モダンAngularJS

Dec 6, 2014 第26回 GDG中国勉強会

自己紹介

- 奥野 賢太郎 - Okuno Kentaro
- 京都市から来ました

週間ストック数TOP20にて

"AngularJS アンチパターン集"が

2位をいただきました

http://qiita.com/armorik83/items/b00818ecaf2e93734b36

AngularJS

ご存知ですよね?

普段（業務、それ以外含む）

AngularJSを書いているという方?

そうでしょう

おさらいAngularJSとは

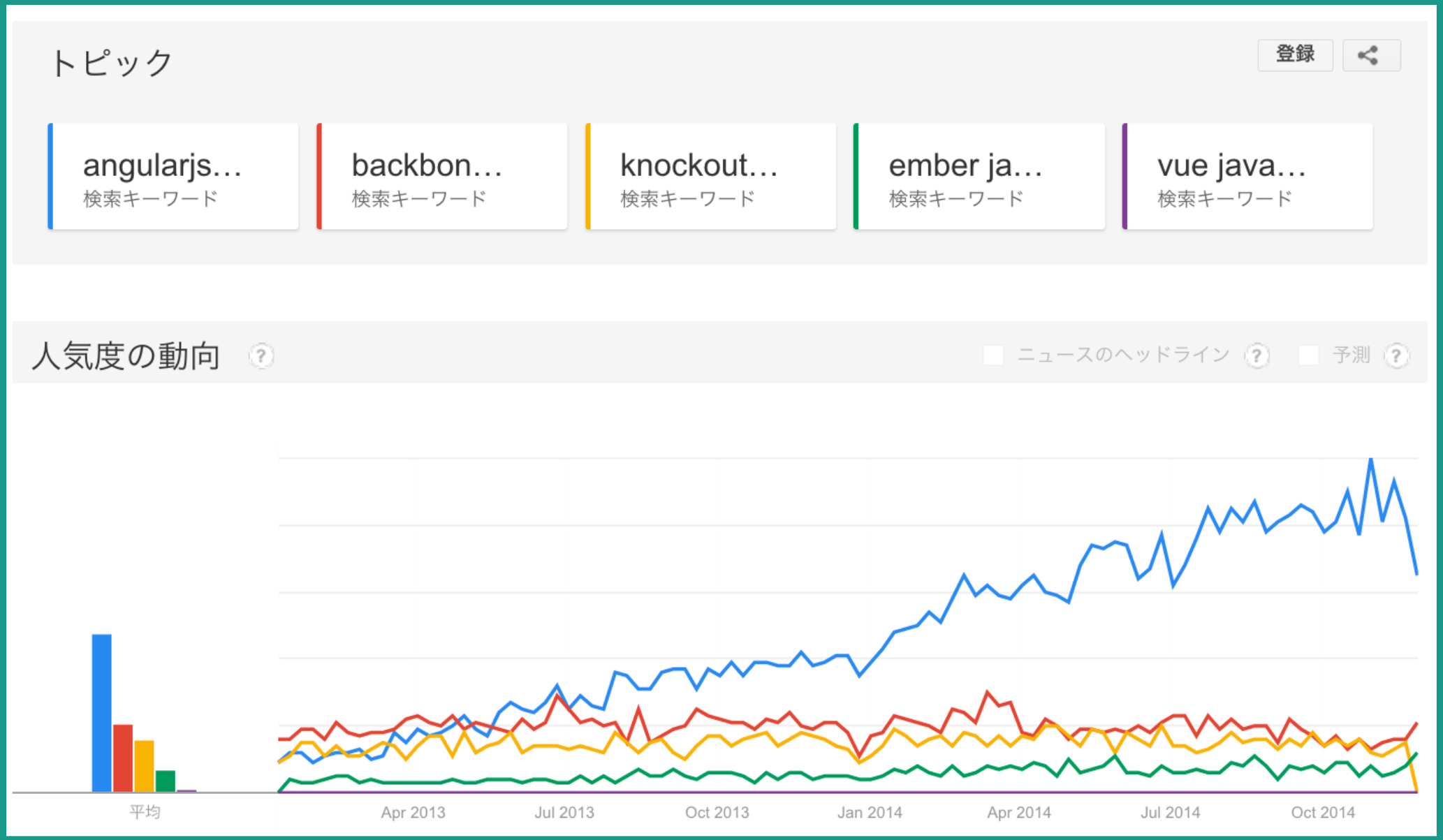
- JavaScript**フレームワーク**
  Googleが開発（公式にはMVW = Whateverとしている）

- **HTMLベース**のテンプレート

- 書きやすい**データ・バインディング**

- **ルーティング機構**
  SPAに最適

- **Ajax**サポート
  REST APIのサーバアプリケーションと相性が良い



圧倒的

※2014/11 angularjs, backbone, knockout, ember, vue に javascriptを付与して検索

「でもAngularJSって**嫌い**」

えっ嫌われてるの?
じゃあ止めておこうか…

「でもAngularJSって**嫌い**」

ちょっと(えっ)待(じょあ)って!

APIがひどい

覚えること多すぎ

導入コスト高い

そんなことはない

現況を知ろう

- とはいえ、手放しでベタ褒めできるわけではない
- でもそれは**どんなフレームワークも一緒**
- **AngularJSの現況**を理解し、的確に利用する
- Angularチームの目指す方向性を知り
  後々**面倒にならない**ためには...?

アウトライン

1. AngularJS 1.x系から2.0系へ
2. 何に向いているのか
3. 覚えたほうがよいもの、覚えなくてよいもの
4. 罠
5. 開発環境の例

アウトライン

1. **AngularJS 1.x系から2.0系へ**
2. 何に向いているのか
3. 覚えたほうがよいもの、覚えなくてよいもの
4. 罠
5. 開発環境の例

ご存知ですか

Angular 2.0が開発中です

とはいえ、リリース時期は未定

おおよその時期すら明言されていません

2.0系の特長

- 2014年10月22-23日に開催された**ng-europe**にて
  ビジョンが示された

- モダンブラウザを意識
  ES 6 (ECMAScript 6) で構築、ES 5でも利用可能

- **AtScript**という拡張言語
  型付け、アノテーション、大規模化への支援

- **jqLiteの廃止**、パフォーマンス向上
  jQuery使用不可となるわけではない

※2014年11月現在の情報

衝撃的な情報も

- **Controllerが無くなる**
  DirectiveとHTMLテンプレートが1対1となる

- **$scopeが無くなる**
  責任分担、情報の参照、共有の仕組みをより明確にする

※2014年11月現在の情報

ガラッと変わるとか怖い

じゃあ止めておこうか…

ガラッと変わるとか怖い

ちょっと待って

2.0のリリース日は未定です

- せっかく**AngluarJS 1.3 stable**があるのに、リリース日未定のAngular 2.0の変化を恐れて止めますか

- ~~何使うんですか~~

- Angular 2.0のビジョンは、開発チームが「よりAngularらしく」を目指した結果

- その意図を組んで現在のAngularJS 1.3を書けば**将来大きな混乱には至らない**のでは

Angularチームが
2.0でガラリと変えようとしている理由は?

Angular 2.0が目指すもの

- モダン・フレームワークへ
  - **ES 6** (ECMAScript 6)での実装
  - **AtScript**による大規模プロダクトの記述支援
- より整理されたAPI
  - 覚えること多すぎ! ってdisられるの、自覚してるらしい

ES 6って何

- **ECMAScript**（エクマ・スクリプト）とはJavaScriptの標準規格
  我々がJavaScriptと呼ぶ言語はブラウザ毎に方言だらけ

- 現在のモダンブラウザ (>=IE 9) はES 5に対応
  ES 6を全て実装するブラウザは**現在無し**

- **class**糖衣構文、**module**機構、**Promise**の標準化などなど

- 詳細はググってね
  `http://kangax.github.io/compat-table/es6/` オススメ

AtScriptって何

- その前に**TypeScript**（Microsoft社）
  これは**静的型付け**構文を備え、コンパイラによって
  変換時にエラーをチェックし**JavaScriptを生成**する言語

- このTypeScriptを**さらに拡張**した
  Angularチームが提案する**構文**（曰く新言語ではない）

- AtScriptでは型付けに加えて、
  メタデータ記述、アノテーションの構文を**補強**

```
@Directive({
  selector: '[blink]'
})
class Blink {
  constructor(elment: Element, options: Options, timeout: Timeout) {
    // ...
  }
}
```

AngularJS + TypeScript

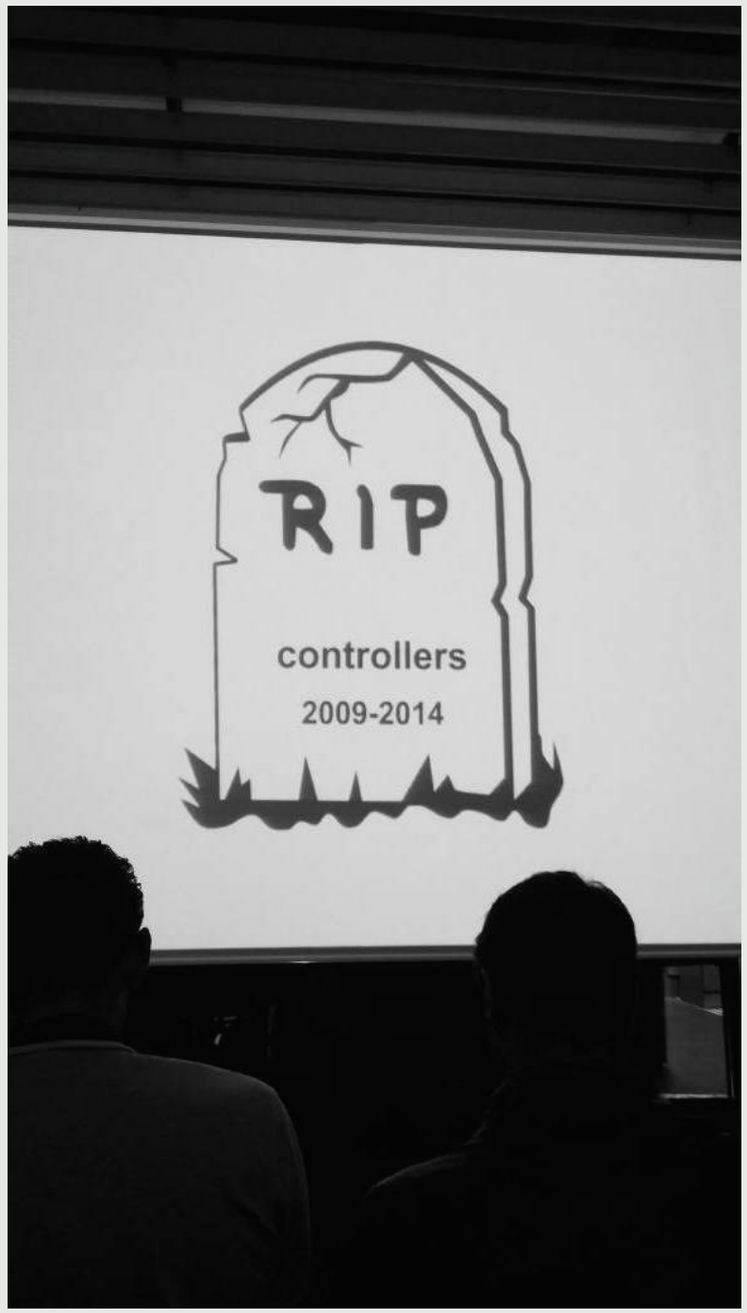
- TypeScriptは**ES 6**の構文を**先行導入**している
  - class, moduleなど
  - もちろん変換するので、ES 5のブラウザで動く
- **静的型付け**とコンパイラによるエラーチェック
  - 大規模開発ほどコンパイラの支援が重要に
- Angular 2.0で提案される**AtScript**に今から慣れるならこれ
- TypeScript自体、全てのJavaScriptに活かせるので
  覚えておいて損はない

モダンAngularJS 1

今からES 6文法が使え
Angularチームも基礎として採用した
TypeScriptで書く

Controller無くなっちゃうん?

- 無くなるようです
- Angular 2.0のリリースは**未定なのに**
  2014年にお亡くなりとされた
- つまり1.3の時点でも廃止を
  見越したほうがよい



ng-controllerって

- 今なお多くのチュートリアルやブログで紹介されている "ng-controller"

- あれは**HTML**の**Attributes**ではなく
  単なるDirectiveの一種である

- **これって必須なのか?**

```
<div ng-controller="MainCtrl">...</div>
```

2.0を見越したController

- **Directive**にも**controller API**があります
- Directiveを作って、**1 Directive - 1 Controller**とする

- **ng-controller**だと`<div>`に一個指定して
  後はズラズラ処理を書きがち

- このルールにすることで個々のDirectiveを高凝集へ
  MVVMの考え方

- **ngRoute**ルーティングでもcontroller指定はしない
  → **Viewだけを指定**し、そのViewにDirectiveだけ置く

モダンAngularJS 2

ng-controllerを使わず

Directive controllerを用いる

$scope無くなっちゃうん?

- 無くなるようです
- 現在においても**controllerAs構文**が認知されてから出番が激減
- $scopeのAPI（watchやbroadcast, on）は今でも使う

2.0を見越した$scope

- 現在実装中または稼働中のシステムにng-controllerがあって**$scope**も使ってますという方
  - 移行が大変そうなら、まずは**controllerAs構文**の採用へ

```
<div ng-controller="MainCtrl as main">...</div>
```

- 親子間で混乱するほど**$scope**の情報をやりとりしてたら、そりゃ設計の問題

$broadcast, onを使う

- 個々のDirectiveはとにかく**低結合**で作る
- 何かのイベントのトリガーに`$broadcast`を使い
  駆動側は`$on`で受ける
- 発信側は、他のコンポーネントがどう動くか
  受信側は、どのタイミングでイベントが発せられるか
  お互い知らなくていい（低結合）

ただし最善ではない

- `$broadcast, $on`は最善のアプローチではない
- 文字列リテラルでやりとりするからである

```
$scope.$broadcast('EventNameString', args);
```

- IDEの支援も受けにくい
- 解決法としては、**`$broadcast`**自体を
  すべてserviceでラッピングし、**メソッドとして**扱う
  `$on`に書くイベント名も、そのserviceから取得する
  メソッド名ならIDEが認識して補完してくれる

ただし最善ではない

- `$broadcast`のserviceラッピングは今のところ上手くいっている
- しかしイベント駆動の全てを**一つのserviceで**ラップすると、それはそれで問題
  自分は分野別に適宜小分けにしています
- **`$scope`のプロパティを無闇に親子で共有し読み書きするよりは確実**
  本音を言うと2.0でもっといい機構求む

モダンAngularJS 3

親子で$scopeの共有は危険

低結合に作り

イベント駆動を意識

1.3も改良されます

- AngularJS 1.x系も、1.3がリリースされて終わりではない
- すでに1.3.4のマイナーアップデートがリリース
- 1.3の直系である**1.4**も進行中
  - ルーティング、フォーム、Ajaxの改良などが検討されている模様

    `http://www.linkplugapp.com/a/1042855`

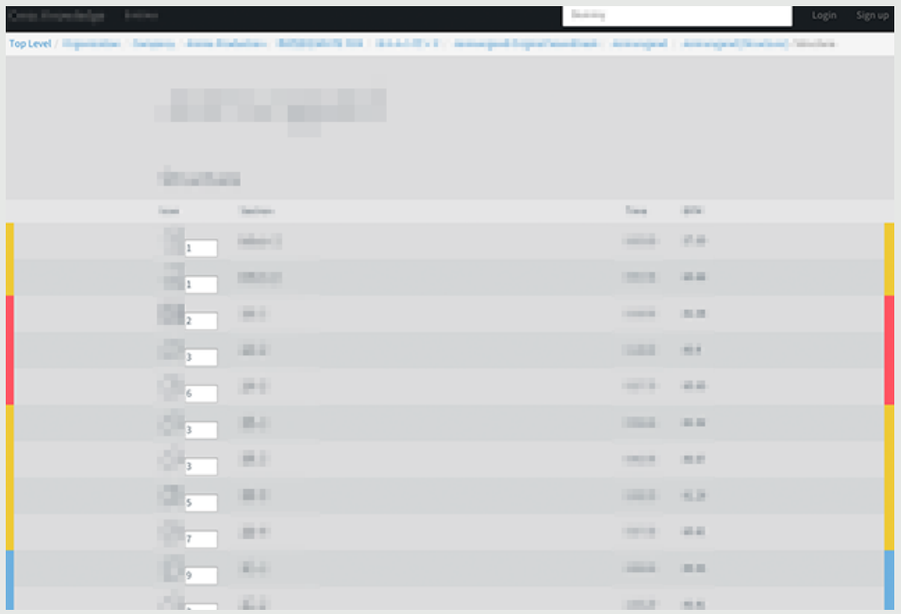
※2014年11月現在の情報

アウトライン

1. AngularJS 1.x系から2.0系へ
2. **何に向いているのか**
3. 覚えたほうがよいもの、覚えなくてよいもの
4. 罠
5. 開発環境の例

私自身のケース

- 自分がどのように**AngularJS**を使っているか
  - 業務システム + 業務ツール、アクセサリの実装
  - 将来的にこの技術の一部を一般公開することを目論んだ基盤作り



こんな感じで作ってます

個人的な感想ではありますが

- 開発は速いです
- **データ・バインディング**周りが整っているので
  1ケースのプロトタイプがすぐ出来る
  すぐ動くのでモチベーションは維持しやすい
- **REST APIと親和性が高い**ので既存のシステムから
  移行しやすい（はず）
  私はCakePHPからnode.jsに移行した

SPAでも、一箇所だけでも

- Directiveなんて使わない、*.html内に直接置いて
  フォームのバリデーションとしてだけ使いたい
  →いいと思います

- ルーティングも全てAngularJS
  フルに機能を使って**Single Page Application**を開発するぞ
  →すばらしい

- AngularJSの得意とする**データ・バインディング**、
  RESTful、**ルーティング**といった分野から何が使えるか
  見極める

テストしやすい

- SPA開発では**Directive**も**Service**もどんどん増えます
- Angularチームはテストのしやすさに
  重点を置いている
  - DIベースのモジュール連携と、モック注入
  - テストランナー **"Karma"**
  - E2Eテスト **"Protractor"**

テスト書いてないとかお前それ…

モダンAngularJS 4

Karma, Protractorで
テストも抜かりなく

アウトライン

1. AngularJS 1.x系から2.0系へ
2. 何に向いているのか
3. **覚えたほうがよいもの、覚えなくてよいもの**
4. 罠
5. 開発環境の例

覚えることが多すぎる?

そうでもない!

まずDirective覚えてください!

Directiveによくある誤解

Directiveによくある誤解

- 難しい（らしい）

Directiveによくある誤解

- 難しい（らしい）
- 面倒くさい（らしい）

Directiveによくある誤解

- 難しい（らしい）
- 面倒くさい（らしい）
- APIが複雑すぎる（らしい）

Directiveによくある誤解

- ~~難しい（らしい）~~
- ~~面倒くさい（らしい）~~
- ~~APIが複雑すぎる（らしい）~~

API全部なんて使わない!

経験談からのDDO

- **DDO** = Directive Definition Object
  あの長い長いオプション、Directiveは面倒と思われる一番の原因

- **restrict** - Directiveの表記方法、属性名か、要素名か…
  - これは**"A"**（属性名として）と**"E"**（要素名として）だけ覚えておく（あとは使わない）

- **scope** - Scopeを生成するか共有するか
  - 常に**オブジェクト・リテラル**で指定
    A new "isolate" scope.
  - 共有前提の設計は避けたほうが良い

```
scope: {
  attr: '@myAttr',
  foo:  '=myFoo',
  bar:  '@myBar'
},
```

使わないAPI

- **replace** - deprecated

- **transclude** - 挙動が直感的でなく、検証の手間を増やすならば使わなくてもいいのでは（APIを読んでも必要と感じたことがない）

- **multiElement** - 必要になったことがない
  他のモジュールを観察しても見かけたことがない

- **priority** - 優先度を気にする必要のある入れ子はリスク
  priorityを使わず**requireを使う**

- **link** - compileの戻り値から関数を返せばよい

※個人の感想です

template

- 常に**templateUrl**を使っている
- あまりJSソース内に文字列として長いHTMLを書きたくない
- ただし、小さなパーツ（1行で済むような）はcompile内に`$templateCache.put()`を使って書くことが多い

モダンAngularJS 5

Angular 2.0に備えるなら

Directiveを積極的に**自作**

【朗報】

DDO (Directive Definition Object) は **Angular 2.0**で**廃止**されます

次にService, Factoryの違いを
覚えておくのがオススメ

ServiceとFactory

- **Service** - コンストラクタ関数を与えてAngularJSがインスタンスを生成
  Singletonとして残る
- **Factory** - 与えた関数を実行し戻り値を返す

```
function MyComponent() {
  // ...
}

angular.module('myModule')
.service('MyComponent', MyComponent);

angular.module('myModule')
.factory('MyComponent', function() {
  return MyComponent;
});
```

ロックイン範囲を少なく

- AngularJSは巨大なフレームワーク
  一度使うとロックインされるから…という意見を聞く

- しかしAngularJSを使っていても
  将来的に1.xから2.0には**移行作業が発生**するはず

- AngularJSと関係ないロジックは
  なるべくAngularJSの外に書くべきじゃね?

ModelとInject

- MVC**すべてをAngularJSで**やろうとするから
  ロックインが気にかかってくる

- AngularJSはただ単に**REST + View Model + Template**
  に特化

- Model、ビジネスロジックはAngularJS APIを一切含まない
  純粋なJSとして実装
  仮にAngularJSがポシャっても他FWで使いまわせるように

ModelとInject

- この純粋なJSのライブラリをAngularJSで用いる場合**Factory経由で**返す

- グローバル変数に置いてAngularJSで使うことも出来るが**モック化できない**のでテストの上で問題が起こる

- **AngularJSのユニットテスト**は振る舞いの検証に留めて、**E2Eテスト**による大きな網と、ビジネスロジックの**細かなユニットテスト**で保つ

```
angular.module('myModule')
.factory('MyBusinessLogic', function() {
  return MyBusinessLogic; // インスタンスではなくコンストラクタ自体を返す
});
```

オブジェクト指向

- 「インスタンスではなくコンストラクタ自体を返す」
  という表現、JavaScriptではちょっと馴染みがないかも

- **TypeScript**ではclass, constructorや型付けの概念がある

- 変換後がJavaScriptでありながらも、JavaやC#的な
  オブジェクト指向の考え方で進めることができる
  JavaScriptだけでも出来るけどprototypeとかいろいろ書くことが面倒

- ServiceもFactoryも、**TypeScript**との相性はとてもよい
  AtScriptが発表されたことも、それを証明している

モダンAngularJS 6

Factoryによるinjectを活用して

AngularJS APIを

使わない処理は外へ

ServiceはAngularJS内に特化

- 一方で**Service**はAngularJS APIの積極的な利用に特化
- `$resource, $routeParams, $location`は経験上コードの重複が生まれやすい
- Core APIをラップした、自分・自チームが使いやすい**Service**を用意して処理を一元化
- **Directive controller**はあくまでも複数のServiceの処理とView, Eventを結びつけるインタフェースに徹する

モダンAngularJS 7

Core APIを使いやすくラップ

自作Serviceで

処理の重複を避ける

全部覚える必要はない

アウトライン

1. AngularJS 1.x系から2.0系へ
2. 何に向いているのか
3. 覚えたほうがよいもの、覚えなくてよいもの
4. **罠**
5. 開発環境の例

慣れると怖くないですが
ちょいちょい罠があります

1. minify対策

- **DIアノテーション**は**必ず**書きましょう
  さもなくばminify時に泣きます
- DIアノテーションとは、インスタンス生成時に
  インジェクトするServiceを決定するための情報
- 引数を文字列でパースして読んでるらしいぜ…
  と変態実装が話題になりましたが、大変よろしくないです

```
angular.module('myModule')
.service('MyService', function(OtherService) {
  // ...                ↑ここに無い!!
});

angular.module('myModule')
.service('MyService', ['OtherService', function(OtherService) {
  // ...
}]);
```

1. minify対策

- 個人的にはこっちの書き方のほうが好き

```
function MyService(OtherService) {
  // ...
}
MyService.$inject = ['OtherService'];
```

```
angular.module('myModule')
.service('MyService', MyService);
```

- .service()内に長々と書くより**ネストも減ってスッキリ**
- AngularJS 1.3から **ng-strict-di** が追加された
  アノテーション記述漏れをエラーとして指摘する

```
<body ng-app="myApp" ng-strict-di>
  ...
</body>
```

2. Filter

- ループ処理は罠となりやすい

- 1.2から1.3になり速度が改善されたが
  あまり多用すると影響の出る恐れがある

- **AngularJS Batarang**というChrome機能拡張を使って
  パフォーマンスが気になったら計測しよう

- JSON生成時にサーバ側で変換するほうが
  明らかに速いので、こちらも要検討

3. $routeProvider

- 非同期処理の解決を待ってからルーティング処理を行うAPI
  "resolve"には注意

- 処理が`$routeProvider`とControllerで分離し、あとあと保守上で
  リスクとなる可能性を含む

- チュートリアル通りにresolveを書いてしまうと
  テストしにくい記述へ

- Directive controllerの考え方でいくなら問題となりにくい

- **AngularUI** RouterというOSSもあるので検討してもよいかも

4. AngularUI

- ただし、そのAngularUI
  めちゃ便利なサードパーティ・ライブラリだが
  AngularJSのリリース・スピードに対して微妙に更新が遅い
- どんなものにも言えるが、ライブラリのコミュニティの
  活性度や**更新頻度**をみながらの採用を
- 実際、AngularUIに足を掴まれてしばらく1.2のままでした
- あっ、でもUI Routerは優秀です

5. directive処理順

- `ng-repeat`やDirectiveの`compile, link`を多用していると高確率で混乱する

- jQueryも併用してリッチなUIを表現しようとか思い始めると、期待する要素が`undefined`だったとかザラ

- 勘をつかむまでは検証が大変

- 先日（12月2日）公開した**Qiita Advent Calendar**の拙記事にて、このややこしい処理順を全てまとめています
ぜひご参考に

モダンAngularJS 8

ハマりどころは先人がハマってる

罠を恐れず**進め**

アウトライン

1. AngularJS 1.x系から2.0系へ
2. 何に向いているのか
3. 覚えたほうがよいもの、覚えなくてよいもの
4. 罠
5. **開発環境の例**

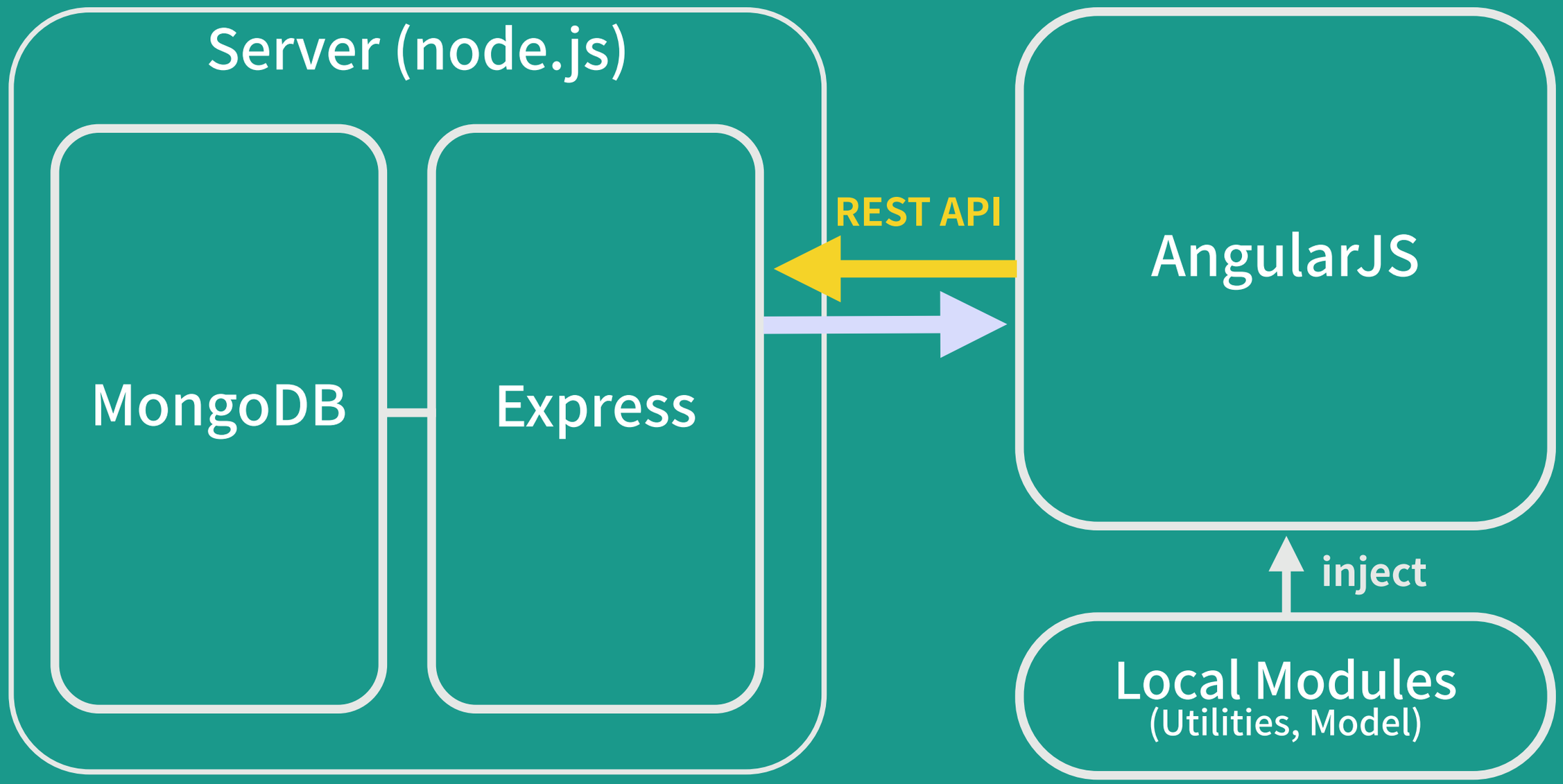
最後に私個人の

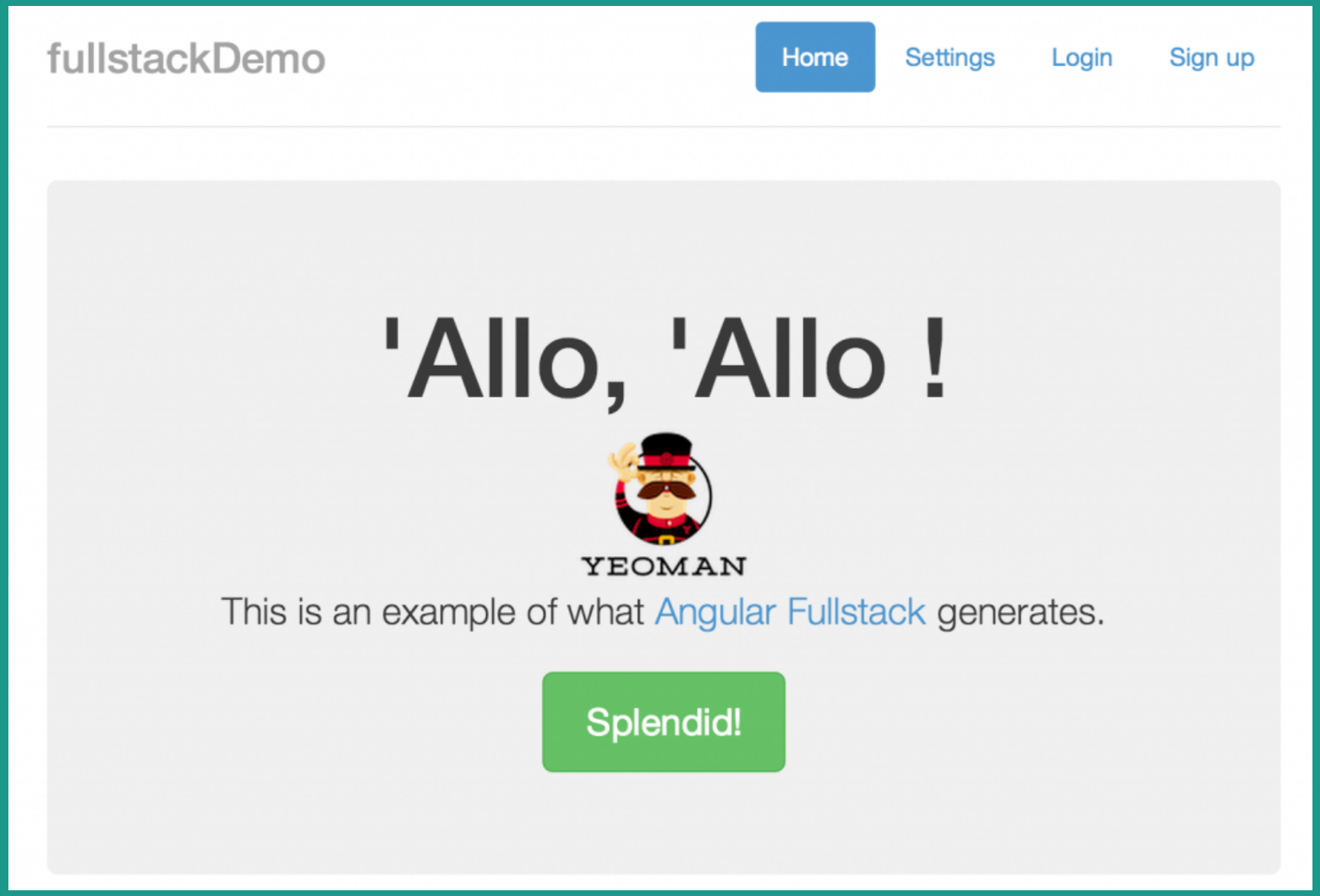
開発環境の例を紹介して

締めくくります

（たぶん駆け足）

構成


Server (node.js)
MongoDB
Express
REST API
AngularJS
inject
Local Modules
(Utilities, Model)



初回は
generator-angular-fullstack

現在のディレクトリ

```
root
├── Gruntfile.js
├── app
│   ├── bower_components
│   ├── my_components
│   │   └── *
│   ├── my_modules
│   │   └── *
│   ├── favicon.ico
│   ├── fonts
│   ├── images
│   ├── robots.txt
│   ├── scripts
│   │   ├── app.js
│   │   ├── controllers
│   │   ├── directives
│   │   ├── filters
│   │   ├── injectors
│   │   ├── modules
│   │   ├── routes.js
│   │   ├── run.js
│   │   └── services
│   ├── styles
│   │   ├── sass
│   │   └── screen.css
│   └── views
│       ├── 404.html
│       ├── index.html
│       ├── partials
│       └── parts
├── bower.json
├── common
│   └── *
├── doc
├── e2e.conf.js
├── karma.conf.js
├── lib
│   ├── bootstrap-models.js
│   ├── config
│   │   └── *
│   ├── controllers
│   │   └── *
│   ├── middleware.js
│   ├── models
│   │   └── *
│   └── routes.js
├── log
├── node_modules
├── package.json
├── server.js
├── test
│   ├── client
│   │   ├── e2e
│   │   ├── mock
│   │   └── spec
│   └── server
│       ├── mock
│       └── spec
├── tsd.json
└── typings
    ├── e2e.d.ts
    └── tsd.d.ts
```

字が小さくてすいません

開発の最初にやったこと

- 何よりも**ログ出力**まわりを揃えた

- 成功・失敗を含めて**永続的に残すログの出力**と、
  エラーのたびにターミナルやブラウザ・コンソールに
  表示させるログ周りの出力と整形

- node.jsの**Express**と**AngularJS**におけるAPIの通信は
  特に細かく記録

- MongoDBを扱うライブラリ**Mongoose**がそこそこログを
  吐くので、データベース・コネクションまわりで途方に
  暮れることは少なかった

コーディング・スタイル

- `angular.module()`の書き方だとか、`inject`の書き方だとか
  みんなけっこうバラバラ

- 最初に見たチュートリアルのスタイルでそのまま進めてしまうケース
  これはとても危ない!
  チュートリアルは長期設計を意識していない
  - ググったら数種類の**コーディング・スタイル**が出てくる
    angularjs style guide [検索]
  - それらにも差はあるが、共通して述べている事項もあるので
    その辺従うべき

- 個人的には `function(){funtion(){function(){...` が
  堪えられないので、可能な限り平らに書いている
  その方が非同期絡みのテストや差し替えもしやすい

npm, bower

- **yo angular-fullstack**が用意したpackage.jsをベースにしている
- コミュニティの流れや廃れていないかを見ながら日々試行錯誤
- **node_modules**は

  express, lodash, log4js, mongoose, mongoose-auto-increment, passport など

- **bower_components**は

  angular, angular-animate, angular-ui-bootstrap, d3, es5-shim, jquery, lodash, node-uuid, underscore.string など

grunt

- ビルドツールにはgruntを使用
- Expressサーバの起動やテスト、コンパイルなど、すべてgruntに任せている
- `connect-livereload`, `grunt-este-watch`, `grunt-express-server`, `grunt-karma`, `grunt-ng-annotate`, `grunt-ts`, `grunt-typedoc`, `karma`, `karma-phantomjs-launcher`, `load-grunt-tasks`, `protractor`, `time-grunt` など
- 一個一個説明できませんが、injectのアノテーション記述を自動化する `grunt-ng-annotate` はおすすめ

テスト

- node.js側はそのまま**Mocha**
  AngularJS側は**Karma + PhantomJS + Jasmine**

- AngularJS公式ではテスト・フレームワークに
  Jasmineを勧めているが**Mocha**が動かないわけではない

- E2Eテスト "Protractor"に関しては
  Jasmineに（事実上）限定される

- アサーションは**power-assert**
  モックテストは**Sinon.JS**、**Jasmine Spy**を併用
  今後**Sinon.JS**側に一本化しようと思っている

大規模化の悩みと解決

- 1年でけっこう膨らんでしまった
  現在1万行ほど（空行、コメントを除く）

- テストの実行が遅かったり、ひとつのDirectiveの検証が
  面倒になってきたり、デメリットが目立ち始めた

- 積極的に**外部モジュール化**、Gitのリポジトリも分け
  変更を追いやすくし、テストの責任範囲も限定した

- 面倒な外部モジュール化には**Yeoman Generator**を自作し半自動化

- 結果、改良すべき点、新機能の実装、バグ発生箇所が
  すぐ分かるようになり**イイことばっかりです!!**
  最初からそうすべきだった

モダンAngularJS 9

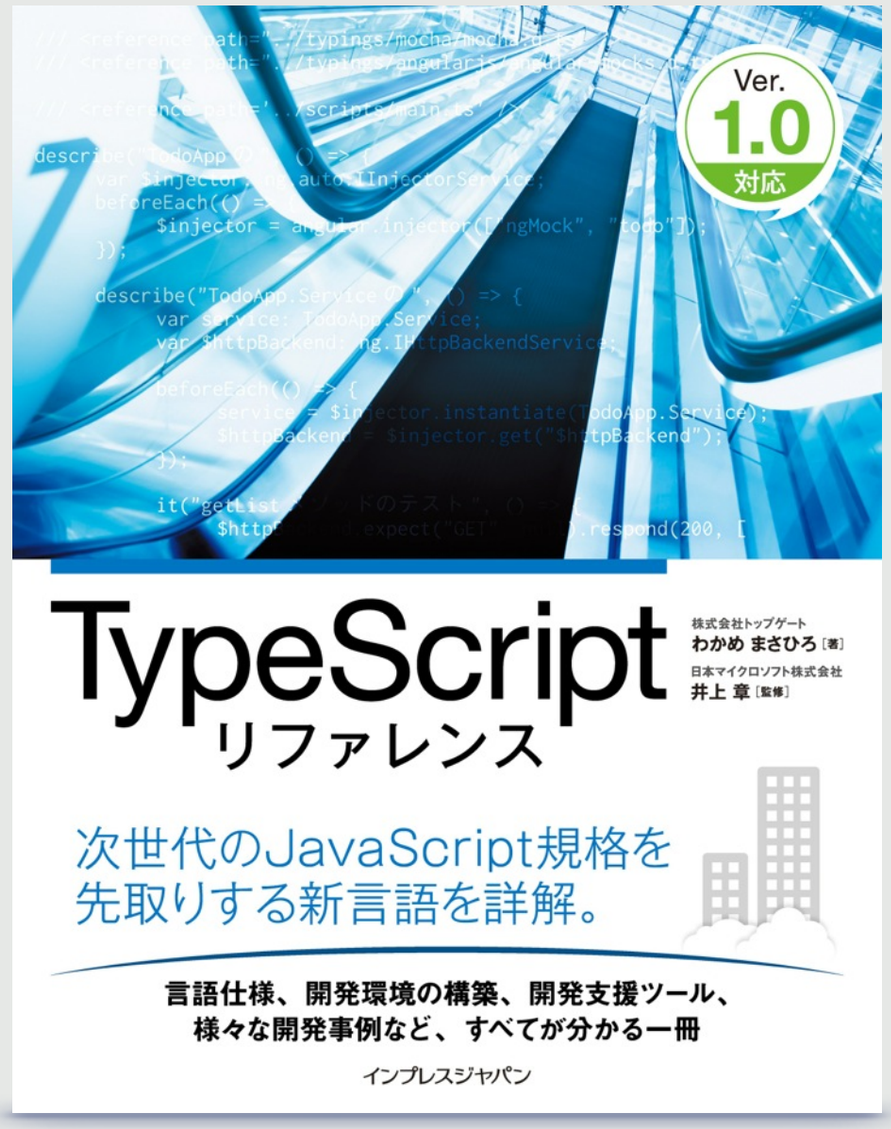
パッケージ管理、ビルドツールで

コンパクトな実装に専念

参考資料・記事（順不同）

- ng-europe - http://ngeurope.org/
- AngularJS カンファレンス（ng-europe 2014）のスライドまとめ
  - http://angularjsninja.com/blog/2014/10/24/slides-at-ngeurope-2014/
- （日本語訳）ng-europe, Angular 1.3, and Beyond
  - http://angularjsninja.com/blog/2014/10/28/ngeurope-angular1.3-and-beyond-in-japanese/
- 世界のJavaScriptを読もう - http://azu.github.io/slide/jser200/javascript-2014.html
- Qiita | Angular 2.0 メモ - http://qiita.com/shuhei/items/bd3376c4a916af559739
- Brace yourselves, future is coming: ES6, AtScript and Angular 2.0
  - http://blog.lingohub.com/2014/11/brace-future-coming-es6-atscript-angular-2-0/
- AngularJSが嫌い - http://mizchi.hatenablog.com/entry/2014/10/06/162103
- AngularJS 嫌いな人が多い昨今について - http://blog.64p.org/entry/2014/11/21/104739
- AngularJS についての所感 - http://havelog.ayumusato.com/develop/javascript/e628-angularjs_thought.html
- なぜAngularJSを薦めるのか ー 個人的な思い - http://blog.mitsuruog.info/2014/11/angularjs.html
- DaftMonk/generator-angular-fullstack - https://github.com/DaftMonk/generator-angular-fullstack
- LIG主催のAngularJS勉強会 #ngCurryが開催されました - http://liginc.co.jp/web/js/other-js/131428
- Todd Motto - http://toddmotto.com/
- twada/power-assert - https://github.com/twada/power-assert

謝辞



TypeScriptリファレンス
わかめ まさひろ 著

AngularJSリファレンス
池添 明宏、金井 健一、吉田 徹生 共著

本講演に至るまで、さまざまなご指導を頂いたわかめ氏、池添氏に
御礼申し上げたく謝辞にかえさせていただきます

今日の復習

モダンAngularJS 1

今からES 6文法が使え
Angularチームも基礎として採用した
TypeScriptで書く

モダンAngularJS 2

ng-controllerを使わず
Directive controllerを用いる

モダンAngularJS 3

親子で$scopeの共有は危険
低結合に作り
イベント駆動を意識

モダンAngularJS 4

Karma, Protractorで
テストも抜かりなく

モダンAngularJS 5

Angular 2.0に備えるなら
Directiveを積極的に**自作**

モダンAngularJS 6

Factoryによるinjectを活用して
AngularJS APIを
使わない処理は外へ

モダンAngularJS 7

Core APIを使いやすくラップ
自作Serviceで
処理の重複を避ける

モダンAngularJS 8

ハマりどころは先人がハマってる
罠を恐れず**進め**

モダンAngularJS 9

パッケージ管理、ビルドツールで
コンパクトな実装に専念

モダンAngularJS 10

それでは!

Have a good モダン AngularJS!

ご清聴ありがとうございました

Dec 6, 2014 第26回 GDG中国勉強会