
TypeScript
@Decorators

# @Decorators

- What is it and why use it ?
- How to write decorators
- At transpiling time
- At execution time
- Examples (how to write decorators)

## Enabling experimental feature, hang tight...

```
// tsconfig.json
{
    "compilerOptions": {
        "target": "ES5",
        "experimentalDecorators": true
    }
}
```

*proposed standard for ECMAScript 7 (stage 1)*

## What is it and why use it ?

> *“ A decorator is the name used for a software design pattern. Decorators dynamically alter the functionality of a function, method, or class without having to directly use subclasses or change the source code of the function being decorated.*

```
# Use of decorators in Python, with Flask

from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

# What is it and why use it ?

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';
import { AppComponent }  from './app.component';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    AppComponent
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

- *"mixin like" vs inheritance*
- *maintainability*
- *syntax sugar*

# Many types of decorators

```
@classDecorator
class Bar {
   @methodDecorator
   method(@paramDecorator x) {}

   @accessorDecorator
   get accessor() {}

   @propertyDecorator
   prop;
}
```

# Three ways to define decorators

# Three ways to define decorators

## Decorator (5 types)

```
@classDecorator
class Bar {
   @methodDecorator
   foo(@paramDecorator x) {}

   @accessorDecorator
   get accessor() {}

   @propertyDecorator
   prop;
}
```

# Three ways to define decorators

## Decorator (5 types)

```
@classDecorator
class Bar {
   @methodDecorator
   foo(@paramDecorator x) {}

   @accessorDecorator
   get accessor() {}

   @propertyDecorator
   prop;
}
```

## Decorator factory

```
class Bar {
   @dec
   foo(@dec x) {}

   @dec
   get accessor() {}

   @dec
   prop;
}
```

# Three ways to define decorators

## Decorator (5 types)

```
@classDecorator
class Bar {
   @methodDecorator
   foo(@paramDecorator x) {}

   @accessorDecorator
   get accessor() {}

   @propertyDecorator
   prop;
}
```

## Decorator factory

```
class Bar {
   @dec
   foo(@dec x) {}

   @dec
   get accessor() {}

   @dec
   prop;
}
```

## Configurable decorator

```
class Bar {
   @deprecated({
      since: 'v1.0'
   })
   foo(x) {}
}
```

# How to write decorators

## Method decorator case

```
declare type MethodDecorator = <T>(
    target: Object,
    propertyKey: string | symbol,
    descriptor: TypedPropertyDescriptor<T>
) => TypedPropertyDescriptor<T> | void;
```

```
function consoleIn(target, key, desc) {
    desc.value = () => {
        console.log('in');
        desc.value.call(desc, desc.value);
    }
    return desc;
}

function consoleOut(target, key, desc) {
    desc.value = () => {
        desc.value.call(desc, desc.value);
        console.log('out');
    }
    return desc;
}
```

# Method decorator explained

```
class Bar {
    @consoleOut
    @consoleIn
    foo(x) { console.log('foo'); }
}

(new Bar).foo()

// print
// "in"
// "foo"
// "out"

// same as :

consoleOut(
   consoleIn(
      (new Bar).foo()
   )
)
```

# At transpiling time

Method decorator case

```
// Emit the call to __decorate. Given the following:
//
//   class Bar {
//     @consoleOut
//     @consoleIn
//     foo() {}
//   }
//
// The emit for a method is:
//
//   __decorate([
//       consoleOut,
//       consoleIn
//   ], Bar.prototype, "foo", null);
```

# At execution time

## Method decorator case

```
//
//  TypeScript                      | Javascript
//  --------------------------------|------------------------------------
//  @dec                            | let Bar = class Bar {
//  class Bar {                     | }
//     @consoleOut @consoleIn foo() | // [...]
//  }                               | __decorate([consoleOut, consoleIn], Bar.prototype, 'foo', null);
//  --------------------------------|------------------------------------

var __decorate = (this && this.__decorate) || function (decorators, target, key, desc) {
    var c = arguments.length,
        r = ((c < 3) ? target : desc === null) ? desc = Object.getOwnPropertyDescriptor(target, key) : desc,
        d;
    if (typeof Reflect === "object" && typeof Reflect.decorate === "function") {
      // Reflect.decorate is defined in 'reflect-metadata' npm package (polyfill)
      r = Reflect.decorate(decorators, target, key, desc);
    } else {
      // with
      // - `c` : arguments length
      // - `d` : current decorator
      // - `r` : descriptor of target OR target
      for (var i = decorators.length - 1; i >= 0; i--) {
        if (d = decorators[i]) {
          r = (c < 3 ? d(r) : c > 3 ? d(target, key, r) : d(target, key)) || r;
        }
      }
    }
    return c > 3 && r && Object.defineProperty(target, key, r), r;
  };
```

# At execution time

Method decorator case

```
function consoleIn(target, key, desc) {
    desc.value = () => {
        console.log('in');
        desc.value.call(desc, desc.value);
    }
    return desc;
}

function consoleOut(target, key, desc) {
    desc.value = () => {
        desc.value.call(desc, desc.value);
        console.log('out');
    }
    return desc;
}
```

```
consoleOut(
    Bar.prototype,
    'method',
    consoleIn(
        Bar.prototype,
        'foo',
         fooDescriptor
    )
)
```

# Examples

```
import deprecated from 'deprecated-decorator';

class Foo {
    @deprecated('otherMethod')
    method() { }

    @deprecated({
        alternative: 'otherProperty',
        version: '0.1.2',
        url: 'http://vane.life/'
    })
    get property() { }
}
```

vilic/deprecated-decorator

# @SpotifyQuery

## Spotify Web API + GraphQL

- Lot of duplicate code → maintainability issue
- Domain specific decorator → maintainability solution
- Good example of special use of decorators

```
class Fetcher {

  @SpotifyQuery(`
    query getTrack($id: String!) {
      track(id: $id) {
        name
        artists {
          name
        }
      }
    }
  `)
  static getTrack (response?): any {
    let track = response.track;
    return `the track ${track.name} was played by ${track.artists[0].name}`;
  }
}


Fetcher.getTrack({ id : '3W2ZcrRsInZbjWylOi6KhZ' }).then(sentence => {
  console.log(sentence);
  // print "the track You & Me - Flume Remix was played by Disclosure"
}, console.error);
```

thefrenchhouse/spotify-graphql

```
class Fetcher {

  static getTrack (variables): Promise<string> {
    return new Promise(resolve, reject) => {
      spotifyApiClient.query(`
        query getTrack($id: String!) {
          track(id: $id) {
            name
            artists {
              name
            }
          }
        }
      `, null, null, variables).then((executionContext) => {
        if (!!executionContext.errors) {
          reject(executionContext.errors);
        } else {
          let track = executionContext.data.track;
          resolve(`the track ${track.name} was played by ${track.artists[0].name}`);
        }
      })
    });
  }
}


Fetcher.getTrack({ id : '3W2ZcrRsInZbjWylOi6KhZ' }).then(sentence => {
  console.log(sentence);
  // output "the track You & Me - Flume Remix was played by Disclosure"
}, error => { console.log(error) });
```

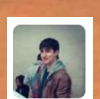
# @SpotifyQuery implementation

```
function SpotifyQuery(query: string) {
   return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
      var originalMethod = descriptor.value;

      descriptor.value = function (variables?) {
         return new Promise((resolve, reject) => {
            spotifyClient.query(query, null, null, variables).then((executionResult) => {
               if (!executionResult.errors) {
                  resolve(
                     originalMethod.call(originalMethod, executionResult.data)
                  );
               } else {
                  reject(executionResult.errors);
               }
            }, reject).catch(reject);
         });
      };

      return descriptor;
   };
}
```

# Any questions ?



@WittyDeveloper

# References


- https://www.typescriptlang.org/docs/handbook/decorators.html
- http://blog.wolksoftware.com/decorators-reflection-javascript-typescript
- https://github.com/wycats/javascript-decorators/blob/master/README.md
- https://www.npmjs.com/package/reflect-metadata
- http://www.2ality.com/2011/09/currying-vs-part-eval.html
- thefrenchhouse/spotify-graphql