TypeScript: Класове

Екип 14: М. Писина, Н. Ангелова, В. Бонев, К. Колчев

Класове

Традиционният JavaScript използва функции и базирано на прототипи наследяване за изграждане на компоненти за многократна употреба.

Това се оказва неудобно за програмистите, свикнали с обектно-ориентирания подход, където класовете наследяват функционалността и обектите се изграждат от тези класове.

Започвайки с ECMAScript 2015, известен също като ECMAScript 6, програмистите на JavaScript могат да създават своите приложения, използвайки този обектно-ориентиран подход, базиран на класове.

В TypeScript е позволено разработчиците да използват тези техники. Те се компилират до JavaScript, който работи във всички основни браузъри и платформи.

Контрол на достъпа до полета и методи

Модификатор за достъп	Достъпен в класа	Достъпен в подклас	Достъпен външно през инстанция на класа (обект)
public	Да	Да	Да
protected	Да	Да	Не
private	Да	Не	Не

Private

```
1   class Person{
2       public personId: number = 0;
3       private personName: string = "";
4       display(): void{
5           alert("Person Id=" + this.personId + "Name=" + this.personName);
6       }
7   }
8   class Student extends Person{
9       showDetails() {
10          alert(this.personId);
11          alert(this.personName);
```

input.ts 1 of 1 problem

Property 'personName' is private and only accessible within class 'Person'. (234

```
12      }
13  }
```

TypeScript 3.8 синтаксис:

```
class Animal {
 #name: string;
 constructor(theName:
string) {
   this.#name = theName;
 }
}

new Animal("Cat").#name;
// Error
```

Private: TS vs C++

```ts
class Person{
    public personId: number = 0;
    private personName: string = "";
    display(): void{
        alert("Person Id=" + this.personId + "Name=" + this.personName);
    }
}
class Student extends Person{
    showDetails() {
        alert(this.personId);
        alert(this.personName);
    }
}
```

input.ts 1 of 1 problem

```
Property 'personName' is private and only accessible within class 'Person'. (234
```

Компонент	Насл. клас	Достъп в насл. клас
public	private	private
protected		private
private		none

```cpp
class Person {
public:
    void details() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }

    size_t personId;

private:
    std::string personName;
};

class Student: public Person { // or private / protected
public:
    void showDetails() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }
};

void testPrivate(const Person& person) {
        std::cout <<
        "Person ID: " <<
        person.personId <<
        " Name: " <<
        person.personName <<
        std::endl;
}
```

typescript.cpp 2 of 2 problems

```
member "Person::personName" (declared at line 20) is inaccessible
```

Public

```
1   export class Person{
2       public personId: number = 0;
3       public personName: string = "";
4       display(): void{
5           alert("Person Id=" + this.personId + "Name=" + this.personName);
6       }
7    class Student
8   class Student extends Person{
9       showDetails() {
10          alert(this.personId);
11          alert(this.personName);
12      }
13  }
14
```

Public: TS vs C++

```typescript
export class Person{
    public personId: number = 0;
    public personName: string = "";
    display(): void{
        alert("Person Id=" + this.personId + "Name=" + this.personName);
    }
class Student
class Student extends Person{
    showDetails() {
        alert(this.personId);
        alert(this.personName);
    }
}
```

Компонент	Насл. клас	Достъп в насл. клас
public	public	public
protected		protected
private		none

```cpp
class Person {
public:
    void details() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }

    size_t personId;
    std::string personName;
};

class Student: public Person { // or private / protected
public:
    void showDetails() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }
};

void testPublic(const Person& person) {
        std::cout <<
        "Person ID: " <<
        person.personId <<
        " Name: " <<
        person.personName <<
        std::endl;
}
```

Protected

```
export class Person{
    public personId: number = 0;
    protected personName: string = "";
    showPerson(): void{
        alert("Person Id=" + this.personId + "Name=" + this.personName);
    }
}
class Student extends Person{
    showStudent():void {
        alert(this.personId);
        alert(this.personName);
    }
}
```

Protected: TS vs C++

```typescript
export class Person{
    public personId: number = 0;
    protected personName: string = "";
    showPerson(): void{
        alert("Person Id=" + this.personId + "Name=" + this.personName);
    }
}
class Student extends Person{
    showStudent():void {
        alert(this.personId);
        alert(this.personName);
    }
}
```

Компонент	Насл. клас	Достъп в насл. клас
public	protected	protected
protected		protected
private		none

```cpp
class Person {
public:
    void details() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }

    size_t personId;

protected:
    std::string personName;
};

class Student: public Person {
public:
    void showDetails() {
        std::cout <<
        "Person ID: " <<
        this->personId <<
        " Name: " <<
        this->personName <<
        std::endl;
    }
};

void testProtected(const Person& person) {
        std::cout <<
        "Person ID: " <<
        person.personId <<
        " Name: " <<
        person.personName <<
        std::endl;
}
```

typescript.cpp 1 of 1 problem

member "Person::personName" (declared at line 20) is inaccessible

Read-Only

Read-Only Член-Данни

```
class Employee {
   readonly empCode: number;
   empName: string;

   constructor(code: number, name: string)
{
        this.empCode = code;
        this.empName = name;
   }
}

let emp = new Employee(10, "John");
emp.empCode = 20; //Compiler Error
emp.empName = 'Bill'; //Compiler Error
```

Read-Only

Read-Only Интерфейси

```
interface IEmployee {
    readonly empCode: number;
    empName: string;
}

let empObj: IEmployee = {
    empCode: 1,
    empName: "Steve"
}

empObj.empCode = 100; // Compiler Error:
                      // Cannot change
                      // readonly 'empCode'
```

Read-Only

Read-Only Типове

```
interface IEmployee {
    empCode: number;
    empName: string;
}

let emp1: Readonly<IEmployee> = {
    empCode: 1,
    empName: 'Steve'
}

emp1.empCode = 100; // Compiler Error: Cannot
// change readonly 'empCode'
emp1.empName = 'Bill'; // Compiler Error:
// Cannot change readonly 'empName'

let emp2: IEmployee = {
    empCode:1,
    empName:"Steve"
}

emp2.empCode = 100; // OK
emp2.empName = 'Bill'; // OK
```

Read-Only - Имплементация в C++

```cpp
#include <iostream>

class A {
    int x;

public:
    void setX(int x) {
        this->x = x;
    }

    int getX() const {
        return this->x;
    }
};
```

```cpp
#include <iostream>

class A {
    int _x; // Private variable

public:
    A() : x(_x) {} // Bind reference variable x to _x

    void setX(int x) {
        this->_x = x;
    }
    const int &x; // Constant variable
};

// Usage:
int main() {
    A a;
    a.setX(50);

    std::cout << a.x << std::endl; // Correct

    a.x = 12; // Error !
```

reado.cpp 1 of 1 problem

expression must be a modifiable lvalue C/C++(137)

```cpp
    return 0;
}
```

Read-Only - Имплементация в C++

```cpp
#include <iostream>

template <class T>
class ReadOnly {
private:
    const T & real_data;

public:
    ReadOnly(const T & real_data_):real_data(real_data_){}
    ReadOnly & operator=(const ReadOnly &)
    {
        cout << "inside ReadOnly assignment operator" << std::endl;
        return *this;
    }
    operator const T &() {return real_data;}
};

class MyClass {
private:
    int real_x;

public:
    ReadOnly<int> x;

    MyClass():real_x(0),x(real_x) {}
    MyClass(const MyClass & obj):real_x(0),x(real_x) {real_x=obj.real_x;}

    void SetX(int n) {real_x=n;}
};

void set(int &n, int m) { n = m; }
```

```cpp
int main() {
    MyClass a;
    //a.x=1; //ERROR!!!
    a.SetX(1);

    MyClass b(a); //Copy Constructor
    b.SetX(2);

    std::cout << "assigning b to c" << std::endl;
    MyClass c; c=b; //Assignment

    //set(c.x,3); //ERROR!!!
    c.SetX(c.x+1);

    std::cout << "a.x: " << a.x << std::endl;
    std::cout << "b.x: " << b.x << std::endl;
    std::cout << "c.x: " << c.x << std::endl;

    return 0;
}
```

Разлики между extends и implements

Extends

Новият клас е “дете“. Той получава функционалности, идващи с **наследяване**. Той има всички свойства и методи като своя родител. Той може да замени някои от тях и да внедри нови, но родителските неща вече са включени.

Implements

Новият клас може да се третира като същата „форма“, но не е “дете“. Той може да бъде предаден на всеки метод, при който се изисква основния клас, независимо от това, че родителят е различен от основния клас.

Extends - TS

```ts
class Person {
    name: string;
    age: number;

    walk(): void {
      console.log('Walking (person Class)')
    }

    constructor(name: string, age: number) {
      this.name = name;
      this.age = age;
    }
}

class child extends Person {}

// ...

(new child('Mike', 12)).walk();
// logs: Walking(person Class)
```

C++

```cpp
#include <iostream>

class Base {
   public:
    void print() {
        std::cout << "Base Function" << std::endl;
    }
};

class Derived : public Base {
   public:
    void print() {
        std::cout << "Derived Function" << std::endl;
    }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

Implements - TS

```ts
interface Point {
    x: number;
    y: number;
    distance(other: Point): number;
}

class PointImplementation implements Point {
    public x: number;
    public y: number;

    constructor(x: number, y: number) {
        this.x = x;
        this.y = y;
    }

    public distance(other: Point): number {
        return Math.sqrt(
            Math.pow(this.x - other.x, 2) +
            Math.pow(this.y - other.y, 2)
        );
    }
}
```

C++

```cpp
class Point {
public:
    Point(int _x, int _y): x(_x), y(_y) {}
    int x, y;
};

class Shape   // An interface class
{
  public:
    virtual ~Shape() {};
    virtual void draw() = 0;
//...
};

class Line : public Shape
{
  public:
    virtual ~Line();
    virtual void draw(); // implements draw
  private:
    Point start, end;
//...
};
```

Особености на конструкторите

Особености на конструкторите

```
class Car {
    public position: number;
    protected speed: number;

    constructor(position: number, speed: number) {
        this.position = position;
        this.speed = speed;
    }

    move() {
        this.position += this.speed;
    }
}
```

```
class Car {
    constructor(public position: number, protected speed: number) {}

    move() {
        this.position += this.speed;
    }
}
```

Конструкторите в C++

```
class Car {
    Car(double position, double speed) {
        this->position = position;
        this->speed = speed;
    }

    void move() {
        this->position += this->speed;
    }

    double position, speed;
};
```

```
class Car {
    Car(double position, double speed):
        speed(speed), position(position) {}

    void move() {
        this->position += this->speed;
    }

    double position, speed;
};
```

Статични променливи и методи

TypeScript

```typescript
class Circle {
    static pi: number = 3.14;

    static calculateArea(radius:number) {
        return this.pi * radius * radius;
    }
}
Circle.pi; // returns 3.14
Circle.calculateArea(5); // returns 78.5
```

C++

```cpp
class Circle {
public:
    static double pi = 3.14;
    static double calculateArea(double radius) {
        return this->pi * radius * radius;
    }
};
```

Типови параметри

Какво са типовите параметри и за какво можем да ги използваме?

```
function getArray(items : any[] ) : any[] {
    return new Array().concat(items);
}

let myNumArr = getArray([100, 200, 300]);
let myStrArr = getArray(["Hello", "World"]);

myNumArr.push(400); // OK
myStrArr.push("Hello TypeScript"); // OK

myNumArr.push("Hi"); // OK
myStrArr.push(500); // OK

console.log(myNumArr); // [100, 200, 300, 400, "Hi"]
console.log(myStrArr); // ["Hello", "World", "Hello TypeScript", 500]
```

Типови интерфейси

```
interface KeyPair<T, U> {
    key: T;
    value: U;
}

let kv1: KeyPair<number, string> = { key:1, value:"Steve" }; // OK
let kv2: KeyPair<number, number> = { key:1, value:12345 }; // OK
```

Типови класове

```
class KeyValuePair<T,U>
{
    private key: T;
    private val: U;

    setKeyValue(key: T, val: U): void {
        this.key = key;
        this.val = val;
    }

    display():void {
        console.log(`Key = ${this.key}, val = ${this.val}`);
    }
}

let kvp1 = new KeyValuePair<number, string>();
kvp1.setKeyValue(1, "Steve");
kvp1.display(); //Output: Key = 1, Val = Steve

let kvp2 = new KayValuePair<string, string>();
kvp2.SetKeyValue("CEO", "Bill");
kvp2.display(); //Output: Key = CEO, Val = Bill
```

TypeScript

```typescript
class KeyValuePair<T,U>
{
    private key: T;
    private val: U;

    setKeyValue(key: T, val: U): void {
        this.key = key;
        this.val = val;
    }

    display():void {
        console.log(`Key = ${this.key}, val = ${this.val}`);
    }
}

let kvp1 = new KeyValuePair<number, string>();
kvp1.setKeyValue(1, "Steve");
kvp1.display(); //Output: Key = 1, Val = Steve

let kvp2 = new KayValuePair<string, string>();
kvp2.SetKeyValue("CEO", "Bill");
kvp2.display(); //Output: Key = CEO, Val = Bill
```

C++

```cpp
template <typename T, typename U>
class KeyValuePair {
public:
    void setKeyValue(const T key, const U val) {
        this->key = key;
        this->val = val;
    }

    void display() const {
        std::cout << "key = " << key <<
            ", val = " << val << std::endl;
    }

private:
    T key;
    U val;
};

int main() {
    KeyValuePair<int, std::string> kvp1;
    kvp1.setKeyValue(1, "Steve");
    kvp1.display();

    return 0;
}
```

За настроение ;)

CppRussianEdition.cpp cpp.hint DefineHorror.cpp

DefineHorror

```cpp
#include <iostream>
#include "CppRussianEdition.cpp"
using namespace std;

инт main() {
    инт а, б;
    ввод >> а >> б;

    если (а больше б) {
        вывод << "a is greater than b" << конецстр;
    }
    иначе если (а меньше б) {
        вывод << "a is smaller than b" << конецстр;
    }
    иначе {
        вывод << "a is equal to b" << конецстр;
    }

    вывод << "Plus or Minus?" << конецстр;
    строка операция;
    ввод >> операция;
    если (операция равно "Plus") {
        вывод << а плюс б;
    }
    иначе если (операция равно "Minus") {
        вывод << а минус б;
    }
    иначе {
        вывод << "Invalid operation";
    }
}
```

CppRussianEdition.cpp cpp.hint DefineHorror.cpp

DefineHorror

```cpp
#define инт int
#define строка string
#define если if
#define иначе else
#define больше >
#define меньше <
#define равно ==
#define плюс +
#define минус -
#define вывод cout
#define ввод cin
#define конецстр endl
```

Край :)

М. Писина, Н. Ангелова, В. Бонев, К. Колчев