Unification: Decorators + Annotation

*Status: (Draft, Final)*

*Authors: username@angularjs.org*

*This document is published to the web as part of the public* [*Angular Design Docs*](https://drive.google.com/#folders/0BxgtL8yFJbacUnUxc3l5aTZrbVk) *folder*

# Objective

List the use case which decorators/annotations need to meet.

# Background

Decorators allow modifying the target (class, method, function, paramater, property) and even replacing them.

Annotations allow attaching meta-data to (class, method, function, paramater, property).

While annotations can be implemented using decorators, the **key point is to make it so that annotations and decorators are composable**.

# Prior Art

List of existing solutions and their corresponding strong/weak points.

# Use Cases

## Common Annotations used in examples

|  |
| --- |
| // used to declare a list of types which needs to be injected  function Inject(...types) {  return function(target) {  Reflect.setMetadata(target, 'inject', types);  }  }  function Component(selector) {  return function(target) {  Reflect.setMetadata(target, 'component', selector);  }  }  function Template(selector) {  return function(target) {  var templates = Reflect.getMetadata(target, 'templates') || [];  templates.push(target);  Reflect.setMetadata(target, 'templates', templates);  }  } |

## TypeScript implicit type annotation

|  |
| --- |
| function Paramaters(types) {  return function(target) {  Reflect.setMetadata(target, 'paramaters', types);  }  }  function ParamaterAnnotations(annotations) {  return function(target) {  Reflect.setMetadata(target, 'paramaterAnnotations', annotations);  }  } |

## Common Decorators used in examples

|  |
| --- |
| function log(target) {  return function() {  console.log(target.name);  return target.apply(this, arguments);  }  }  function profile(target) {  return function() {  var start = performance.now()  try {  return target.apply(this, arguments);  } finally {  console.log(target.name, performance.now() - start);  }  }  } |

## Usecase: function Injection

|  |
| --- |
| describe('some test', () => {  it('should do something', @Inject() (a:DepA) => {  ...  });  }); |

TypeScript will transpile this into:

|  |
| --- |
| describe('some test', () => {  it('should do something', @Inject() @Paramaters([DepA]) (a) => {  ...  });  }); |

## Usecase: constructor Injection

|  |
| --- |
| // Typescript  @Component('some-component')  class {  // @Paramaters([(DepA, DepB)]) is autogenerated  // @ParamaterAnnotation([null, [parent]]) is autogenerated  constructor(a:DepA, @parent b:DepB) {  }  } |

|  |
| --- |
| // ES6 (without typescript requires explicit types)  @Component('some-component')  class {  // For readability the annotation should be as close to  // constructor as possible.  @Inject(DepA, @parent DepB)  constructor(a, b) {  }  } |

* While class and constructor method are the same thing, it sometimes makes logical sense to put annotation on the class and other times on constructor. For example in the case of @Inject() the list of dependencies should be as close to method parameters as possible to easily catch errors.

## Usecase: Annotation non-Inheritance

|  |
| --- |
| @Component('tab');  @Template({url: '...';})  @Template({locale: 'us', url: '...'}  class Tab {}  @Component('ios-tab');  @Template({platform: 'ios', url: '...';})  class IOSTab extends Tab {}  class MockTab extends Tab {} |

* It is upto the framework to decide if an annotation inherits to children or not. (Use case specific)
* MockTab has no @Component and so it can not be used in HTML. (It would be wrong to assume that <tab> is MockTab.)
* IOSTab has different set of @Templates than Tab. It would be wrong to merge them.

## Usecase: Mixin

|  |
| --- |
| class AuthMixin {  @Inject()  onLogin(a:A, b:B) {}  @Inject()  onLogout(b:B, c:C) {}  }  @Component(...)  @Template(...)  class MyApp {  }  Mixin(MyApp, AuthMixin);// copies methods from AuthMixin to MyApp |

* Annotations need to travel with the mixin method.

## Usecase: Composability

|  |
| --- |
| @Component(...)  @Template(...)  class MyApp {  @log  @Inject(DepA, DepB)  @profile  constructor(a, b) {  }  @log  @Inject(DepA, DepB)  @profile  someMethod(a, b) {  }  } |

* Both @log and @profile replace constructor/method instance. The annotation should be properly preserved

## Usecase: Tearoff Methods

|  |
| --- |
| @Component()  class MyApp {  constructor(@Query(MyApp.prototype.predicate) query) {  }  predicate(@parent obj) {  }  } |

## Usecase:

|  |
| --- |
|  |

## Usecase:

|  |
| --- |
|  |

## Usecase:

|  |
| --- |
|  |

# Requirements

* Annotations should be extractable from a function ref. (In practice it means that the annotations must be on the method which they are decorating.)

# Detailed Design

Details on how you’ll implement. Should be in sufficient detail for other engineers to materially comment on structure to affect the end result.

# Caveats

You may need to describe what you did not do or why simpler approaches don't work. Mention other things to watch out for (if any).

# Security Considerations

How you’ll be secure

# Performance Considerations / Test Strategy

How you’ll be fast.

# Work Breakdown

Description of development phases and approximate time estimates.