ StartTvPlayerPrototype.__setup
0.1 ms	0.0 %	66.3 ms	6.5 %	▶ ContextPrototype.processActions
4.2 ms	0.4 %	58.7 ms	5.7 %	▶ ProductPagePrototype.__setup
3.4 ms	0.3 %	54.9 ms	5.3 %	▶ StartTvKeyButtonPrototype.__setup
1.2 ms	0.1 %	54.6 ms	5.3 %	▶ forEach
3.0 ms	0.3 %	53.5 ms	5.2 %	▶ SearchPagePrototype.__setup
1.3 ms	0.1 %	52.4 ms	5.1 %	▶ _globals.core.Item
5.5 ms	0.5 %	51.7 ms	5.0 %	▶ PlayerOsdPrototype.__setup
0.4 ms	0.0 %	49.4 ms	4.8 %	▶ DigitGridKeyPrototype.__setup

Async Dom

Наблюдения

- **Node.clone()** - работает быстрее чем **createElement**

Async Dom

Наблюдения

- **Node.clone()** - работает быстрее чем **createElement**
- **Document.createDocumentFragment()** - отложенная модификация DOM
 - **Fragment** не вызывает **Layout**, можно накидать много нод, поправить их атрибуты, а уже после всего этого добавлять в DOM

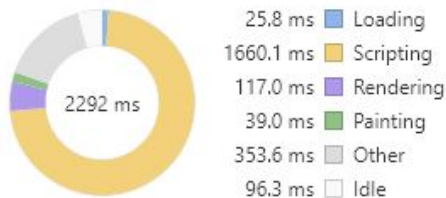
Async Dom

Наблюдения

rendering speed **+40%**

- Node.clone() - для новых элементов
- Document.createDocumentFragment() - отложенная модификация DOM

Range: 248 ms – 2.54 s



Async Dom

Что не так?

70%

Scripting занимает более 70% времени

Async Dom

Что не так?

> 16ms

JS наплевать на ваш **RequestAnimationFrame**

- 16 ms - скорость, за которую нужно закончить JS операцию, чтобы обеспечить отзывчивость интерфейса на уровне 60 fps

Async Dom

Что не так?

Если JS выполняется больше 16ms, интерфейс тормозит

Async Dom

Что не так?

Если JS выполняется больше 16ms, интерфейс тормозит

- но это все и так знают

Async Dom

Что не так?

А что может повлиять на скорость исполнения JS кода?

Async Dom

Что не так?

А что может повлиять на скорость исполнения JS кода?

- работа с DOM внутри JS,

Async Dom

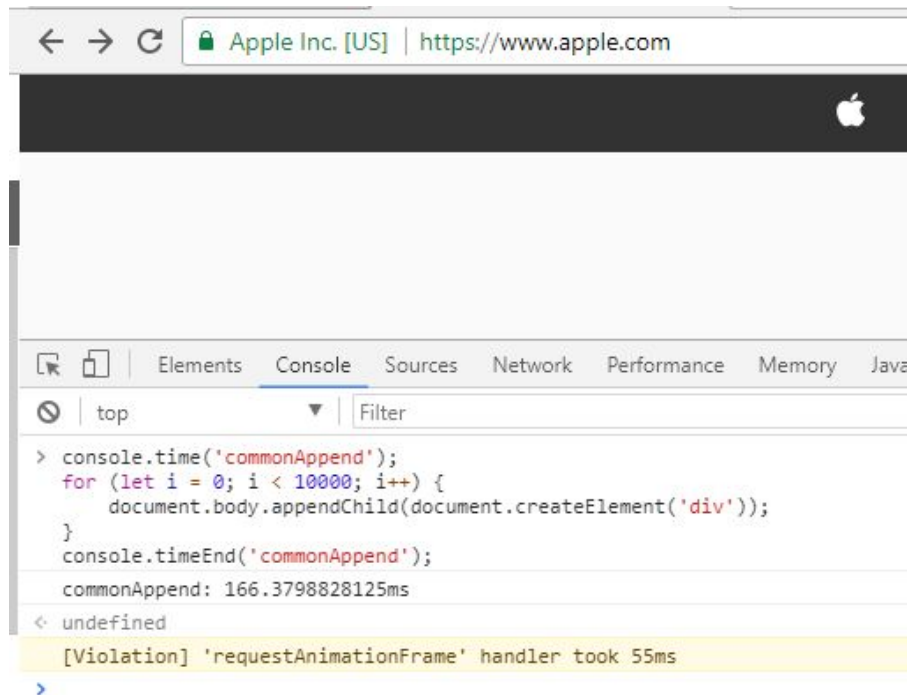
Скорость JS кода* зависит от CSS,

- если много стилей, JS начинает тормозить

* - кода, связанного с DOM

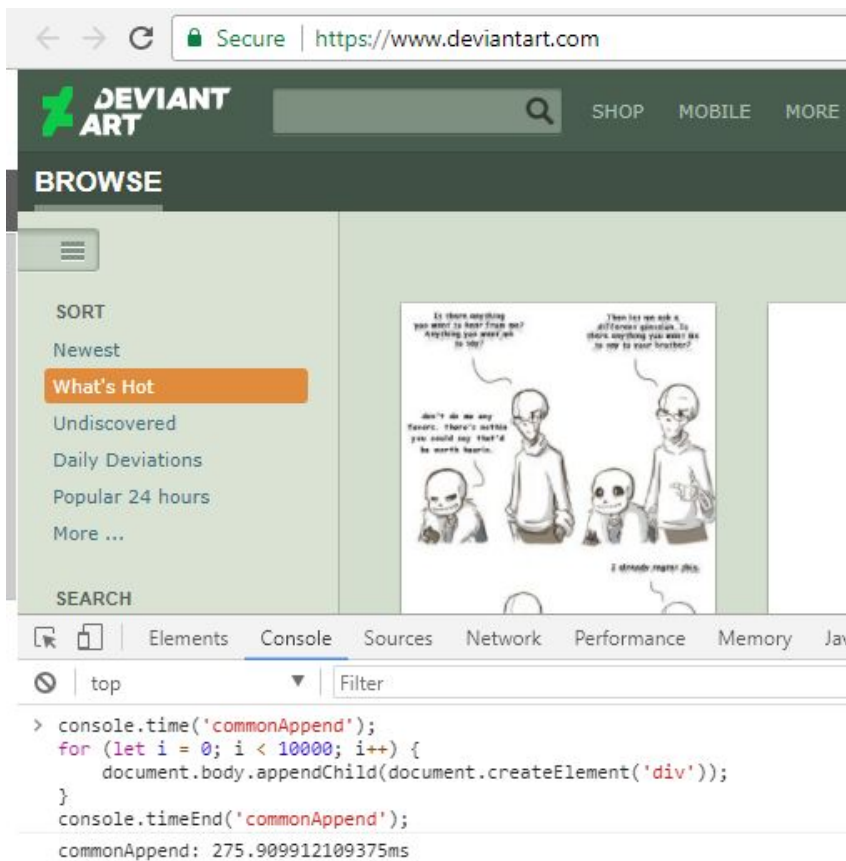
Async Dom

Тестируем



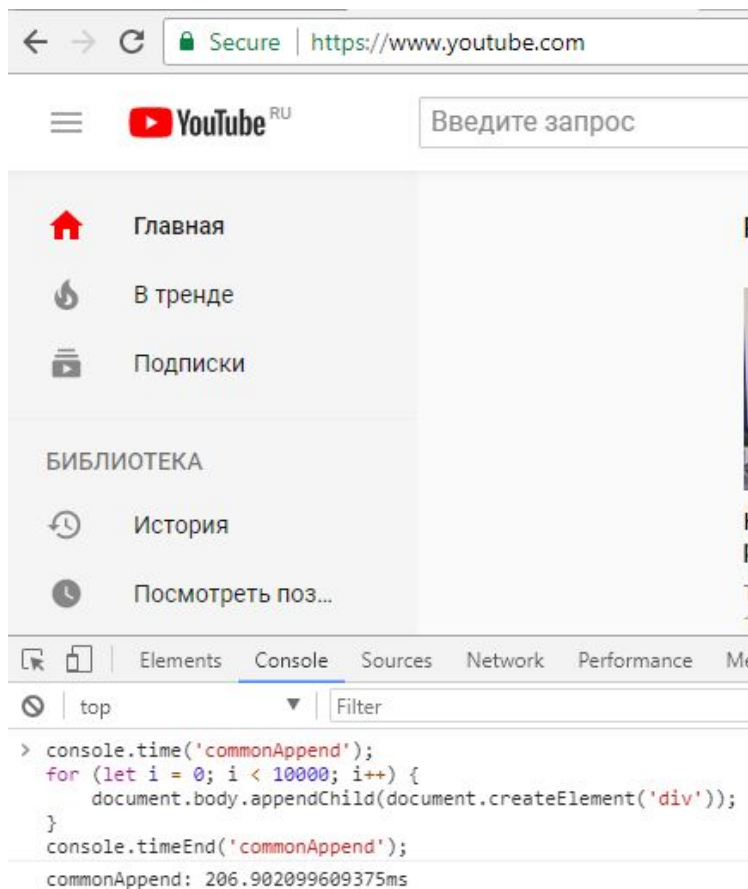
Async Dom

Тестируем



Async Dom

Тестируем



Async Dom

Тестируем



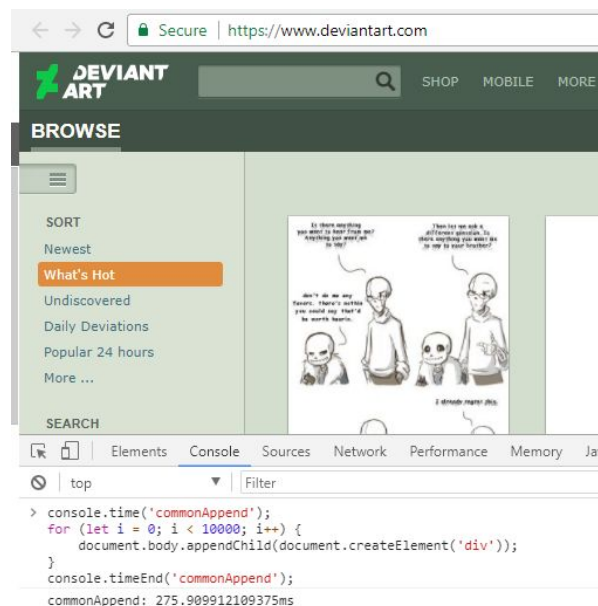
Async Dom

Почему один и тот-же кусок JS кода работает с разной скоростью?



The screenshot shows the MDN web docs page for 'Using JavaScript'. The browser's developer console is open, displaying the following JavaScript code and its execution time:

```
> console.time('commonAppend');  
for (let i = 0; i < 10000; i++) {  
  document.body.appendChild(document.createElement('div'));  
}  
console.timeEnd('commonAppend');  
commonAppend: 57.537109375ms
```

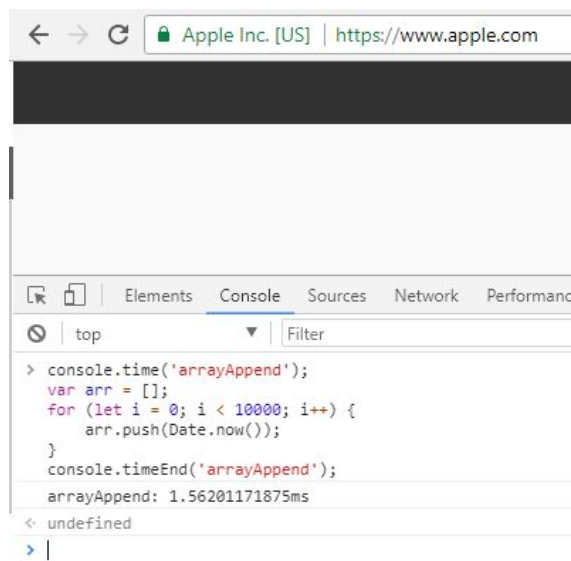


The screenshot shows the DeviantArt website. The browser's developer console is open, displaying the same JavaScript code as the first screenshot, with a different execution time:

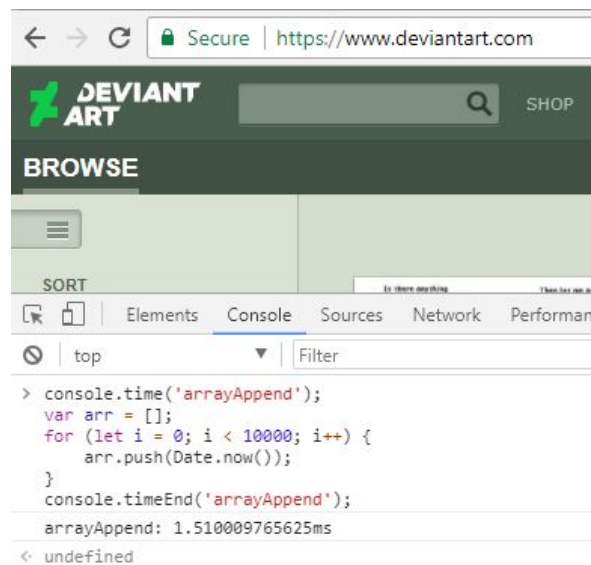
```
> console.time('commonAppend');  
for (let i = 0; i < 10000; i++) {  
  document.body.appendChild(document.createElement('div'));  
}  
console.timeEnd('commonAppend');  
commonAppend: 275.909912109375ms
```

Async Dom

Почему один и тот-же кусок JS кода работает с разной скоростью?



```
<  →  ↻  Apple Inc. [US] | https://www.apple.com  
  
Elements Console Sources Network Performance  
top ▼ Filter  
> console.time('arrayAppend');  
var arr = [];  
for (let i = 0; i < 10000; i++) {  
  arr.push(Date.now());  
}  
console.timeEnd('arrayAppend');  
arrayAppend: 1.56201171875ms  
< undefined  
> |
```



```
<  →  ↻  Secure | https://www.deviantart.com  
  
DEVIAN  
ART  
SHOP  
BROWSE  
SORT  
Elements Console Sources Network Performance  
top ▼ Filter  
> console.time('arrayAppend');  
var arr = [];  
for (let i = 0; i < 10000; i++) {  
  arr.push(Date.now());  
}  
console.timeEnd('arrayAppend');  
arrayAppend: 1.510009765625ms  
< undefined
```

Async Dom

Почему один и тот-же кусок JS кода работает с разной скоростью?

- потому что **Layout**.

<http://kellegous.com/j/2013/01/26/layout-performance/>

<https://gist.github.com/paulirish/5d52fb081b3570c81e3a>

Async Dom

Что призывает кракена?

Element

Box metrics

- `elem.offsetLeft`, `elem.offsetTop`, `elem.offsetWidth`, `elem.offsetHeight`, `elem.offsetParent`
- `elem.clientLeft`, `elem.clientTop`, `elem.clientWidth`, `elem.clientHeight`
- `elem.getClientRects()`, `elem.getBoundingClientRect()`

Scroll stuff

- `elem.scrollBy()`, `elem.scrollTo()`
- `elem.scrollIntoView()`, `elem.scrollIntoViewIfNeeded()`
- `elem.scrollWidth`, `elem.scrollHeight`
- `elem.scrollLeft`, `elem.scrollTop` also, setting them

Focus

- `elem.focus()` can trigger a *double* forced layout ([source](#))

Also...

- `elem.computedRole`, `elem.computedName`
- `elem.innerText` ([source](#))

getComputedStyle

`window.getComputedStyle()` will typically force style recalc ([source](#))

Async Dom

Что делать?

- не писать код, который не умеет батчить работу с DOM

Async Dom

Что делать?

- не писать код, который не умеет батчить работу с DOM
- вынести Logic в отдельный поток, пускай View сам с собой разбирается

Async Dom

Что делать?

- не писать код, который не умеет батчить работу с DOM
- вынести Logic в отдельный поток, пускай View сам с собой разбирается
- использовать Async Dom

Async Dom

Немного подробнее?

Выносим всю бизнес-логику в **WebWorker** / **node.js** / в браузер соседа

Async Dom

Немного подробнее?

Выносим всю бизнес-логику в **WebWorker** / **node.js** / в браузер соседа

- но я не хочу ничего переписывать!

Async Dom

Немного подробнее?

Выносим всю бизнес-логику в **WebWorker** / **node.js** / в браузер соседа

- но я не хочу ничего переписывать!
- используй **PseudoDom**

Async Dom

PseudoDom это:

- Библиотека, эмулирующая браузерный DOM
- Простая как тапок
- Сама всё синхронизирует с View слоем

Async Dom

А что кроме калькулятора удалось запустить?

- Ember.js (1 мб чистейшего JS кода)

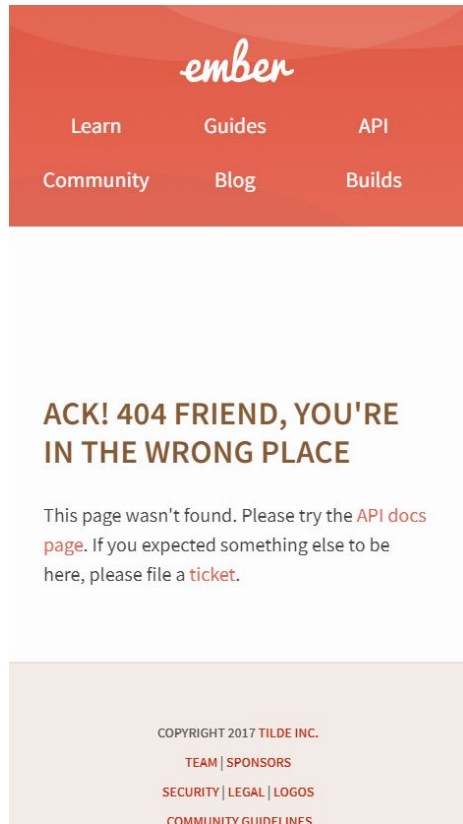
Async Dom

А что кроме калькулятора удалось запустить?

- Ember.js (1 мб чистейшего JS кода)

<https://lifeart.github.io/demo-async-dom/ember/index.html>

<https://lifeart.github.io/demo-async-dom/glimmer-port/index.html>

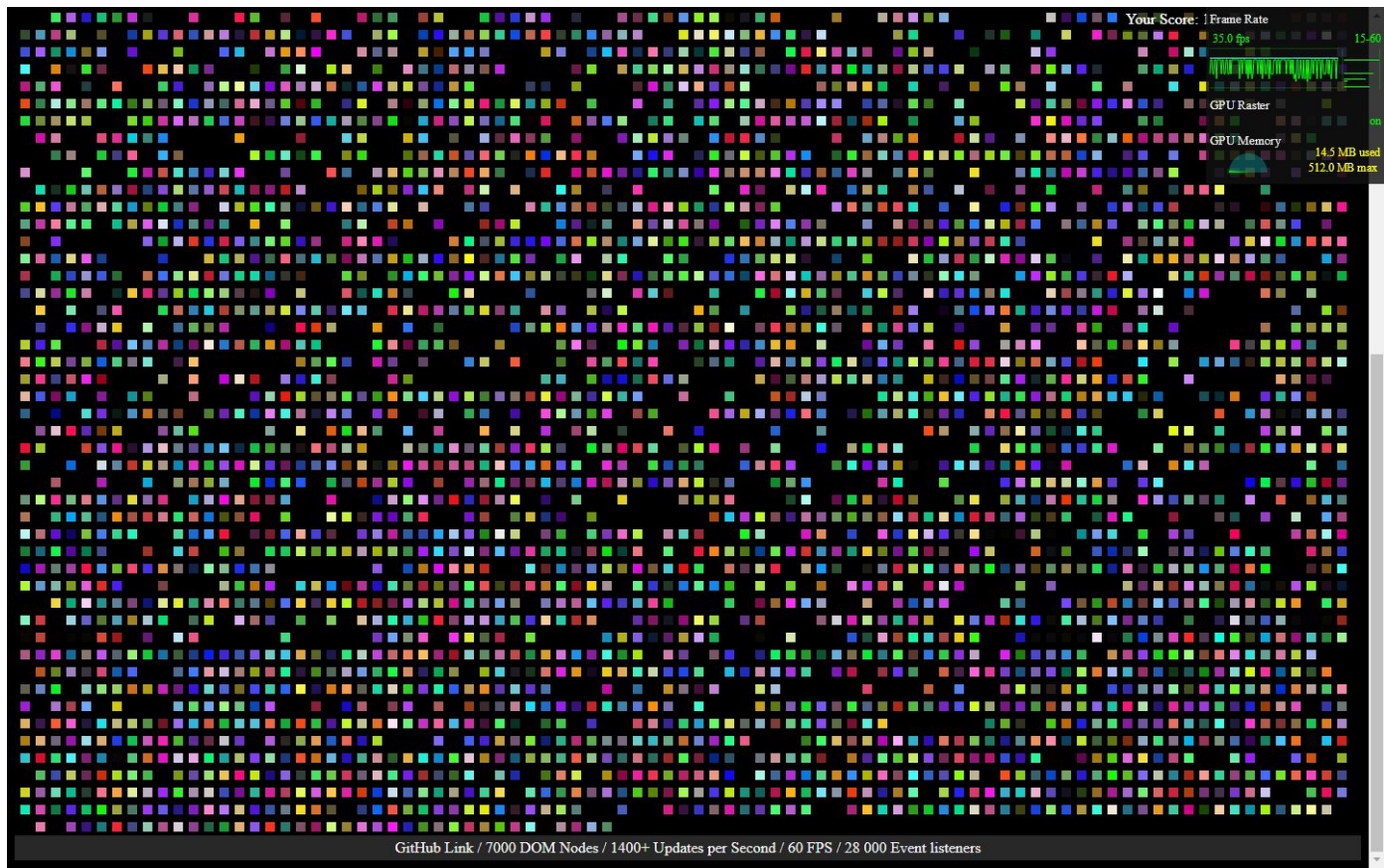


Async Dom

- А что со скоростью?

Async Dom

lifeart.github.io/demo-async-dom/



Async Dom

- 40 **FPS**
- 1400 **DOM** updates per second
- 7000 **DOM** nodes
- 28 000 **Event** listeners

это работает довольно быстро.

Async Dom

А как же оверхэд на транспорте?

Async Dom

А как же оверхэд на транспорте?

Классический вариант:

```
console.time('commonAppend');  
for (let i = 0; i < 10000; i++) {  
    document.body.appendChild(document.createElement('div'));  
}  
console.timeEnd('commonAppend');  
//commonAppend: 62.622802734375ms - 300+ms (depend at layout calculation time, t != const)
```

Async Dom

А как же оверхэд на транспорте?

Модерн:

```
console.time('asyncAppend');
for (let i = 0; i < 10000; i++) {
  let id = i;
  asyncSendMessage({
    action: 'createNode',
    id: id,
    tag: 'div'
  });
  asyncSendMessage({
    action: 'bodyAppendChild',
    id: id
  });
}
console.timeEnd('asyncAppend');
//asyncAppend: 277.938232421875ms (not depend at layout calculation time, t = const)
```

Async Dom

А как же оверхэд на транспорте?

Модерн с группировкой:

```
console.time('asyncAppendGroup');
for (let i = 0; i < 10000; i++) {
  let id = i;
  asyncSendMessage([
    {
      action: 'createNode',
      id: id,
      tag: 'div'
    },
    {
      action: 'bodyAppendChild',
      id: id
    }
  ]);
}
console.timeEnd('asyncAppendGroup');
//asyncAppend: 117.579833984375ms (not depend at layout calculation time, t = const)
```

Async Dom

А как же оверхэд на транспорте?

Модерн с группировкой:

```
console.time('asyncAppendGroup');
for (let i = 0; i < 10000; i++) {
  let id = i;
  asyncSendMessage([
    {
      action: 'createNode',
      id: id,
      tag: 'div'
    },
    {
      action: 'bodyAppendChild',
      id: id
    }
  ]);
}
console.timeEnd('asyncAppendGroup');
//asyncAppend: 117.579833984375ms (not depend at layout calculation time, t = const)
```

Async Dom

А как же оверхэд на транспорте?

Ультра модерн:

```
console.time('asyncAppendBatch');
var msgs = [];
for (let i = 0; i < 10000; i++) {
  let id = i;
  msgs.push({
    action: 'createNode',
    id: id,
    tag: 'div'
  });
  msgs.push({
    action: 'bodyAppendChild',
    id: id
  });
}
asyncSendMessage(msgs);
console.timeEnd('asyncAppendBatch');
//asyncAppend: 23.794189453125ms (not depend at layout calculation time, t = const)
```

Async Dom

Классика VS Ультра-модерн

```
console.time('commonAppend');
for (let i = 0; i < 10000; i++) {
    document.body.appendChild(document.createElement('div'));
}
console.timeEnd('commonAppend');
//commonAppend: 62.622802734375ms - 300+ms (depend at layout calculation time, t != const)
```

```
console.time('asyncAppendBatch');
var msgs = [];
for (let i = 0; i < 10000; i++) {
    let id = i;
    msgs.push({
        action: 'createNode',
        id: id,
        tag: 'div'
    });
    msgs.push({
        action: 'bodyAppendChild',
        id: id
    });
}
asyncSendMessage(msgs);
console.timeEnd('asyncAppendBatch');
//asyncAppend: 23.794189453125ms (not depend
```

Async Dom

Классика VS Ультра-модерн

- 60ms **VS** 20ms

Async Dom

Классика VS Ультра-модерн

- 60ms **VS** 20ms
- т.е. я просто пушу в массив, а мой JS код работает в 3 раза быстрее?

Async Dom

Классика VS Ультра-модерн

- 60ms **VS** 20ms
- т.е. я просто пушу в массив, а мой JS код работает в 3 раза быстрее?
- да, именно так

Async Dom

Бонусы

$t \sim \text{const}$

предсказуемая скорость выполнения клиентского JS кода, без привязи к DOM

Async Dom

А где это можно использовать?

- Увеличение отзывчивости старого кода
- Мультимедийные и встраиваемые системы, где есть HTML и CSS, но слабый JS
- Разработка игр для низкопроизводительных устройств
- В **Pet** проектах
- Интерфейсах статистики с обработкой данных на JS

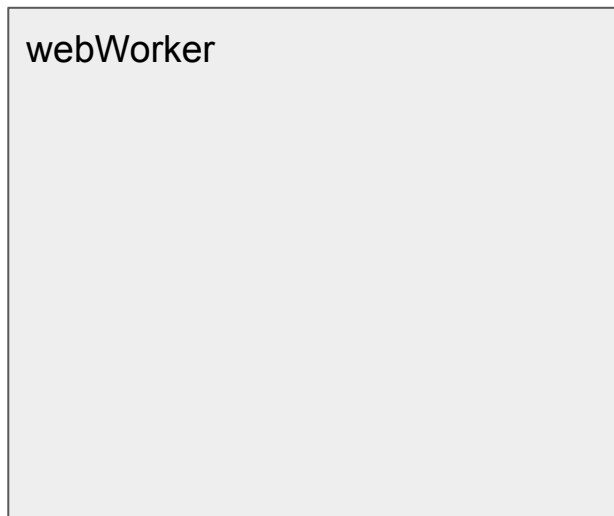
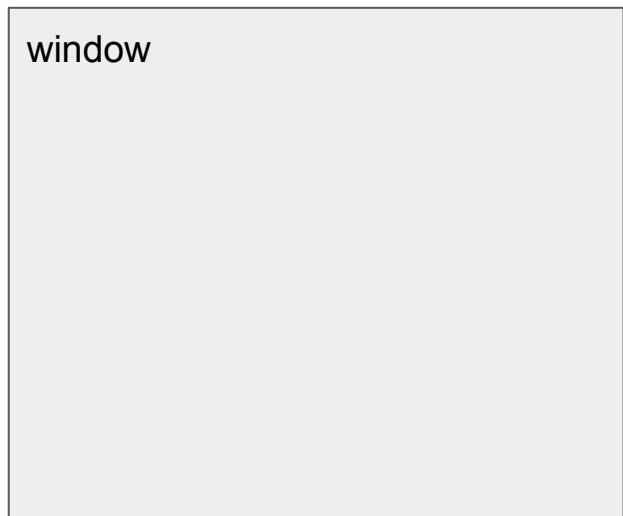
Async Dom

Где посмотреть?

<https://github.com/lifeart/demo-async-dom>

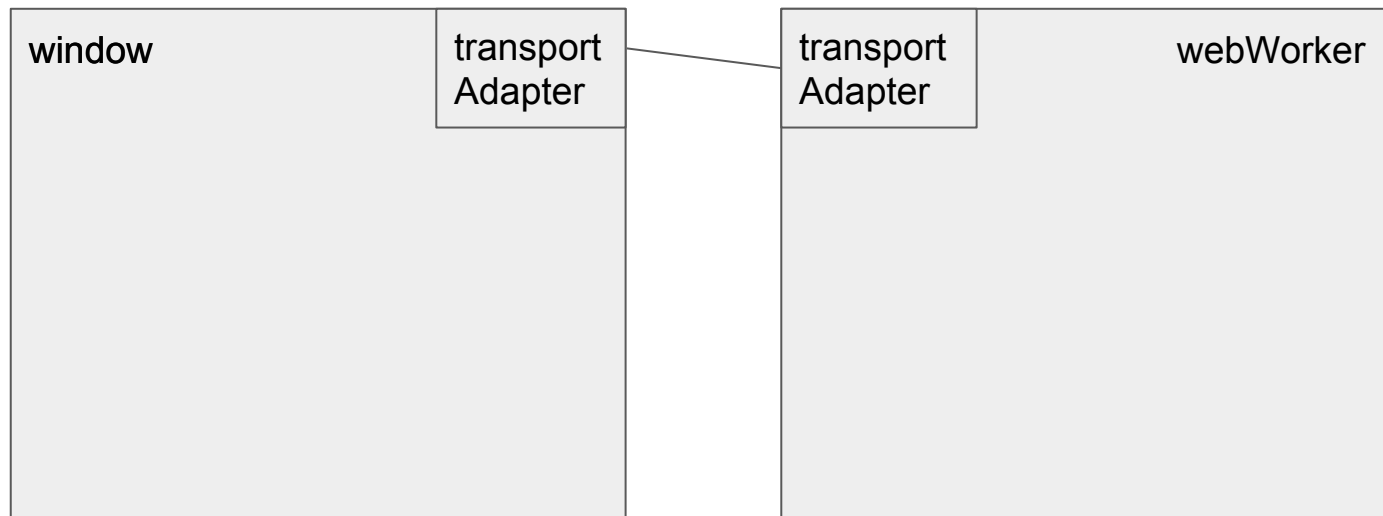
Async Dom

Архитектура



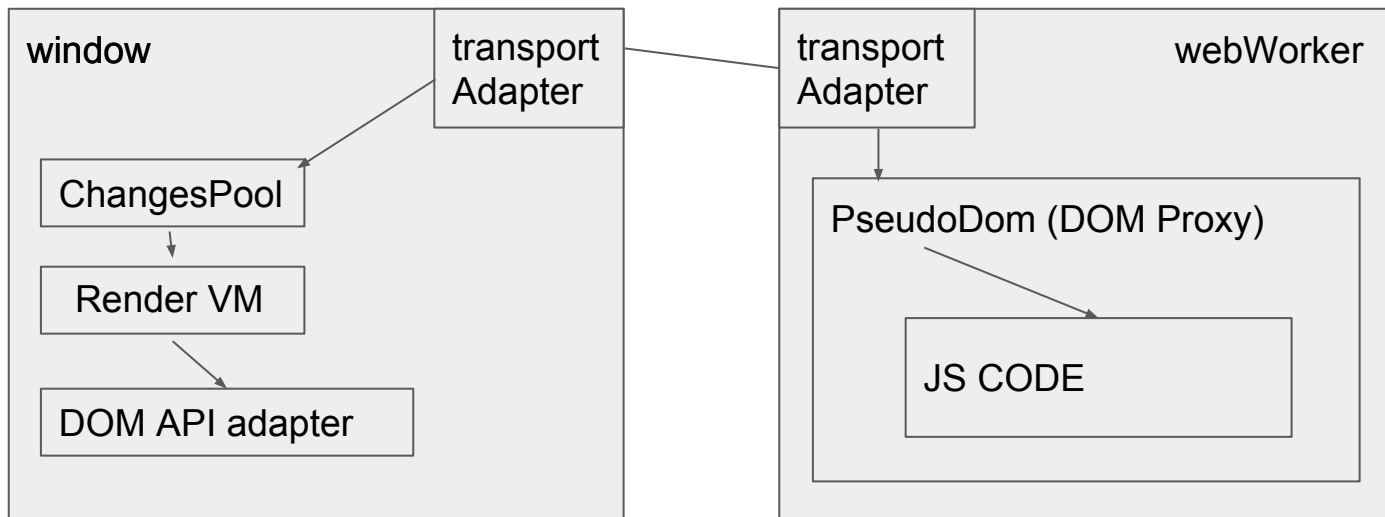
Async Dom

Архитектура



Async Dom

Архитектура



Ember.js / St.Petersburg / Russia

Aleksandr Kanunnikov / lifeart@protonmail.com / <https://github.com/lifeart/>

Ember.js in tg: https://t.me/ember_js