Практикум 1

Практикум: Ссылочные абстракции для взаимодействия с DOM в Angular

Цель практикума

Закрепить знания о работе с `TemplateRef`, `ViewContainerRef`, `@ViewChild`, `@ContentChild` и создании пользовательских структурных директив в Angular через сложные и наглядные задания.

Задание 1: Динамическое управление шаблоном с TemplateRef и ViewContainerRef

Цель: Реализовать механизм динамического добавления и удаления элементов с возможностью передачи данных в шаблон.

Шаги выполнения:

1. Создайте новый Angular компонент `DynamicContainerComponent`.
2. В шаблоне компонента добавьте `<ng-template>`:

   Здесь `ng-template` используется для хранения шаблона, который может быть динамически создан в коде.

   ```
   <ng-template #tpl let-name>
     <p>Привет, {{ name }}!</p>
   </ng-template>
   <button (click)="addTemplate('Алиса')">Добавить Алису</button>
   <button (click)="addTemplate('Боб')">Добавить Боба</button>
   <button (click)="clearTemplates()">Очистить</button>
   ```

3. В классе компонента:

```
import { Component, ViewChild, TemplateRef, ViewContainerRef } from '@
angular/core';

@Component({
  selector: 'app-dynamic-container',
  templateUrl: './dynamic-container.component.html',
})
export class DynamicContainerComponent {
  @ViewChild('tpl', { read: TemplateRef }) tpl!: TemplateRef<any>;

  constructor(private viewContainer: ViewContainerRef) {}

  addTemplate(name: string) {
    this.viewContainer.createEmbeddedView(this.tpl, { name });
  }

  clearTemplates() {
    this.viewContainer.clear();
  }
}
```

- `@ViewChild('tpl', { read: TemplateRef })` получает доступ к шаблону.
- `createEmbeddedView` создает динамический элемент на основе шаблона и передает в него данные.
- `clearTemplates` очищает все вставленные элементы.

4. Запустите приложение и убедитесь, что элементы динамически добавляются и удаляются.

Задание 2: Работа с ViewChild, ContentChild и динамическим контентом

Цель: Создать взаимодействие между родительским и дочерним компонентами с `@ViewChild` и `@ContentChild`.

Шаги выполнения:

1. Создайте `ChildComponent` с разметкой:

   Здесь
   `ng-content` позволяет передавать контент внутрь компонента.

   ```
   <p>Дочерний компонент</p>
   <ng-content></ng-content>
   ```

2. В `ParentComponent` добавьте ссылку на дочерний компонент и переданный контент:

   ```
   <app-child #child>
     <p #contentRef>Этот контент передан из родителя</p>
   </app-child>
   <button (click)="logChildContent()">Логировать контент</button>
   ```

3. В `ParentComponent` используйте `@ViewChild` и `@ContentChild`:

   ```
   import { Component, ViewChild, ContentChild, ElementRef, AfterViewInit }
   from '@angular/core';
   import { ChildComponent } from '../child/child.component';

   @Component({
     selector: 'app-parent',
     templateUrl: './parent.component.html',
   })
   export class ParentComponent implements AfterViewInit {
     @ViewChild('child') child!: ChildComponent;
     @ContentChild('contentRef', { static: true }) content!: ElementRef;

     ngAfterViewInit() {
       console.log('Дочерний компонент:', this.child);
       console.log('Переданный контент:', this.content.nativeElement.textCon
   tent);
   ```

```
  }

  logChildContent() {
    console.log('Динамически вызванный контент:', this.content.nativeEle
ment.textContent);
  }
}
```

- `@ViewChild('child')` получает доступ к экземпляру дочернего компонента.
- `@ContentChild('contentRef', { static: true })` позволяет получить доступ к переданному контенту.
- `ngAfterViewInit` выполняется после инициализации представления и логирует данные.

4. Запустите приложение и убедитесь, что при клике на кнопку выводится содержимое контента.

Задание 3: Создание мощной структурной директивы

Цель: Реализовать директиву, которая будет управлять отображением элементов в зависимости от нескольких условий.

Шаги выполнения:

1. Создайте директиву `appAdvancedIf`.
2. Внедрите `TemplateRef` и `ViewContainerRef`.
3. Реализуйте `@Input() set appAdvancedIf({ condition, delay }: { condition: boolean; delay: number })`, который добавляет или удаляет элемент с задержкой.
4. Пример реализации:

```
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angula
r/core';

@Directive({
```

```
  selector: '[appAdvancedIf]'
})
export class AdvancedIfDirective {
  private timeout: any;

  constructor(private templateRef: TemplateRef<any>, private viewContain
er: ViewContainerRef) {}

  @Input() set appAdvancedIf({ condition, delay }: { condition: boolean; del
ay: number }) {
    clearTimeout(this.timeout);

    if (condition) {
      this.timeout = setTimeout(() => {
        this.viewContainer.createEmbeddedView(this.templateRef);
      }, delay);
    } else {
      this.viewContainer.clear();
    }
  }
}
```

- `TemplateRef` позволяет получить шаблон директивы.
- `ViewContainerRef` управляет вставкой и удалением DOM-элементов.
- Если `condition` истинно, элемент добавляется с задержкой `delay`.
- Если `condition` ложно, элемент удаляется сразу.

5. Используйте директиву:

```
<p *appAdvancedIf="{ condition: isVisible, delay: 1000 }">Этот текст поя
вится с задержкой 1 секунда</p>
<button (click)="isVisible = !isVisible">Переключить</button>
```

6. Убедитесь, что при переключении состояние отображения изменяется с заданной задержкой.