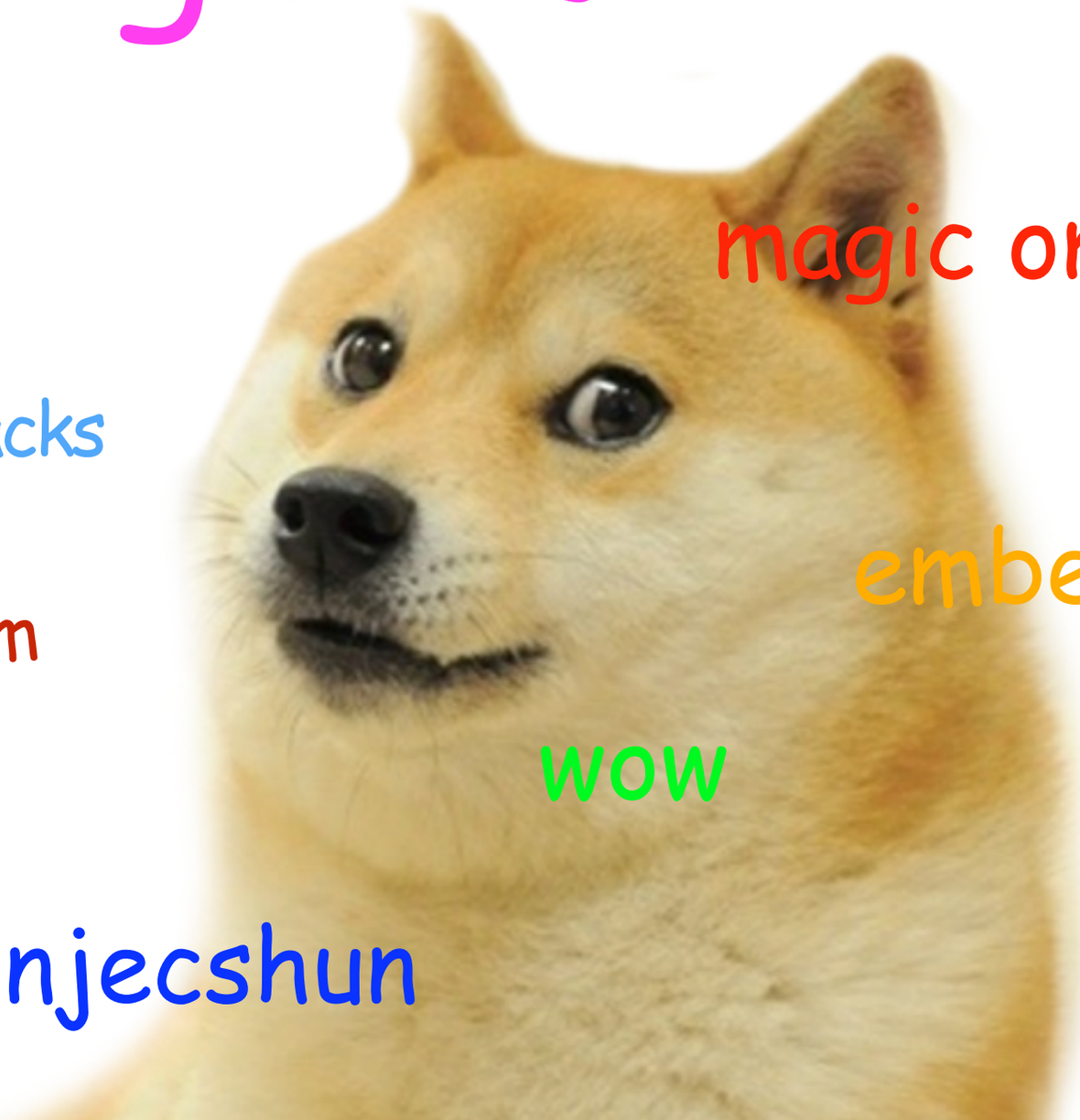
Dependency Injection with AngularJS

Gabe Scholz
T: @daddyflapjacks
G: @garbles
E: gabe@zozi.com



How does Angular determine dependencies?

```
var app = angular.module(“app”, [])

app.controller(‘MyCtrl’, function($scope, $http) {
    $scope.bingo = ‘?????’;
    $http.get(…);
});
```

How does Angular determine dependencies?

```
var app = angular.module(“app”, [])

app.controller(‘MyCtrl’, function($http, $scope) {
    $scope.bingo = ‘?????’;
    $http.get(…);
});
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

How does Angular determine dependencies?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

```
app.controller(‘MyCtrl’, function($scope, $http) {
    $scope.bingo = ‘?????’;
    $http.get(…);
});
```

JAVASCRIPT MINIFICATION FAIL :(:(:(:(

```
app.controller("MyCtrl",function(e,t){e.bingo="?????";t.get()})
```

A way around this:

pre-populate **$inject**?!

Define $inject outside of the annotation function?

```
myCtrlFunction = function($scope, $http) {
   $scope.bingo = ‘?????’;
   $http.get(…);
}
```

```
myCtrlFunction.$inject = [‘$scope’,’$http’];

app.controller(‘MyCtrl’, myCtrlFunction);
```

Define $inject outside of the annotation function?

```
myCtrlFunction = function(e, t) {
    e.bingo = ‘?????’;
    t.get(…);
}
```

```
myCtrlFunction.$inject = [‘$scope’,’$http’];

app.controller(‘MyCtrl’, myCtrlFunction);
```

Define our function as an Array?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

Define our function as an Array?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

Define our function as an Array?

```
function annotate(fn) {
  …

  if (typeof fn == 'function') {
    if (!($inject = fn.$inject)) {
      $inject = [];
      if (fn.length) {
        fnText = fn.toString().replace(STRIP_COMMENTS, '');
        argDecl = fnText.match(FN_ARGS);
        forEach(argDecl[1].split(FN_ARG_SPLIT), function(arg){
          arg.replace(FN_ARG, function(all, underscore, name){
            $inject.push(name);
          });
        });
      }
      fn.$inject = $inject;
    }
  } else if (isArray(fn)) {
    last = fn.length - 1;
    assertArgFn(fn[last], 'fn');
    $inject = fn.slice(0, last);
  } else {
    assertArgFn(fn, 'fn', true);
  }
  return $inject;
}
```

Just make it an array

```
app.controller('MyCtrl', ['$scope', function($scope) {
        $scope.bingo = '?????';
})];
```

What's going on inside app.controller(…)?!

Providers and Services

(A **service**) is a singleton object created by a **service factory**.

These **service factories** are … created by a **service provider**.

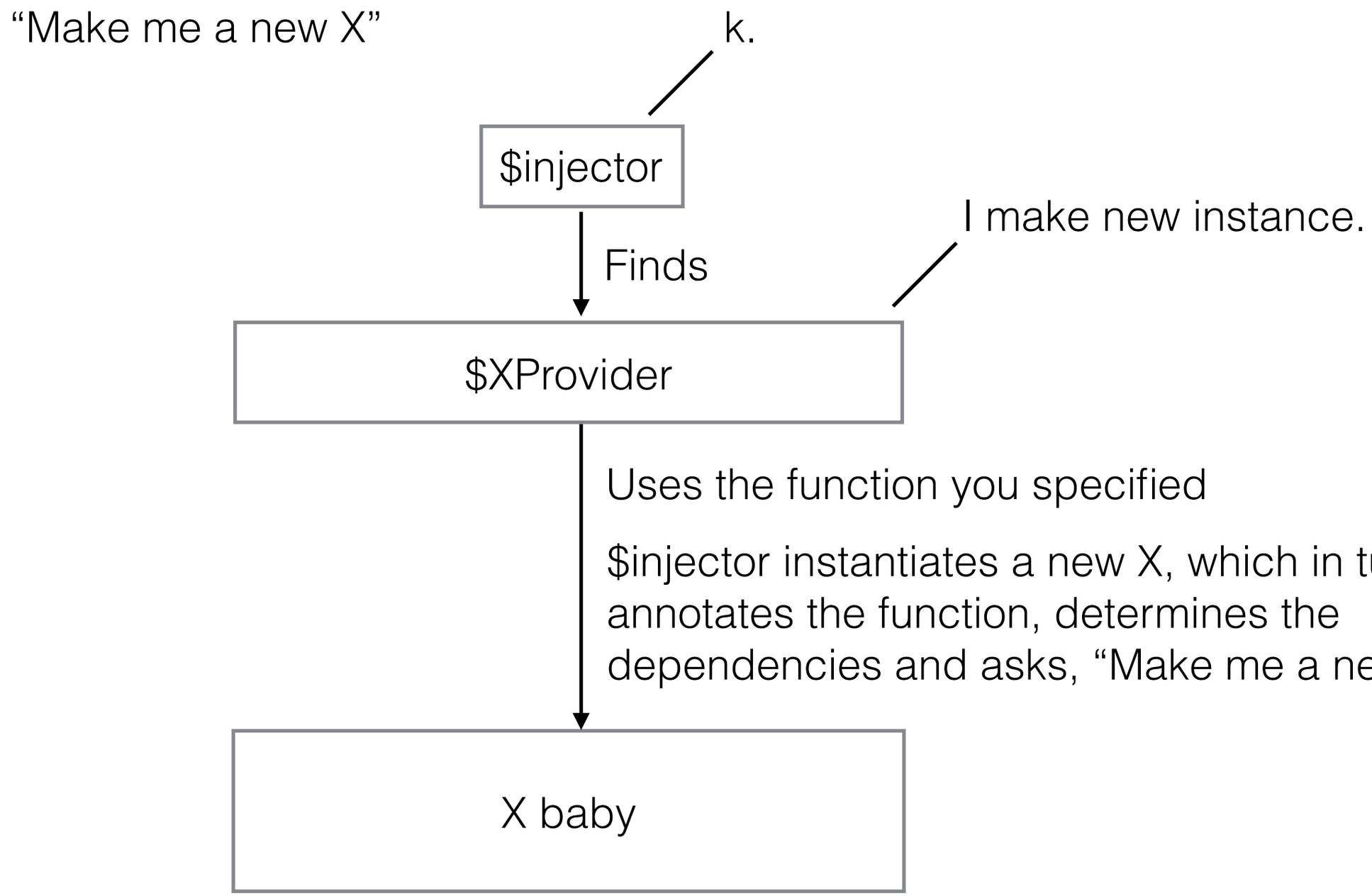
The **service providers** are **constructor functions**.

When you request a **service**, the **$injector** will find the correct **service provider**, instantiate it and then **get the instance of the service**.

Providers and Services

app.service(...) and **app.factory(...)** are simplifications of the **app.provider(...)** which registers are new provider

Providers and Services with a picture!



All of the standard injectable dependencies have providers for instantiating new versions of themselves except for **$scope**, **$element**, **$attrs** which are part of the directive logic because…

The function you write in app.controller(…) is the controller function of the **ngController** directive.

Dat Summary

- Angular turns your function into a string
- It sets an $inject property on the function which is then, in turn, used to fetch dependencies from a cache
- You can also create service X with an array instead
- When you create new controllers/filters/services/ etc. you're actually creating service providers

QUESTIONS?
(Make sure you ask the correct ones)