TypeScript static classes

Asked 6 years, 11 months ago Active 1 year, 1 month ago Viewed 137k times

120

15

I wanted to move to TypeScript from traditional JS because I like the C#-like syntax. My problem is that I can't find out how to declare static classes in TypeScript.

In C#, I often use static classes to organize variables and methods, putting them together in a named class, without needing to instatiate an object. In vanilla JS, I used to do this with a simple JS object:

```
var myStaticClass = {
    property: 10,
    method: function(){}
}
```

In TypeScript, I would rather go for my C-sharpy approach, but it seems that static classes don't exist in TS. What is the appropriate solution for this problem ?

javascript class static typescript

edited Jul 12 '17 at 15:30

asked Nov 3 '12 at 19:08

Rayjax
3,824 10 47 69

11 Answers

151

TypeScript is not C#, so you shouldn't expect the same concepts of C# in TypeScript necessarily. The question is why do you want static classes?

In C# a static class is simply a class that cannot be subclassed and must contain only static methods. C# does not allow one to define functions outside of classes. In TypeScript this is possible, however.

```
module M {
    var s = "hello";
    export function f() {
        return s;
    }
}
```

So that you can access M.f() externally, but not s, and you cannot extend the module.

See the TypeScript specification for more details.

edited Aug 31 '18 at 9:33
Fabio Milheiro
4,400 12 45 83

answered Nov 3 '12 at 19:24
Marcus
4,424 2 19 36

so a module can have a static method, but a class cannot? but modules cannot have static data. It's not as convenient as JS for wrapping data and code without having to instantiate anything. – dcsan Aug 24 '15 at 21:23

It might be useful to include that you'll need to include the `.js` in your `html` . So for `Angular 2` you're probably using `System` ... so'd do `System.import("Folder/M");` (or whatever the path is to the compiled `.js` file) before the `bootstrap import` – Serj Sagan Jan 9 '16 at 8:11

6 This is deprecated. Also tslint won't let you do that anymore for modules and namespaces. Read here: palantir.github.io/tslint/rules/no-namespace – Florian Leitgeb Nov 14 '17 at 13:24

I have an use-case for having static classes. Right now, I have a class that only contains static methods. In the project we need to provide some sort of configuration for each class that can be instantiated. Declaring such class as static would not only help me notice that it won't be instantiated but also avoid other developers adding instance members to it. – mdarefull Feb 16 '18 at 15:48

137

Abstract classes have been a first-class citizen of TypeScript since TypeScript 1.6. You cannot instantiate an abstract class.

Here is an example:

```
export abstract class MyClass {
    public static myProp = "Hello";

    public static doSomething(): string {
      return "World";
```

```
//const errors = new MyClass(); // Error
```

edited Feb 19 '18 at 12:43

answered Nov 4 '12 at 19:54
Fenton
169k 48 304 332

4 When dealing with static classes, this is the best response and I upvote this. `Singleton` is for the shared memory pattern of the same class instance. Also, a static class has no instance by definition, so you must throw an exception if the client tries to initialize it. – loretoparisi Jul 28 '16 at 16:03

1 Can we make an abstract class? – KimchiMan Jul 1 '17 at 23:36

@KimchiMan - yes, the `abstract` keyword is supported in TypeScript. – Fenton Jul 2 '17 at 18:48

1 Updated to use `abstract` ! @KimchiMan - great idea! – Obsidian Sep 28 '17 at 19:13

Is this a better approach than marking the constructor as private? – Joro Tenev Feb 19 '18 at 11:12

70

Defining static properties and methods of a class is described in 8.2.1 of the Typescript Language Specification:

```
class Point {
  constructor(public x: number, public y: number) { }
  public distance(p: Point) {
    var dx = this.x - p.x;
    var dy = this.y - p.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
  static origin = new Point(0, 0);
  static distance(p1: Point, p2: Point) {
    return p1.distance(p2);
  }
}
```

where `Point.distance()` is a static (or "class") method.

edited Sep 7 '17 at 12:56
msanford
7,565 7 47 69

answered Nov 3 '12 at 19:31
Rob Raisch
13.5k 1 35 49

methods). – Marcus Nov 3 '12 at 19:45

17 Thanks for the comment. It describes how to create static properties and methods, which taken together allows one to create a class with data and functionality without the need for instantiation. While not specifically a "static class", it does fulfill the requirement as described by the OP's JavaScript example. – Rob Raisch Nov 3 '12 at 19:48

c# didn't have static classes until version 2. they exist in c# only to prevent you instantiating them. you can't do that with javascript so it doesn't make much sense – Simon_Weaver May 23 '14 at 1:59

4 @Simon_Weaver You cant do what in javascript? Prevent classes to be instantiated? Typescript don't care much about runtime anyway, as long as you get a compile error when you try to do stuff you're not allowed to that's all we need. – Alex Mar 1 '16 at 16:41

I guess what I meant was 'we didn't have the static KEYWORD at the class level (which means you can't create an instance of a class)' until version 2. but reading it again I think my comment kind of missed the point anyway. the OP wasn't really looking for a 'static keyword' for the whole class – Simon_Weaver Mar 1 '16 at 16:45

9

This question is quite dated yet I wanted to leave an answer that leverages the current version of the language. Unfortunately static classes still don't exist in TypeScript however you can write a class that behaves similar with only a small overhead using a private constructor which prevents instantiation of classes from outside.

```
class MyStaticClass {
    public static readonly property: number = 42;
    public static myMethod(): void { /* ... */ }
    private constructor() { /* noop */ }
}
```

This snippet will allow you to use "static" classes similar to the C# counterpart with the only downside that it is still possible to instantiate them from inside. Fortunately though you cannot extend classes with private constructors.

answered Apr 25 '18 at 11:57
Christian Ivicevic
4,589 6 27 52

This is one way:

7

```
class SomeClass {
```

`__static_ctor` here is an immediately invoked function expression. Typescript will output code to call it at the end of the generated class.

Update: For generic types in static constructors, which are no longer allowed to be referenced by static members, you will need an extra step now:

```
class SomeClass<T> {
    static myStaticVariable = "whatever";
    private ___static_ctor = (() => { var someClass:SomeClass<T> ; /* do static
constructor stuff :) */ })();
    private static __static_ctor = SomeClass.prototype.___ctor();
}
```

In any case, of course, you could just call the generic type static constructor *after* the class, such as:

```
class SomeClass<T> {
    static myStaticVariable = "whatever";
    private __static_ctor = (() => { var example: SomeClass<T>; /* do static constructor
stuff :) */ })();
}
SomeClass.prototype.__static_ctor();
```

Just remember to NEVER use `this` in `__static_ctor` above (obviously).

edited Apr 28 '17 at 22:35

answered Jul 18 '13 at 16:57
James Wilkins
3,828 2 27 46

This way still emits a constructor for the class. – theycallmemorty Apr 6 '16 at 18:52

1 "This way" is a hack, and doesn't change the compiler's normal operation, as expected. Even `class SomeClass {}` generates a constructor - hardly worth commenting on as though a new issue is introduced. ;) FYI: There are no real "constructors" in JS - only functions that have a "this" when called on objects, or via `new`. This exists no matter what for any "class". – James Wilkins Apr 7 '16 at 0:37

I got the same use case today(31/07/2018) and found this to be a workaround. It is based on my research and it worked for me

5

```
var myStaticClass = {
    property: 10,
    method: function(){}
}
```

I did this:

```
//MyStaticMembers.ts
namespace MyStaticMembers {
        class MyStaticClass {
           static property: number = 10;
           static myMethod() {...}
        }
        export function Property(): number {
           return MyStaticClass.property;
        }
        export function Method(): void {
           return MyStaticClass.myMethod();
        }
     }
```

Hence we shall consume it as below:

```
//app.ts
/// <reference path="MyStaticMembers.ts" />
    console.log(MyStaticMembers.Property);
    MyStaticMembers.Method();
```

This worked for me. If anyone has other better suggestions please let us all hear it !!! Thanks...

answered Jul 31 '18 at 9:08
KoRa
51 1 3

Static classes in languages like C# exist because there are no other top-level constructs to group data and functions. In JavaScript, however, they do and so it is much more natural to just declare an object like you did. To more closely mimick the class syntax, you can declare methods like so:

```
const myStaticClass = {
    property: 10,

    method() {

    }
}
```

answered Mar 31 '17 at 5:21

Yogu
6,638 2 26 46

1 Doesn't this approach muddle your workflow? On the one hand you use classes and instances, and then suddenly you start declaring data in regular old JS objects again... Using static classes also aids readabilty, it's not just about the inner workings of the language. – Kokodoko Apr 19 '17 at 11:41

4 I don't find it messing with my workflow; on the contrary, I find it very convenient to use classless JavaScrpit objects or just plain functions and constants in a module. However, I also try to avoid having global state as much as possible, so I rarely have a need for something like static class variables. – Yogu Apr 20 '17 at 7:23

1

See http://www.basarat.com/2013/04/typescript-static-constructors-for.html

This is a way to 'fake' a static constructor. It's not without its dangers - see the referenced codeplex item.

```
class Test {
    static foo = "orig";

    // Non void static function
    static stat() {
        console.log("Do any static construction here");
        foo = "static initialized";
        // Required to make function non void
        return null;
    }
    // Static variable assignment
    static statrun = Test.stat();
}

// Static construction will have been done:
```

edited Mar 14 '17 at 23:11
senshin
7,881 5 35 48

answered May 23 '14 at 2:03
Simon_Weaver
80.6k 68 498 553

1

One possible way to achieve this is to have static instances of a class within another class. For example:

```
class SystemParams
{
  pageWidth:  number = 8270;
  pageHeight: number = 11690;
}

class DocLevelParams
{
  totalPages: number = 0;
}

class Wrapper
{
  static System: SystemParams = new SystemParams();
  static DocLevel: DocLevelParams = new DocLevelParams();
}
```

Then parameters can be accessed using Wrapper, without having to declare an instance of it. For example:

```
Wrapper.System.pageWidth = 1234;
Wrapper.DocLevel.totalPages = 10;
```

So you get the benefits of the JavaScript type object (as described in the original question) but with the benefits of being able to add the TypeScript typing. Additionally, it avoids having to add 'static' in front of all the parameters in the class.

answered Nov 28 '17 at 17:00
Steve Roberts
2,086 1 10 8

0

I am working on a BulkLoader class to load different types of files and wanted to use the Singleton pattern for it. This way I can load files from my main application class and retrieve the loaded files easily from other classes.

Below is a simple example how you can make a score manager for a game with TypeScript and the Singleton pattern.

class SingletonClass {

```
private static _instance:SingletonClass = new SingletonClass();

private _score:number = 0;

constructor() {
    if(SingletonClass._instance){
        throw new Error("Error: Instantiation failed: Use SingletonDemo.getInstance()
instead of new.");
    }
    SingletonClass._instance = this;
}

public static getInstance():SingletonClass
{
    return SingletonClass._instance;
}

public setScore(value:number):void
{
    this._score = value;
}

public getScore():number
{
    return this._score;
}

public addPoints(value:number):void
{
    this._score += value;
}

public removePoints(value:number):void
{
    this._score -= value;
}   }
```

```
var scoreManager = SingletonClass.getInstance();
scoreManager.setScore(10); scoreManager.addPoints(1);
scoreManager.removePoints(2); console.log( scoreManager.getScore() );
```

answered May 9 '16 at 17:34
Devin Stokes
103 2 12

-1

You can also use keyword `namespace` to organize your variables, classes, methods and so on. See doc

```
namespace Validation {
    export interface StringValidator {
        isAcceptable(s: string): boolean;
    }

    const lettersRegexp = /^[A-Za-z]+$/;
    const numberRegexp = /^[0-9]+$/;

    export class LettersOnlyValidator implements StringValidator {
        isAcceptable(s: string) {
            return lettersRegexp.test(s);
        }
    }

    export class ZipCodeValidator implements StringValidator {
        isAcceptable(s: string) {
            return s.length === 5 && numberRegexp.test(s);
        }
    }
}
```

answered Dec 23 '17 at 6:52
jaguar7021
1 1

See Florian's comment above "This is deprecated. Also tslint won't let you do that anymore for modules and namespaces. Read here: palantir.github.io/tslint/rules/no-namespace" – reggaeguitar Sep 10 '18 at 15:57