JavaScript 研修

2015/04/15(水) @株式会社ガイアックス

ほと（@hoto17296）

- 株式会社ガイアックス
- 新卒２年目
- Web関係ならなんでもやるマン

この研修の目的

JavaScript とその周辺技術
最近のトレンドなどについて
概要を押さえる

特に押さえて欲しいポイント

- JavaScript流オブジェクト指向
- イベント駆動
- DOM
- AjaX

今日の流れ

1. JavaScript 概論
2. クライアントサイド JavaScript

ワーク ゲームを作ろう

3. サーバーサイド JavaScript

ワーク ChatOps してみよう

1. JavaScript 概論

JavaScript とは

- ブラウザで動作するプログラミング言語
- 様々な場所で使われている
- Java とは似て非なるもの

ここ数年のJSブーム

- 遅かったJSがサクサク動くようになった
- Flash から HTML5 に置き換わる流れ
- CommonJS の登場

JavaScript の種類

	クライアントサイド JavaScript	サーバーサイド JavaScript
実行環境	ブラウザ	node.js
用途	Web	色々
モジュール読み込み	<script>	require

なぜ JavaScript を学ぶのか

- エンジニアにとって避けては通れない道
- 唯一のブラウザ標準言語
- ゲーム系エンジニアでも
  何かしらで触ると思う(よく知らない)

なぜ JavaScript を学ぶのか

- なんだかんだでよくできている言語
- 極めてシンプルなオブジェクト指向
- 非同期処理・イベント駆動

JavaScript の文法

JavaScript のキホン

- リテラル
- オブジェクトリテラル
- 変数
- 式と演算子
- 関数
- 構文
    - 条件文
    - ループ文

リテラル

リテラル	例
数値	123
文字列	"hogehoge"
論理値	true / false
配列	[1, 1, 2, 3, 5, 8]
NULL	null
undefined	undefined
オブジェクト	{ hoge: 123, fuga: 'foo' }
関数	function hoge(param1) { ... }

オブジェクトリテラル

```
// 定義
var obj = {
  hoge: 'ほげほげ',
  fuga: 1234,
  foo: { a: 1, b: 2 },
  bar: function(a, b){
    return a + b;
  },
};
```

```
// 参照
console.log( obj.hoge );
console.log( obj.fuga );
console.log( obj.foo.a );
console.log( obj.bar(1, 2) );
```

- RubyやPerlでいうところのハッシュ
- 配列や関数も入れることができる

変数

- **var文** を使って新しい変数を作成
- 独特のスコープの概念があるので注意
  - スコープチェーンという
  - とりあえず「中から外が見える」でOK

```
var hoge;
var fuga = 123;
```

演算子

演算子名	例
代入演算子	= += -= *= /= %= $= ^= |= など
比較演算子	== != === !== > >= < <=
算術演算子	+ - * / % ++ — -
ビット演算子	& | ^ ~ << >> >>>
論理演算子	&& || !
文字列演算子	+ +=
特殊演算子	delete in new this typeof など

関数

```
// 関数定義
function hoge(param1, param2){
  var result = param1 + param2;
  return result;
}

// 関数呼び出し
hoge(1,2);
```

条件文

```
if (hoge < 10){
    ...
} else if (hoge < 20){
    ...
} else {
    ...
}
```

ループ文

```
while (条件式){
    ...
}

for (初期化式; 条件式; 変化式){
    ...
}
```

JavaScript の
オブジェクト指向

クラスベースではない

- オブジェクト指向にクラスは必須ではない
- オブジェクトリテラルがすべて
  - {} ← これ
  - 極めてシンプル

関数は第一級オブジェクト

- 変数に代入したり、
  関数の引数に関数を渡したりできる

- JavaScriptの「メソッド」は
  オブジェクトに生えているただの関数

- メソッドはオブジェクトに束縛されない

- this は呼び出し時に決まる

プロトタイプチェーン

- 「継承(extend)」ではなく「移譲(delegate)」
- 説明しきれないので、詳しくはこのスライドを参照
  - 最強オブジェクト指向言語 JavaScript 再入門！

JavaScript の
オブジェクト指向

- (わかりづらすぎるので実演)

非同期処理とイベント駆動

イベント駆動

- 「◯◯が起きたら××を実行する」

- あとでワークやります

非同期処理

- 処理の終了を待たずして次の処理を行う
- 非同期処理の結果に対する処理は
  コールバック関数を用いる

コールバック関数の罠

- コールバック関数を使いすぎると
  - コールバック地獄にハマる
    - コードの可読性が著しく低下する
- Promise パターンを用いるとスッキリする

altJS

altJS

- JavaScriptはシンプルで軽い反面、
機能的に物足りない部分も多い
  - 型チェック / class / リッチな構文 など

altJS

- 物足りなさを解決するために生まれたのが altJS

- JavaScript にコンパイルできる言語のことを
  総称して altJS と呼ぶ

代表的な altJS

- CoffeeScript
  - Ruby / Python っぽい文法
- TypeScript
  - Microsoft製
  - 静的型付け
- JSX
  - DeNA製
  - React.js で一躍有名に
- Dart
  - Google製
  - JSを置き換えようとしたが頓挫した

2. クライアントサイド JavaScript

DOM 操作

予備知識

- HTML と CSS
    - id と class
- DOM

HTML と CSS

- HTML： 要素を記述する言語
    - 何があるか
- CSS： 見た目を記述する言語
    - どんな風に見えるか

id と class

- どちらも要素にラベルをつけるもの
- id はページ内で唯一の要素
- class はページ内に複数あるかもしれない要素

id と class

```
<html>
  <head>
  </head>
  <body>
    <div id="list">
      <div class="item">あ</div>
      <div class="item">い</div>
      <div class="item">う</div>
      <div class="item">え</div>
      <div class="item">お</div>
    </div>
  </body>
</html>
```

id と class

- とりあえず覚えてほしいこと
  - id は 「 **#** 」
  - class は 「 **.** 」
- CSSセレクタをはじめ、このルールを用いて
  ラベル(id/class)を探すケースが多い

DOM

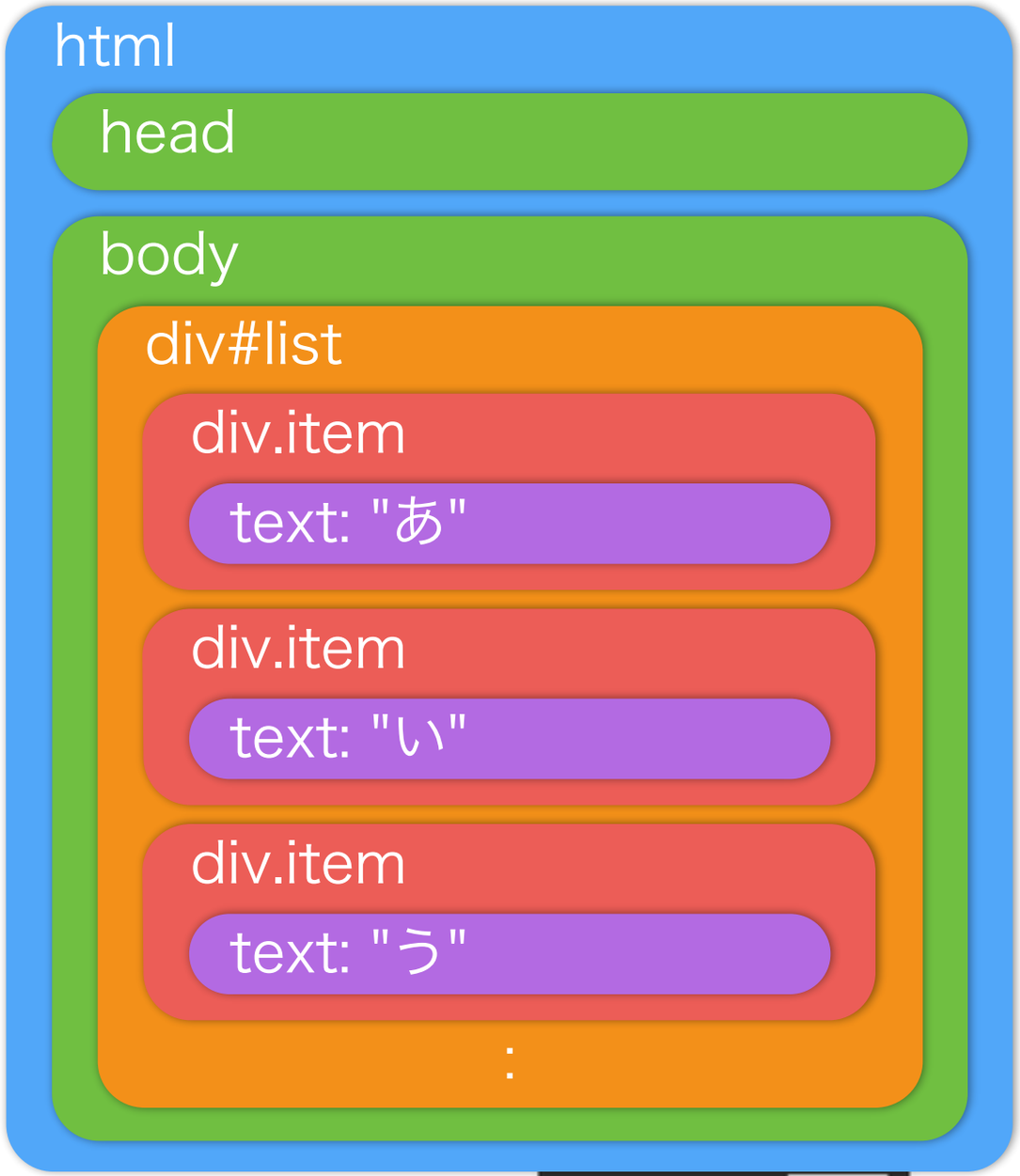
- Document Object Model
- HTML(XML) を木構造として扱い、
  構造を取得したり操作したりする
  API を提供する仕組み
- HTML をもとに作られた
  構造的なナニか(雑)

DOM のイメージ

```html
<html>
  <head>
  </head>
  <body>
    <div id="list">
      <div class="item">あ</div>
      <div class="item">い</div>
      <div class="item">う</div>
      <div class="item">え</div>
      <div class="item">お</div>
    </div>
  </body>
</html>
```

↑ HTML

DOM →



クライアントサイド JavaScript でやりたいこと

- DOMの状態を取得する
- DOMを書き換える
- DOMにイベントを割り当てる

JavaScript の問題点：

DOM操作はわかりづらい

DOM操作がしづらい

```
<html>
  <body>
    <p class="hoge">ほげほげ</p>
  </body>
</html>
```

hoge クラスの文字列を取得したい

```
document.getElementsByClassName('hoge')[0].textContent;
```


jQuery
write less, do more.


JS

jQuery

- DOM操作やAjaxを直感的に
  行えるようにするライブラリ
- ブラウザ間の差を吸収してくれる
- 盛り上がりは収まったけどまだまだ人気

jQuery を読み込む

```
<html>
  <head>
    <script src="jquery-1.11.2.min.js"></script>
  </head>
  <body>
    <p class="hoge">ほげほげ</p>
  </body>
</html>
```

DOMの状態を取得する

- jQuery関数を使う
- **$(‘セレクタ’)**
- CSSライクなセレクタで様々なDOMを選択できる

DOMの状態を取得する

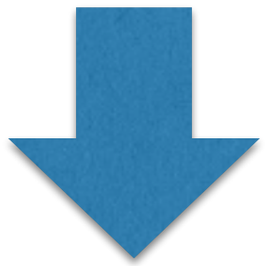
```
<html>
  <body>
    <p class="hoge">ほげほげ</p>
  </body>
</html>
```

hoge クラスの文字列を取得したい

```
$('.hoge').text();
```

DOMを書き換える

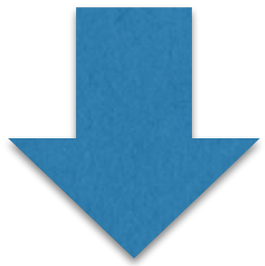
```
$('.hoge').text('ふがふが');
```



```
<html>
  <body>
    <p class="hoge">ふがふが</p>
  </body>
</html>
```

DOMを書き換える

```
$('.hoge').addClass('newhoge');
```



```
<html>
  <body>
    <p class="hoge newhoge">ふがふが</p>
  </body>
</html>
```

JS

DOMを書き換える

```
$('.hoge').removeClass('hoge');
```

```
<html>
  <body>
    <p class="newhoge">ふがふが</p>
  </body>
</html>
```

ワーク

DOMの書き換えをやってみる

- dom / index.html を開いて、
  ブラウザコンソールから要素を操作してみましょう

DOMにイベントを割り当てる

```
$('.hoge').click(function(){
   alert('クリックしたな！！！');
});
```

DOMにイベントを割り当てる

```
$('.hoge').click(function(){
  var text = $(this).text();
  alert(text + 'をクリックしたな！！！');
});
```

ワーク

イベント割り当てをやってみる

- dom / index.html を開いて、
  ブラウザコンソールから要素を操作してみましょう

jQuery 補足

- すべての要素に対して操作をする
- 要素にデータを埋め込む

すべての要素に対して操作をする

```
var i = 1;
$('.hoge').each(function(){
    var text = $(this).text();
    $(this).text( i + ':' + text );
    i = i + 1;
});
```

要素にデータを埋め込む

```
// 要素にデータを埋め込む
$('#header').data({ hoge: 'fuga' });

// 要素のデータを取り出す
console.log( $('#header').data('hoge') ); //=> 'fuga'
```

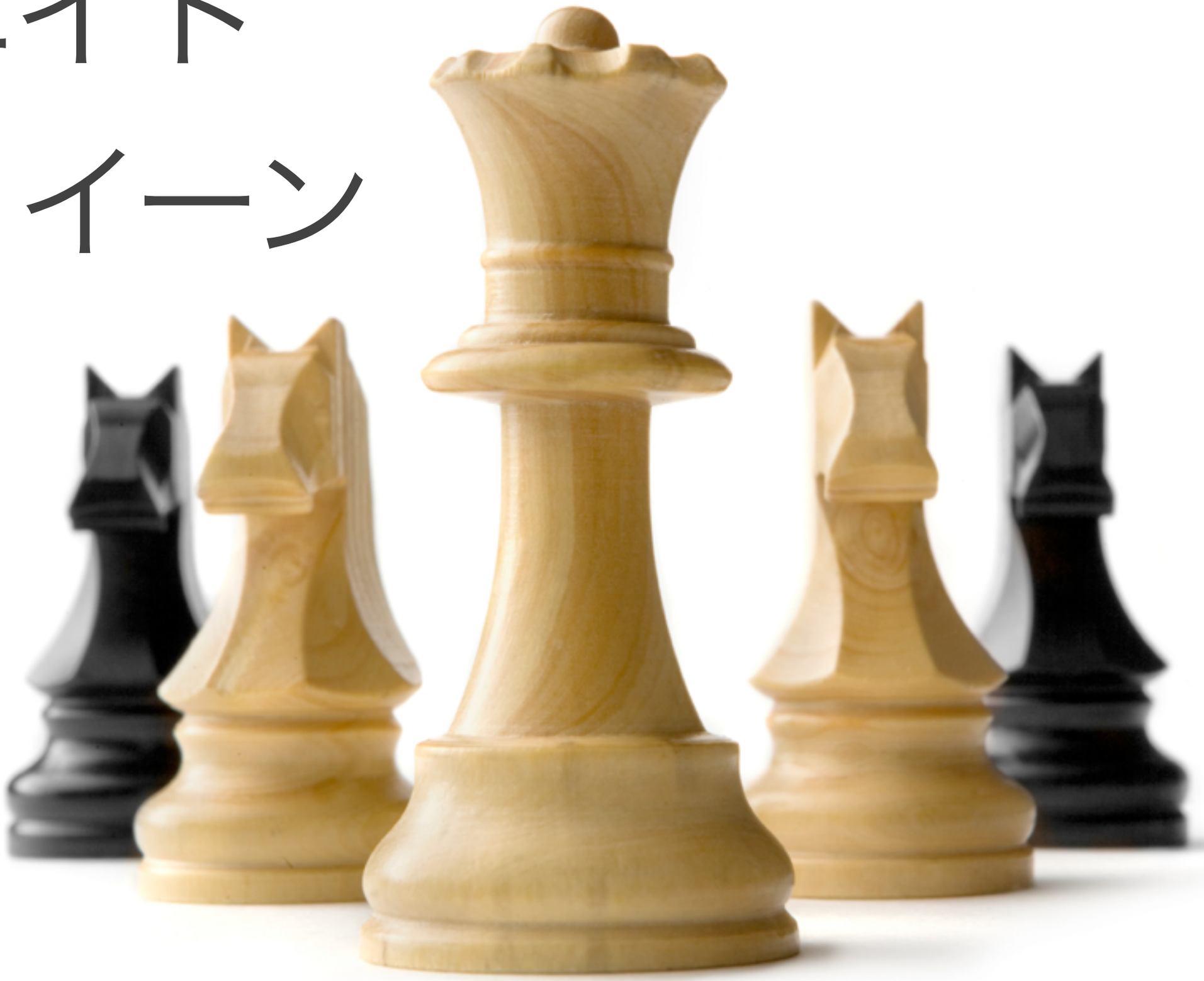
- HTML5のカスタムデータ属性
- 要素そのものに任意のデータを
  紐付けることができる

ワーク

ゲームを作ろう

ワーク

エイト
クイーン



ワーク

エイトクイーン

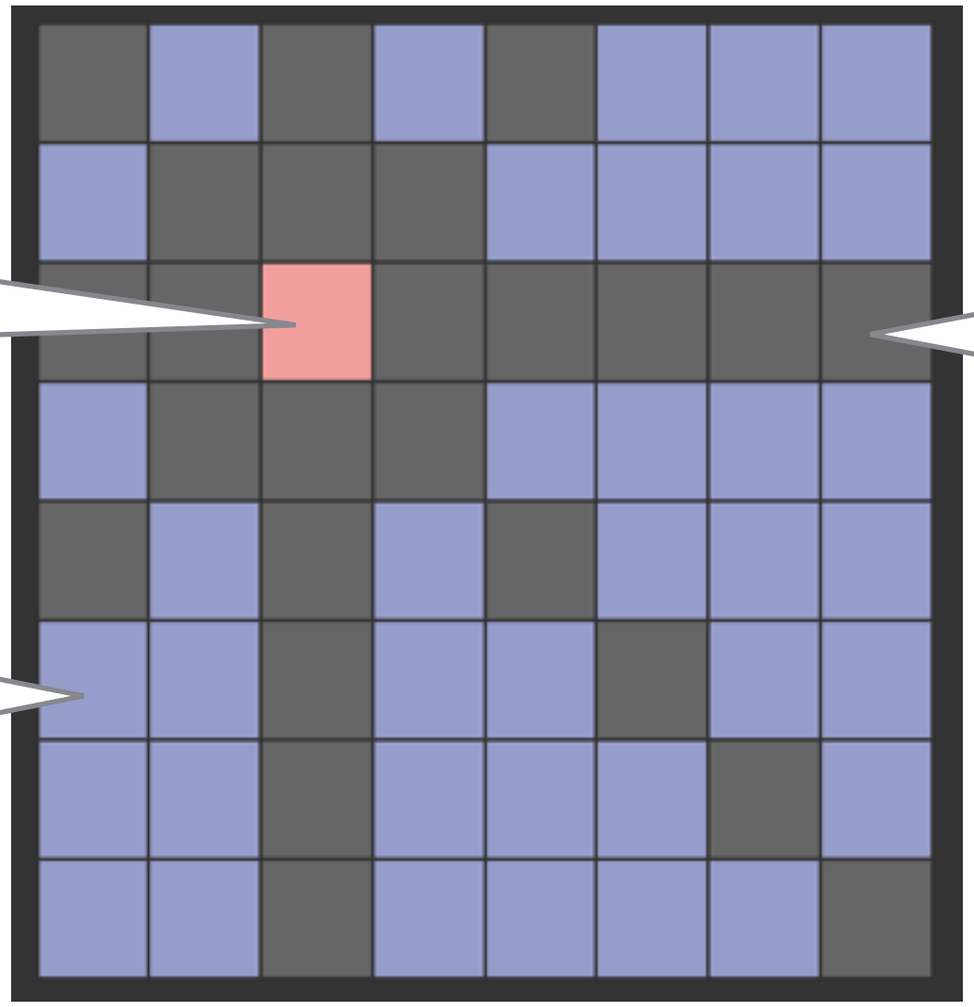
- クイーンを8体ならべるゲーム
- どのクイーンも他のクイーンを取れない位置に置かないといけない

ワーク

サンプルコードの仕様

.queen
クイーンを
置いたマス

.disable
クイーンを
置けないマス



.able
クイーンを
置けるマス

Ajax

Ajax

- Asynchronous JavaScript + XML
- 非同期通信を行う技術の総称
    - Ajax という名前の何かがあるわけではない
- ページ遷移なしで画面を更新することが可能

SPA

- Single Page Application
- Ajax などを駆使してページ遷移なしで動作する
  Webアプリケーションのこと
- 巨大な SPA の例： Gmail
- ちゃんと設計しないと死ぬ

ワーク

jQuery で Ajax してみよう

ワーク

jQuery での Ajax

- jQuery.ajax()
- jQuery.get()
- jQuery.getJSON()
- jQuery.post()

ワーク

課題

- 今日の天気を API から取得して表示してみましょう

- よさげな API を探してきて遊んでみましょう

JSフレームワーク

JSフレームワーク

- クライアントサイドも大規模になってくるとコードが複雑になりがち
    - 枠組みにそって開発すると楽
        - ドメインロジックとプレゼンテーションの分離
        - テスタビリティ
- JSだとMVVMパターンが多い

主要JSフレームワーク

- AngularJS
- Backbone.js
- Ember.js
- Vue.js
- Flux + React

3. サーバーサイド JavaScript

CommonJS

- ここ数年でJSのポテンシャルが再発見された
    - ブラウザ以外でもJS使おう！！
- しかし独自実装JSが乱立した
    - 統一するために作られた規格が CommonJS

従来のJSは I/O が貧弱

- 標準入出力
- File I/O
- Network I/O
- require/include

従来のJSは I/O が貧弱

(すごい雑なイメージ)

今までのJS + I/O
= CommonJS

Node.js

- 最もメジャーな CommonJS 実装
- Google V8 エンジンで動作する

npm

- Node.js のパッケージ管理ツール
    - Gem とか CPAN みたいなもん
- package.json で管理する
    - 手で書くことはあまりない

npm の使い方

- (資料を参照)

その他、流行りの
ツールとか

ガンガンいくけど、まぁ
「こんなのがあるんだー」程度で

nodebrew

- node(npm)のバージョン管理ツール
  - 現状の Node.js の最新版(0.12.x)は結構不安定
  - 0.10.x と使い分けたいことも多い
- plenv とか rbenv みたいなもん

Grunt / Gulp

- タスクランナー

- Gulp の方がシンプルに
  記述できて人気っぽい

Socket.IO

- WebSocket を実現するライブラリ
    - (正確にはちょっと違う)
- リアルタイムWebを構築できる
    - チャットみたいな

Browserify

- クライアントサイドJSでも
  nodeモジュールが使いたい！
- しかしブラウザでは require() は使えない
- 全部ひとつのファイルに固めればいいんじゃね！？

Bower

- フロントエンド開発用
  パッケージ管理ツール
    - Twitter製

- サーバーサイドとクライアントサイドの
  モジュールを区別して管理できる

    - 二重管理が面倒くさいという説も

Bower

- npm で全部やったらだめなの？
    - 別に問題ないです

Yeoman

- 「よーまん」と読むらしい
- Scaffolding のツール
- サーバーサイドJSはツールが多くてよくわからん
    - Yeoman「俺が全部面倒見てやるよ！！！」
- 「Yo」「Bower」「Grunt」
  で構成されている

ワーク

ChatOps してみよう

ワーク

ChatOps

- Chat の Bot を使っていろいろなことをやる(雑)

ワーク

Slack

- いまイケイケのチャットツール

- Bot を作るのが簡単

ワーク

Hubot

- チャット Bot を簡単に作るためのツール
    - GitHub製
    - JavaScript または CoffeeScript で Bot の挙動を記述する
- Slack や HipChat をはじめ、さまざまなチャットツールの Bot を作れる

ワーク

課題

1. おみくじ機能を作ってみましょう

2. 天気を教えてくれる機能を作ってみましょう

3. 好きな機能を考えて実装してみましょう