Creating and Using TypeScript Decorators

INTRODUCTION TO TYPESCRIPT DECORATORS

David Tucker
TECHNICAL ARCHITECT & CLOUD CONSULTANT
@_davidtucker_ davidtucker.net

TypeScript Decorator Example

```
@controller("/api")
class APIServer {

 @resolver
 @log("Route added to API Server")
 public addRoute(path:string, action:Action) {
  this.app.addRoute(path, action);
 }

}
```

Globomantics

Manufacturing company transitioning to TypeScript for internal API's

Interested in using Decorators to define cross-cutting data rules

Looking look automate route creation for data types using Node with Express

Looking to integrate data validation standards across all data types

Course API Project

Node.js

Server runtime for JavaScript that allows TypeScript server execution

Express

Web application framework for Node.js that will be used to serve our API's

Overview

Introducing TypeScript decorators

Configuring a TypeScript project with support for decorators

Implementing each of the different decorator types

Integrating decorators with Node and Express for route creation

Understanding Decorators in TypeScript

“**Decorators** provide a way to add both annotations and a meta-programming syntax for class declarations and members. Decorators are available as an experimental feature of TypeScript.”

TypeScript Documentation

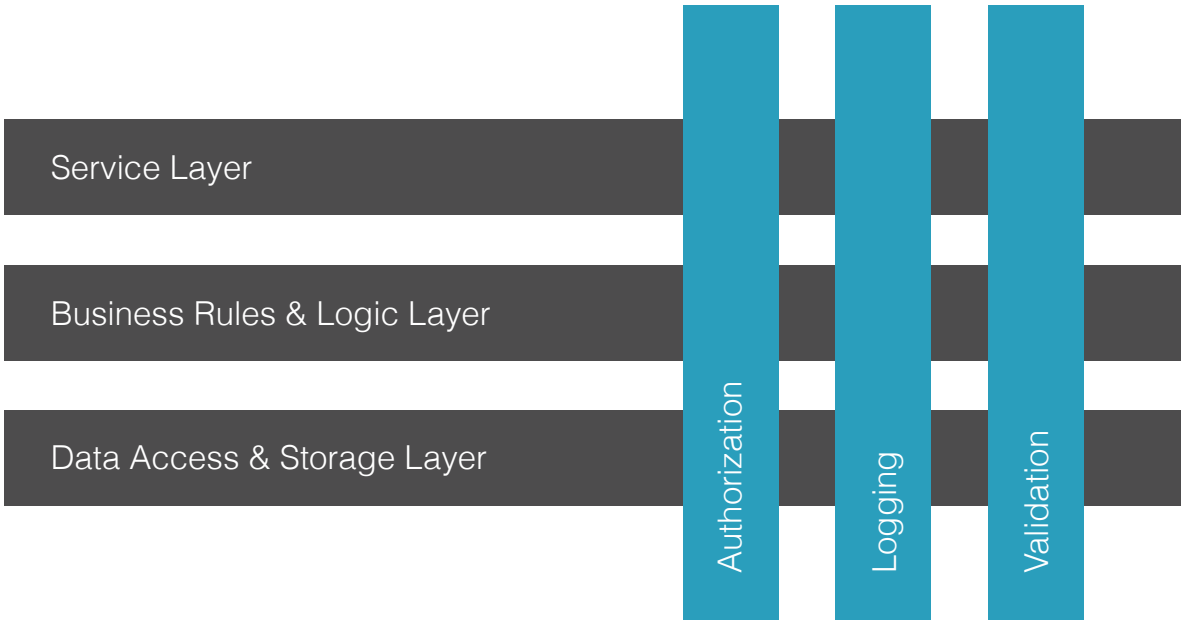
TypeScript Decorator Example

```
@controller("/api")
class APIServer {

 @log("Route added to API Server")
 public addRoute(path:string, action:Action) {
  this.app.addRoute(path, action);
 }

}
```

Cross-cutting Concerns



"In computing, **aspect-oriented programming** (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns."

Wikipedia

Decorators & Annotations

Annotations are limited to setting metadata

Decorators are functions that can modify what they describe when executed

Traceur (from Google) provides support for annotations within JavaScript

Typescript supports decorators as an experimental feature

Types of TypeScript Decorators

Class

Method

Property

Parameter

```
@controller("api")
class APIServer {

 constructor(private app:Express) {}

}
```

Class Decorators

Applied to class constructor for observing, modifying, or replacing class definition

```
class APIServer {

 @log("Route added to API Server")
 public addRoute(path:string, action:Action) {
  this._app.addRoute(path, action);
 }

}
```

Method Decorators

Applied to method property descriptor for observing, modifying, or replacing method definition

```
class APIServer {

 @inject
 public app:Express;

}
```

Property Decorators

Can only be leveraged to observe that a property has been declared for a class

```
class APIServer {

 @validate
 public addStatic(@required path:string, content:string) {
  this._app.addStaticRoute(path, contentPath);
 }

}
```

Parameter Decorators

Applied to method declaration for observing that a parameter has been defined on a method

Creating Decorators

TypeScript Decorator Signatures

TypeScript defines specific function signatures for each of the four different decorator types. As this is an experimental feature, these signatures have changed over time.

```
// Receives the constructor of the class
declare type ClassDecorator = <TFunction extends
Function>(target: TFunction) => TFunction | void;

// Implementation
function log(constructor: Function) {
   console.log(`${constructor} Decorator Invoked`);
}
```

Class Decorators

Applied to class constructor for observing, modifying, or replacing class definition

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

// Implementation
function log(target:Object, propertyKey:string,
descriptor:PropertyDescriptor) {
   console.log(`Method Invoked`);
}
```

Method Decorators

Applied to method property descriptor for observing, modifying, or replacing method definition

```
// Receives the target and propertyKey
declare type PropertyDecorator = (target: Object,
propertyKey: string | symbol) => void;

// Implementation
function log(target: Object, propertyKey: string) {
   console.log(`${propertyKey} Decorator Invoked`);
}
```

Property Decorators

Can only be leveraged to observe that a property has been declared for a class

```
// Receives the target, propertyKey, and param index
declare type ParameterDecorator = (target: Object,
propertyKey: string | symbol, parameterIndex: number) => void;

// Implementation
function log(target: Object, propertyKey: string, index:
number) {
    console.log(`${propertyKey} decorator (${index}) Invoked`);
}
```

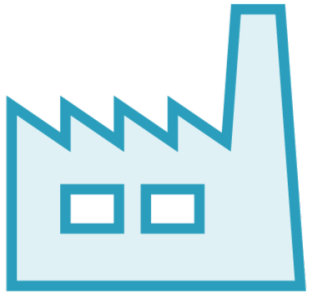
Parameter Decorators

Applied to method declaration for observing that a parameter has been defined on a method

TypeScript Decorator Approaches

Decorator Function

Function with a specific signature evaluated at runtime



Decorator Factory

Function that returns a decorator function that can optionally receive parameters

Decorator Approaches

Decorator functions must implement specific signatures based on use

Decorator factories enable you to receive parameters and construct the decorator

Decorators are called at runtime with either approach

Decorators using factories need to have parenthesis in declaration

Decorator Factory Example

```
class APIServer {
  @enumerable(true)
  getRoutes() {
   return this.routes;
  }
}

function enumerable(val: boolean) {
  return function(target: any, propertyKey: string,
  descriptor: PropertyDescriptor) {
    descriptor.enumerable = val;
  }
}
```

Multiple Decorators

Multiple decorators can be applied in any of the four decorator types

Decorator expressions are evaluated in from top to bottom

Results of expressions are called as functions from bottom to top

This order makes sense if you envision them as nested functions

Multiple Decorators

```
class APIServer {
  @d1
  @d2
  getRoutes() {
   return this.routes;
  }
}

// Order: d1 evaluated, d2 evaluated, d2 called, d1 called
// Structure: d1(d2())
```

Implementing a Basic Decorator

Demo

Create a new TypeScript project in VS Code

Configure TypeScript for Decorator support

Implement each type of TypeScript decorator

Configuring Express Routes with Decorators

Globomantics

Manufacturing company transitioning to TypeScript for internal API's

Interested in using Decorators to define cross-cutting data rules

Looking look automate route creation for data types using Node with Express

Looking to integrate data validation standards across all data types

Demo

Review a base Express project for upcoming demos

Utilize decorators for defining Express routes

Summary

Summary

Introduced TypeScript decorators

Configured a TypeScript project with support for decorators

Implemented each of the four different decorator types

Integrated decorators with Node and Express for route creation