Creating and Using Decorators

Brice Wilson

@brice_wilson www.BriceWilson.net

Overview

What are decorators?

How are they implemented?

Decorator syntax

Different type of decorators

What Are Decorators?

Proposed feature for JavaScript

Declarative programming

Implemented as functions

May be attached to the following:

- Classes
- Methods
- Accessors
- Properties
- Parameters

Currently requires the *experimentalDecorators* compiler option

Decorator Syntax

```
function uielement(target: Function) { // do ui stuff }
```

Decorator Syntax

```
function uielement(target: Function) { // do ui stuff }

function deprecated(t: any, p: string, d: PropertyDescriptor)
{
    console.log('This method will go away soon.');
}
```

Decorator Syntax

```
function uielement(target: Function) { // do ui stuff }

function deprecated(t: any, p: string, d: PropertyDescriptor)
{
    console.log('This method will go away soon.');
}
```

Decorator Syntax

```
function uielement(target: Function) { // do ui stuff }
function deprecated(t: any, p: string, d: PropertyDescriptor)
{
    console.log('This method will go away soon.');
}

@uielement
class ContactForm {



}
```

Decorator Syntax

```
function uielement(target: Function) { // do ui stuff }
function deprecated(t: any, p: string, d: PropertyDescriptor)
{
    console.log('This method will go away soon.');
}


@uielement
class ContactForm {
    @deprecated
    someOldMethod() { // ... }
}
```

Decorator Factories

```
function uielement(element: string) {



}
```

Decorator Factories

```
function uielement(element: string) {
    return function(target: Function) {
        console.log(`Creating new element: ${element}`);
    }
}
```

Decorator Factories

```
function uielement(element: string) {
    return function(target: Function) {
        console.log(`Creating new element: ${element}`);
    }
}
```

Decorator Factories

```
function uielement(element: string) {
    return function(target: Function) {
        console.log(`Creating new element: ${element}`);
    }
}
```

Decorator Factories

```
function uielement(element: string) {
    return function(target: Function) {
        console.log(`Creating new element: ${element}`);
    }
}
@uielement('SimpleContactForm')
class ContactForm {
    // contact properties here
}
```

```
// ClassDecorator type

<TFunction extends Function>(target: TFunction) => TFunction | void;
```

Class Decorators

```
// ClassDecorator type

<TFunction extends Function>(target: TFunction) => TFunction | void;
```

Class Decorators

```
// ClassDecorator type

<TFunction extends Function>(target: TFunction) => TFunction | void;
```

Class Decorators

Class constructor will be passed as parameter to decorator

```
// ClassDecorator type

<TFunction extends Function>(target: TFunction) => TFunction | void;
```

Class Decorators

Class constructor will be passed as parameter to decorator

Constructor is replaced if there is a return value

```
// ClassDecorator type

<TFunction extends Function>(target: TFunction) => TFunction | void;
```

Class Decorators

Class constructor will be passed as parameter to decorator

Constructor is replaced if there is a return value

Return void if constructor is not to be replaced

Demo

Creating and using class decorators

Demo

Creating class decorators that replace constructor functions

```
function MyPropertyDecorator(target: Object,
                             propertyKey: string) {
    // do decorator stuff
}
```

Property Decorators

First parameter is either constructor function or class prototype

```
function MyPropertyDecorator(target: Object,
                             propertyKey: string) {
    // do decorator stuff
}
```

Property Decorators

First parameter is either constructor function or class prototype

Second parameter is the name of the decorated member

```
function MyParameterDecorator(target: Object,
                              propertyKey: string,
                              parameterIndex: number) {
    // do decorator stuff
}
```

Parameter Decorators

First parameter is either constructor function or class prototype

```
function MyParameterDecorator(target: Object,
                              propertyKey: string,
                              parameterIndex: number) {
    // do decorator stuff
}
```

Parameter Decorators

First parameter is either constructor function or class prototype

Second parameter is the name of the decorated member

```
function MyParameterDecorator(target: Object,
                              propertyKey: string,
                              parameterIndex: number) {
    // do decorator stuff
}
```

Parameter Decorators

First parameter is either constructor function or class prototype

Second parameter is the name of the decorated member

Third parameter is the ordinal index of the decorated parameter

```
interface PropertyDescriptor {
    configurable?: boolean;
    enumerable?: boolean;
    value?: any;
    writable?: boolean;
    get? (): any;
    set? (v: any): void;
}
```

Property Descriptors

Object that describes a property and how it can be manipulated

```
interface PropertyDescriptor {
    configurable?: boolean;
    enumerable?: boolean;
    value?: any;
    writable?: boolean;
    get? (): any;
    set? (v: any): void;
}
```

Property Descriptors

Object that describes a property and how it can be manipulated

“value” property contains the function definition for class methods

```
interface PropertyDescriptor {
    configurable?: boolean;
    enumerable?: boolean;
    value?: any;
    writable?: boolean;
    get? (): any;
    set? (v: any): void;
}
```

Property Descriptors

Object that describes a property and how it can be manipulated

“value” property contains the function definition for class methods

“writable” property specifies if “value” is read-only

```
function MyMethodDecorator(target: Object,
                           propertyKey: string,
                           descriptor: PropertyDescriptor) {
    // do decorator stuff
}
```

Method and Accessor Decorators

First parameter is either constructor function or class prototype

```
function MyMethodDecorator(target: Object,
                           propertyKey: string,
                           descriptor: PropertyDescriptor) {
    // do decorator stuff
}
```

Method and Accessor Decorators

First parameter is either constructor function or class prototype

Second parameter is the name of the decorated member

```
function MyMethodDecorator(target: Object,
                           propertyKey: string,
                           descriptor: PropertyDescriptor) {
    // do decorator stuff
}
```

Method and Accessor Decorators

First parameter is either constructor function or class prototype

Second parameter is the name of the decorated member

Third parameter is the property descriptor of the decorated member

Demo

Creating and using method decorators

Summary

Declarative programming

Future JavaScript feature

Available in TypeScript now!!!

Syntax

Different types of decorators

Function signatures for each type