Язык JavaScript (Веб-компоненты)

КОМПОНЕНТНАЯ АРХИТЕКТУРА

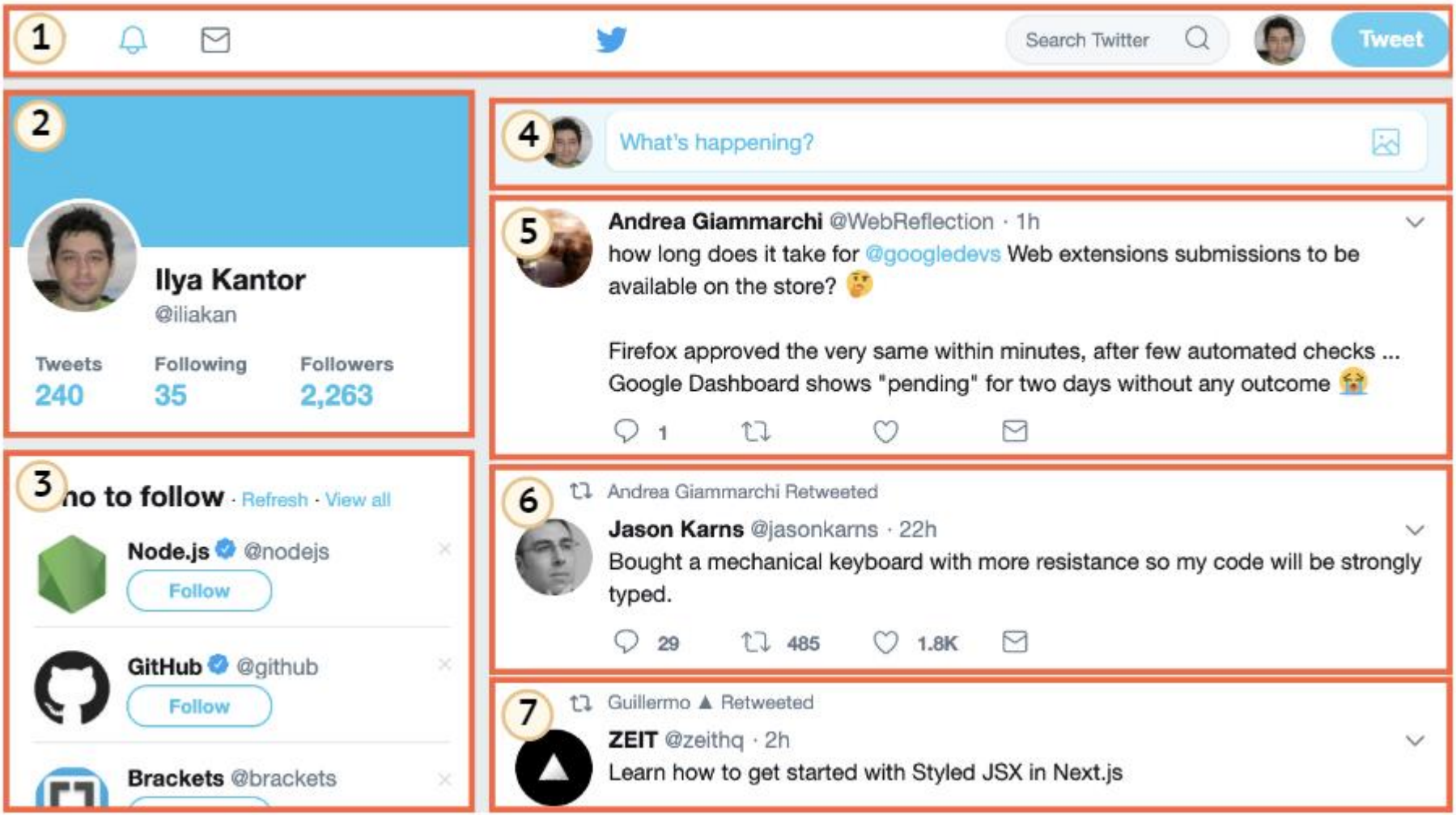
Хорошо известное правило разработки сложного программного обеспечения гласит: не создавай сложное программное обеспечение.

Если что то становится сложным – раздели это на более простые части и соедини наиболее очевидным способом.

Хороший архитектор – это тот, кто может сделать сложное простым.

Мы можем разделить пользовательский интерфейс на визуальные компоненты: каждый из них занимает своё место на странице, выполняет определённую задачу, и отделен от остальных.

КОМПОНЕНТНАЯ АРХИТЕКТУРА



1. Верхняя навигация. 2. Данные пользователя. 3. Предложения подписаться.
4. Форма отправки сообщения. 5. 6. 7. – сообщения.

КОМПОНЕНТНАЯ АРХИТЕКТУРА

Компонент имеет:

- свой собственный JavaScript-класс.
- DOM-структура управляется исключительно своим классом, и внешний код не имеет к ней доступа (принцип «инкапсуляции»).
- CSS-стили, применённые к компоненту.
- API: события, методы класса и т.п., для взаимодействия с другими компонентами.

Существует множество фреймворков и методов разработки для создания компонентов, каждый из которых со своими плюсами и минусами. Обычно особые CSS классы и соглашения используются для эмуляции компонентов – области видимости CSS и инкапсуляция DOM.

«Веб-компоненты» предоставляют встроенные возможности браузера для этого, поэтому нам больше не нужно эмулировать их.

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ

Мы можем создавать пользовательские HTML-элементы, описываемые нашим классом, со своими методами и свойствами, событиями и так далее.

Существует два вида пользовательских элементов:

- Автономные пользовательские элементы – «полностью новые» элементы, расширяющие абстрактный класс HTMLElement.
- Пользовательские встроенные элементы – элементы, расширяющие встроенные, например кнопку HTMLButtonElement и т.п.

Чтобы создать пользовательский элемент, нам нужно сообщить браузеру ряд деталей о нём: как его показать, что делать, когда элемент добавляется или удаляется со страницы и т.д.

Это делается путём создания класса со специальными методами.

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ

Пример шаблона класса автономного пользовательского элемента:

```
class MyElement extends HTMLElement {
  constructor() {
    super(); // элемент создан
  }
  connectedCallback() {
    /* браузер вызывает этот метод при добавлении элемента в документ
(может вызываться много раз, если элемент многократно
добавляется/удаляется) */
  }
  disconnectedCallback() {
    /* браузер вызывает этот метод при удалении элемента из документа
 (может вызываться много раз, если элемент многократно
добавляется/удаляется) */
  }
  static get observedAttributes() {
    return [/* массив имён атрибутов для отслеживания их изменений */];
  }
```

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ

Пример шаблона (продолжение):

```
  attributeChangedCallback(name, oldValue, newValue) {
    // вызывается при изменении одного из перечисленных выше атрибутов
  }

  adoptedCallback() {
    /* вызывается, когда элемент перемещается в новый документ
(происходит в document.adoptNode, используется очень редко) */
  }

  // у элемента могут быть ещё другие методы и свойства
}
```

После создания класса нам нужно зарегистрировать элемент:

```
// сообщаем браузеру, что <my-element> обслуживается классом
customElements.define("my-element", MyElement);
```

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ

Пример:

```
<script>
    class Counter extends HTMLElement {
        constructor() {
            super();
            this.onclick = (e)=>{ this.count++; this.render(); }
        }
        count = 0
        render(){
            this.innerHTML = `<span style="cursor:pointer;">${this.count}</span>`;
        }
        connectedCallback() {
            this.render()
        }
        static get observedAttributes() {
            return ['start'];
        }
```

ПОЛЬЗОВАТЕЛЬСКИЕ ЭЛЕМЕНТЫ

Пример (продолжение):

```
        attributeChangedCallback(name, oldValue, newValue) {
            if(name === 'start'){
                this.count = parseInt(newValue);
            }
            this.render();
        }

    }

    customElements.define("my-counter", Counter);
</script>
<my-counter start="5"></my-counter>
```

МОДИФИЦИРОВАННЫЕ ВСТРОЕННЫЕ ЭЛЕМЕНТЫ

Мы можем расширять и модифицировать встроенные HTML-элементы, наследуя их классы. Пример:

```
<script>
    class Counter extends HTMLButtonElement {
        constructor() {
            super();
            this.onclick = (e)=>{
                this.count++;
                this.render();
            }
        }
        count = 0;
        render(){
            this.innerHTML = this.count;
        }
        connectedCallback() {
            this.render()
        }
```

МОДИФИЦИРОВАННЫЕ ВСТРОЕННЫЕ ЭЛЕМЕНТЫ

Пример (продолжение):

```
        static get observedAttributes() {
            return ['start'];
        }

        attributeChangedCallback(name, oldValue, newValue) { // (4)
            if(name === 'start'){
                this.count = parseInt(newValue);
            }
            this.render();
        }

    }

    customElements.define("my-counter", Counter, {extends: 'button'});
</script>
<button is="my-counter" start="5"></button>
```

SHADOW DOM

Теневой DOM («Shadow DOM») используется для инкапсуляции. Благодаря ему в компоненте есть собственное «теневое» DOM-дерево, к которому нельзя просто так обратиться из главного документа, у него могут быть изолированные CSS-правила и т.п.

Каждый DOM-элемент может иметь 2 типа поддеревьев DOM:

- Light tree – обычное, «светлое», DOM-поддерево, состоящее из HTML-потомков.
- Shadow tree – скрытое, «теневое», DOM-поддерево, не отражённое в HTML, скрытое от посторонних глаз.

Для отображения теневого дерева в Chrome нужно активировать пункт «Show user agent shadow DOM» и посмотрите например на:
<input type="range">

SHADOW DOM

Если у элемента имеются оба поддерева (светлое и теневое), браузер отрисовывает только теневое дерево.

Теневое дерево можно использовать в пользовательских элементах (Custom Elements), чтобы спрятать внутренности компонента и применить к ним локальные стили.

```
<script>
    customElements.define('show-hello', class extends HTMLElement {
      connectedCallback() {
        const shadow = this.attachShadow({mode: 'open'});
        shadow.innerHTML = `<p>
          Привет, ${this.getAttribute('name')}!
        </p>`;
      }
    });
</script>
<show-hello name="Саша"></show-hello>
```

SHADOW DOM

Вызов elem.attachShadow({mode: …}) создаёт теневое дерево.

Есть два ограничения:

- Для каждого элемента мы можем создать только один shadow root.
- В качестве elem может быть использован пользовательский элемент (Custom Element), либо один из следующих элементов: «article», «aside», «blockquote», «body», «div», «footer», «h1…h6», «header», «main» «nav», «p», «section» или «span». Остальные, например, <img>, не могут содержать теневое дерево.

Свойство mode задаёт уровень инкапсуляции. У него может быть только два значения:

- "open" – корень теневого дерева («shadow root») доступен как elem.shadowRoot. Любой код может получить теневое дерево elem.
- "closed" – elem.shadowRoot всегда возвращает null.

До теневого DOM в таком случае мы сможем добраться только по ссылке, которую возвращает attachShadow (и, скорее всего, она будет спрятана внутри класса). Встроенные браузерные теневые деревья, такие как у <input type="range">, закрыты. До них не добраться.

SHADOW DOM

Теневой DOM отделён от главного документа:

- Элементы теневого DOM не видны из обычного DOM через querySelector. В частности, элементы теневого DOM могут иметь такие же идентификаторы, как у элементов в обычном DOM (light DOM). Они должны быть уникальными только внутри теневого дерева.
- У теневого DOM свои стили. Стили из внешнего DOM не применятся.

ЭЛЕМЕНТ "TEMPLATE"

Встроенный элемент <template> предназначен для хранения шаблона HTML. Браузер полностью игнорирует его содержимое, проверяя лишь синтаксис, но мы можем использовать этот элемент в JavaScript, чтобы создать другие элементы.

Особенности <template>:

- его содержимым может быть любой корректный HTML-код, даже такой, который обычно нуждается в специальном родителе (например <tr>).
- браузер рассматривает содержимое <template> как находящееся «вне документа»: стили, определённые в нём, не применяются, скрипты не выполнятся, <video autoplay> не запустится и т.д.
- содержимое шаблона доступно по его свойству content в качестве DocumentFragment – особый тип DOM-узла. Можно обращаться с ним так же, как и с любыми другими DOM-узлами, за исключением одной особенности: когда мы его куда-то вставляем, то в это место вставляется не он сам, а его дети. Для пере использования (вставка в несколько мест) необходимо клонировать узел.

ЭЛЕМЕНТ "TEMPLATE"

```
<template id="tmpl">
   <script>
      alert("Привет");
   </script>
   <div class="message">Привет, Мир!</div>
</template>
<script>
   let elem = document.createElement('div');
   elem.append(tmpl.content.cloneNode(true));
   //elem.append(document.importNode(tmpl.content, true));

   document.body.append(elem);
</script>
```

СЛОТЫ ТЕНЕВОГО DOM, КОМПОЗИЦИЯ

Так же, как встроенный в браузер <select> ожидает получить контент пунктов <option>, и наш пользовательский компонент <custom-tabs> может ожидать, что будет передано фактическое содержимое вкладок, а например <custom-menu> – пунктов меню.

Теневой DOM поддерживает элементы <slot>, которые автоматически наполняются контентом из обычного, «светлого» DOM-дерева.

СЛОТЫ ТЕНЕВОГО DOM, КОМПОЗИЦИЯ

```
<script>
   customElements.define('user-card', class extends HTMLElement {
      connectedCallback() {
         this.attachShadow({mode: 'open'});
         this.shadowRoot.innerHTML = `
            <div>Имя:
               <slot name="username"></slot>
            </div>
            <div>Дата рождения:
               <slot name="birthday"></slot>
            </div>`;
      }
   });
</script>
<user-card>
  <span slot="username">Иван Иванов</span>
  <span slot="birthday">01.01.2001</span>
</user-card> <!-- Атрибут slot="..." могут иметь только дети первого уровня -->
```

СЛОТЫ ТЕНЕВОГО DOM, КОМПОЗИЦИЯ

```
<script>
   customElements.define('user-card', class extends HTMLElement {
      connectedCallback() {
         this.attachShadow({mode: 'open'});
         this.shadowRoot.innerHTML = `
            <div>Имя: <!– содержимое по умолчанию-->
               <slot name="username">Аноним</slot>
            </div>
            <div>Дата рождения:
               <slot name="birthday"></slot>
            </div><slot></slot>`; <!– слот по умолчанию-->
      }
   });
</script>
<user-card>
  <span slot="username">Иван Иванов</span>
  <span slot="birthday">01.01.2001</span>
</user-card> <!-- Атрибут slot="…" могут иметь только дети первого уровня -->
```

ЧТО ПОЧИТАТЬ ДОПОЛНИТЕЛЬНО

https://learn.javascript.ru/web-components - Раздел онлайн учебника «Веб компоненты»