Web 应用架构指南

45分钟学会五个前端框架

@island205

@寸志 是我的真名

前端架构师@陆金所

陆金所大前端招聘！
https://github.com/island205/Lu-Hire

点评、Teambition、陆金所打杂工
JavaScript 模块化、前端框架、前端性能优化
写书、翻译、Github、微博、知乎

招聘！

前端外刊评论

qianduan.guru

阿里、腾讯等 10 位左右的前端工程师组成的编辑团队，每周一篇值得阅读的前端原创或翻译文章

红包！

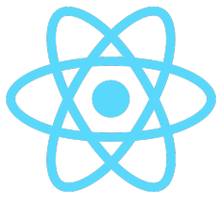
2010.06

2010.10

2011.12

辣么多的前端框架？！

2010.10
reborn in 2016?



2013.03

2014.03

Backbone.js Knockout Ember.js Angular 2 React Vue

怎么学得过来?

看看 TodoMVC

1. 这么多框架，要学到什么时候?
2. 看 TodoMVC 的例子，跟着做了，简单，但复杂的应用怎么做?

万变不离其宗[指南]

了解了宗，十八班武艺，众多框架马上学会
了解了宗，十八班武艺，众多框架信手拈来

这个宗，就是我今天要讲的指南

第一式： 学会编写组件

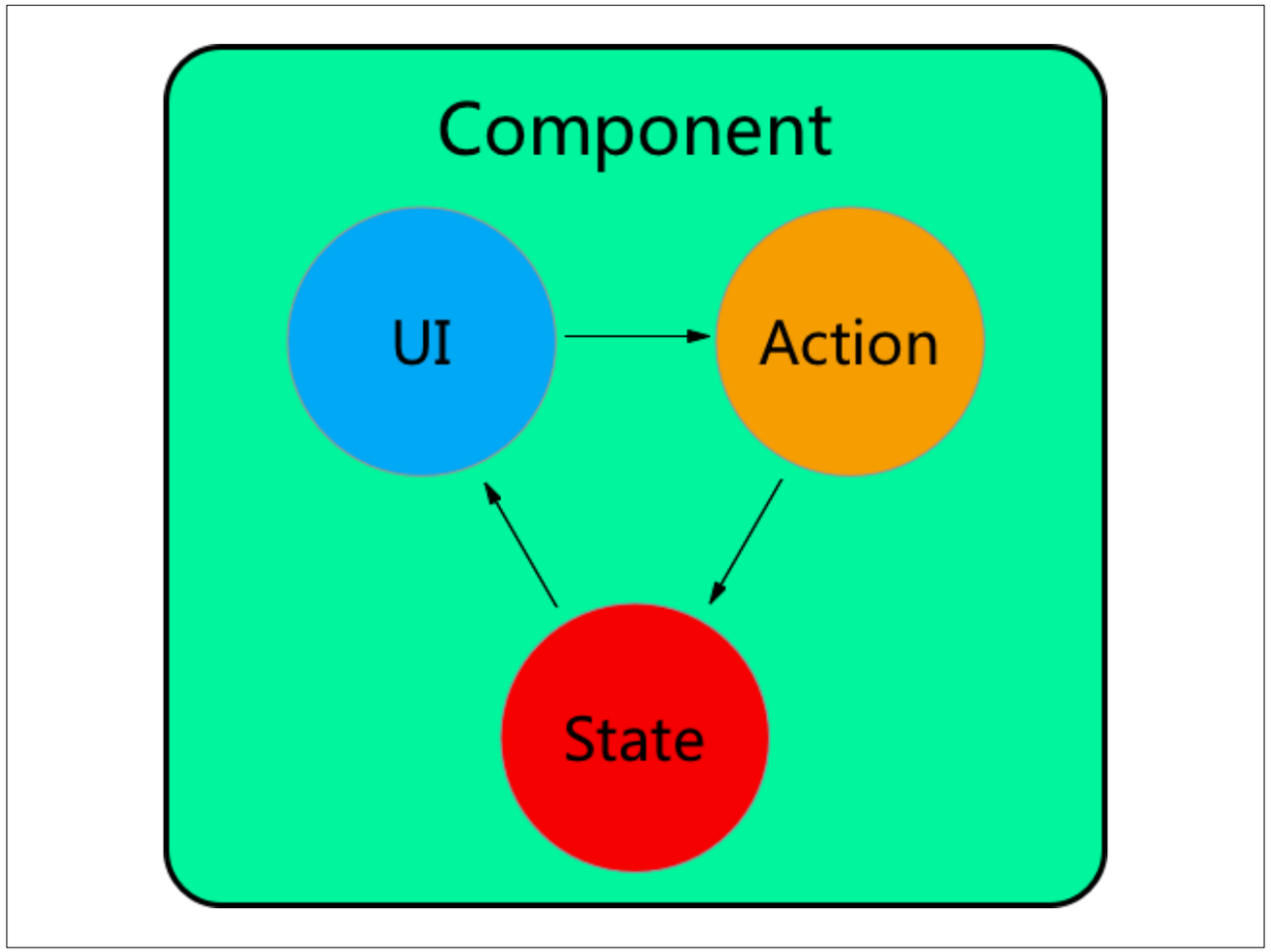
相比较 Angular 1 庞杂的概念而言，React 提出的组件模式简单，但也很强大。React 的流行，影响了整个前端框架所提供的开发模式——倾向于提供组件化开发的方式

将应用中的任何 UI 相关的部分都看成了一个个组件，就如 HTML 标签一样

那像这样的 HTML 组件该如何编写呢?

按照 MVC 的思路解构组件！

组件化开发的核心模型是 MVC，我始终认为 MVC 是设计模式中最重要的概念！



一个组件分成三部分：视图、控制器、状态

框架提供了哪些要素，便于我们编写一个组件?

1. M 绑定到 V：数据更新，View 自动更新
    1. 根据 M 更新到 V 的方式不一样，可以把框架分成两类
        1. React Virtual DOM：对比虚拟 DOM
        2. 不是 Virtual DOM 的
    2. 根据 M 变化检测的触发方式，也可以把框架分成两类
        1. Angular 一 1 $apply 脏检测，2 Zone.js
        2. 非 Angular 的 Setter
2. V 绑定到 C：绑定 DOM 事件
3. C 更新 M
    1. 也可以分成两种架构
        1. 直接操作 M
        2. 不直接操作 M 的

```
var checkbox = new Checkbox()
$(document.body).append(checkbox.el)
```

Backbone 组件最基础的用法

```
export default Backbone.View.extend({
  tagName: 'span',
  className: 'Checkbox',
  events: {'click .fake': 'toggle'},
  initialize: function () {
    this.state = new Backbone.Model({ checked: false })
    this.state.on('change:checked', this.render.bind(this))
    this.render()
  },
  toggle: function () {
    this.state.set({checked: !this.state.get('checked')})
  },
  render: function() {
    var checked = this.state.get('checked')
    this.$el.html(`
      <i class="fake${checked ? ' checked' : ''}"></i>
      <input type="checkbox" ${checked ? 'checked' : ''}></input>
    `)
  }
})
```

state 是 M

events 是 C

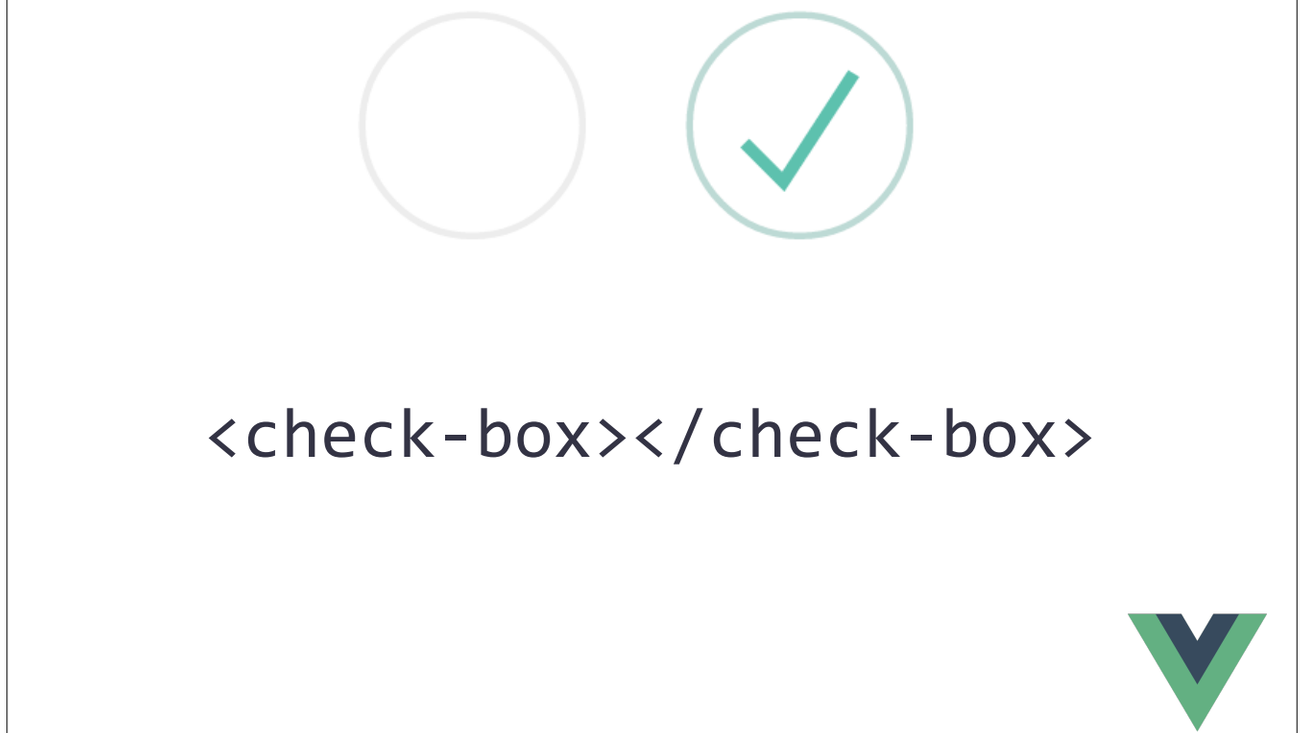
render 是 V

Backbone 组件最基础的用法

```
<span class="Checkbox">
    <i class="fake {{if checked 'checked' ''}}"
        {{action 'toggle' on="click"}}></i>
    <input type="checkbox"
        checked="{{if checked true false}}"/>
</span>
```

```
export default Ember.Component.extend({
  checked: false,
  actions: {
    toggle() {
      this.set('checked', !this.get('checked'));
    }
  }
});
```

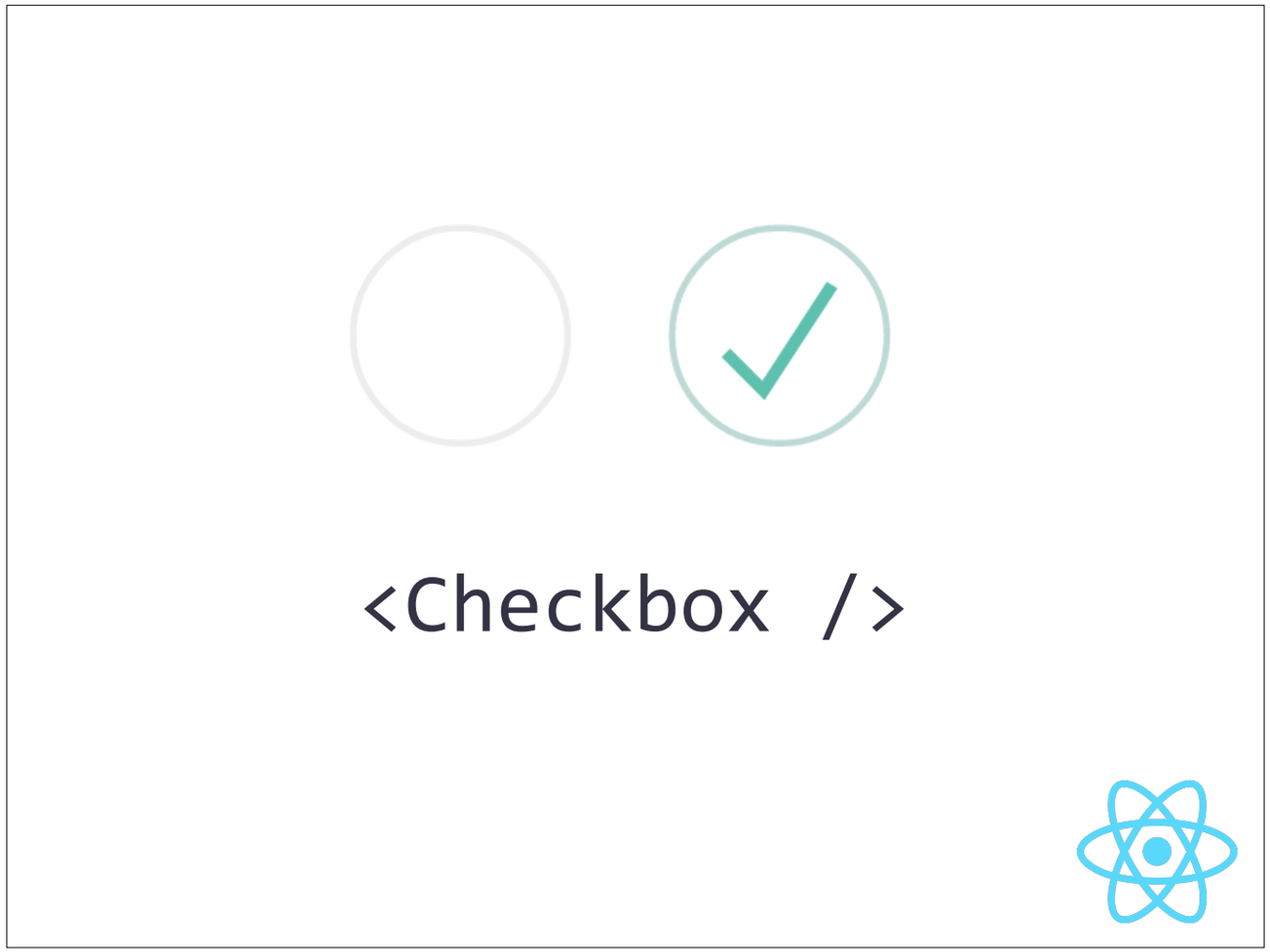
checked M
hbs V
actions C



Vue 组件的书写方法就和 HTML 基本一致

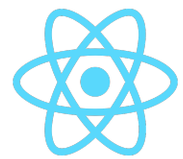
```
var Checkbox = Vue.extend({
  template: `
    <span class="Checkbox">
      <i class="fake" :class="{checked: checked}”
         @click="toggle"></i>
      <input type="checkbox" v-model="checked">
    </span>
  `,
  data: function () {
    return {
      checked: false
    }
  },
  methods: {
    toggle: function () {
      this.checked = !this.checked
    }
  }
})
```

Data M
Template V
methods C



React 的也是如此

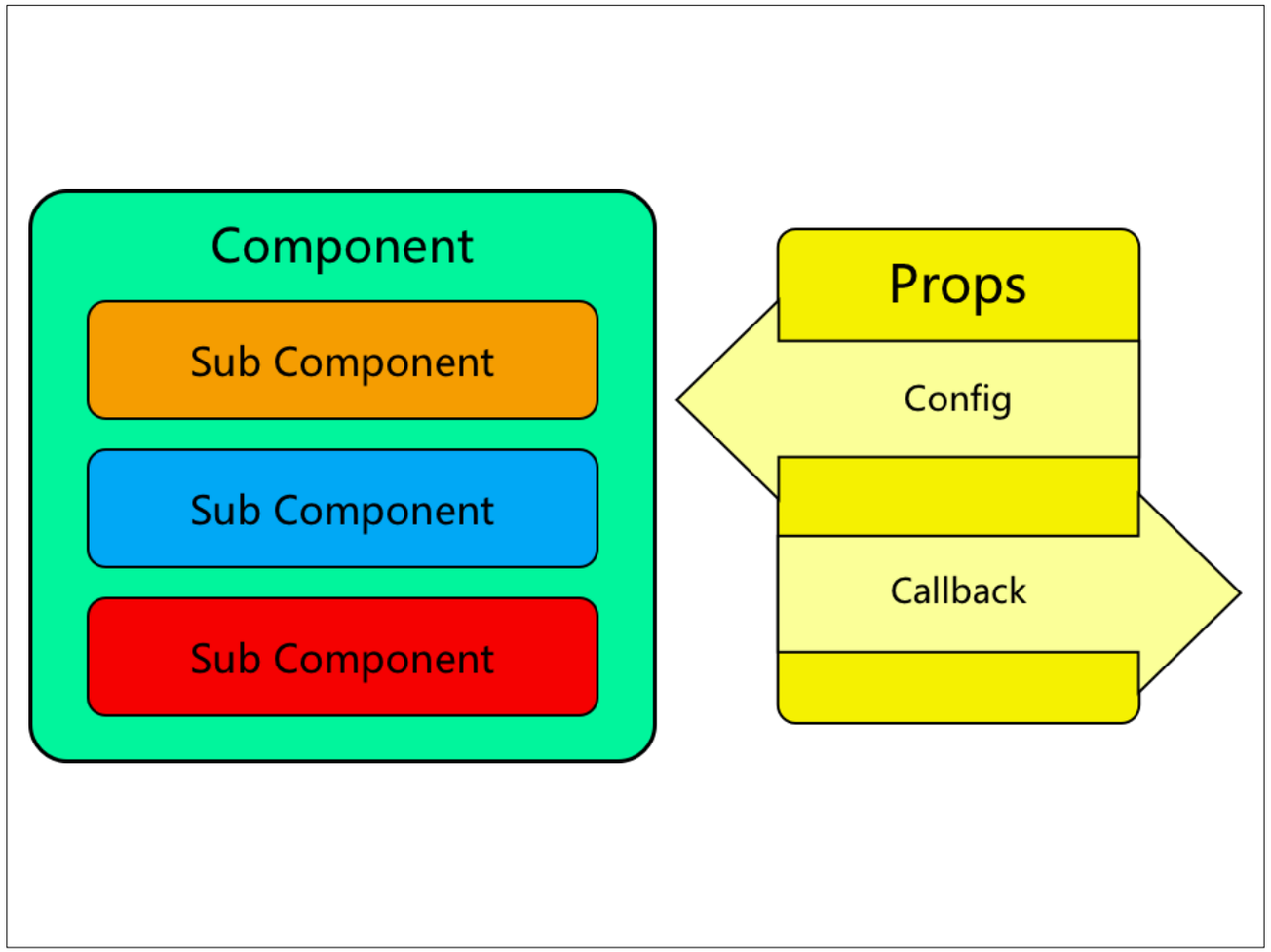
```
export class Checkbox extends React.Component {
  state = {
    checked: false
  }
  toggle() {
    this.setState({
      checked: !this.state.checked
    })
  }
  render() {
    var checked = this.state.checked
    return (
      <span className="Checkbox">
        <i className={checked ? 'fake checked' 'fake'}
          onClick={this.toggle.bind(this))}></i>
        <input type="checkbox"
          checked={checked}/>
      </span>
    )
  }
}
```



没有 Angular 的例子？

在 Angular 1 中，我们可以用 Directive 来实现组件架构，封装和隔离在复杂的应用面前是很有意义的。

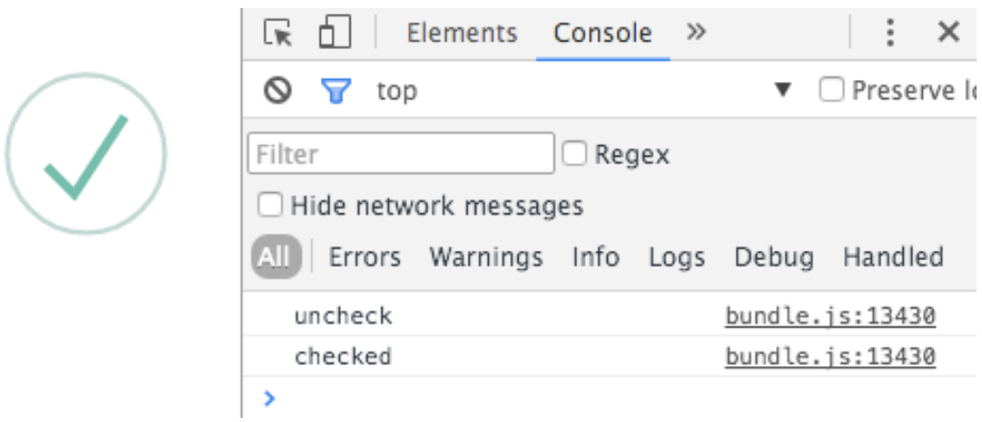
第二式： 用组件搭积木



小组件组合成大组件，分而治之

子组件存在关联关系，可以通过大组件来处理

大组件通过属性，注入配置和事件回调，来控制小组件的行为，响应小组件的事件




```
var checkbox = new Checkbox({
  checked: true,
  onChange: function(checked) {
    console.log(checked ? 'checked' : 'uncheck')
  }
})
$(document.body).append(checkbox.el)
```

传入配置和回调

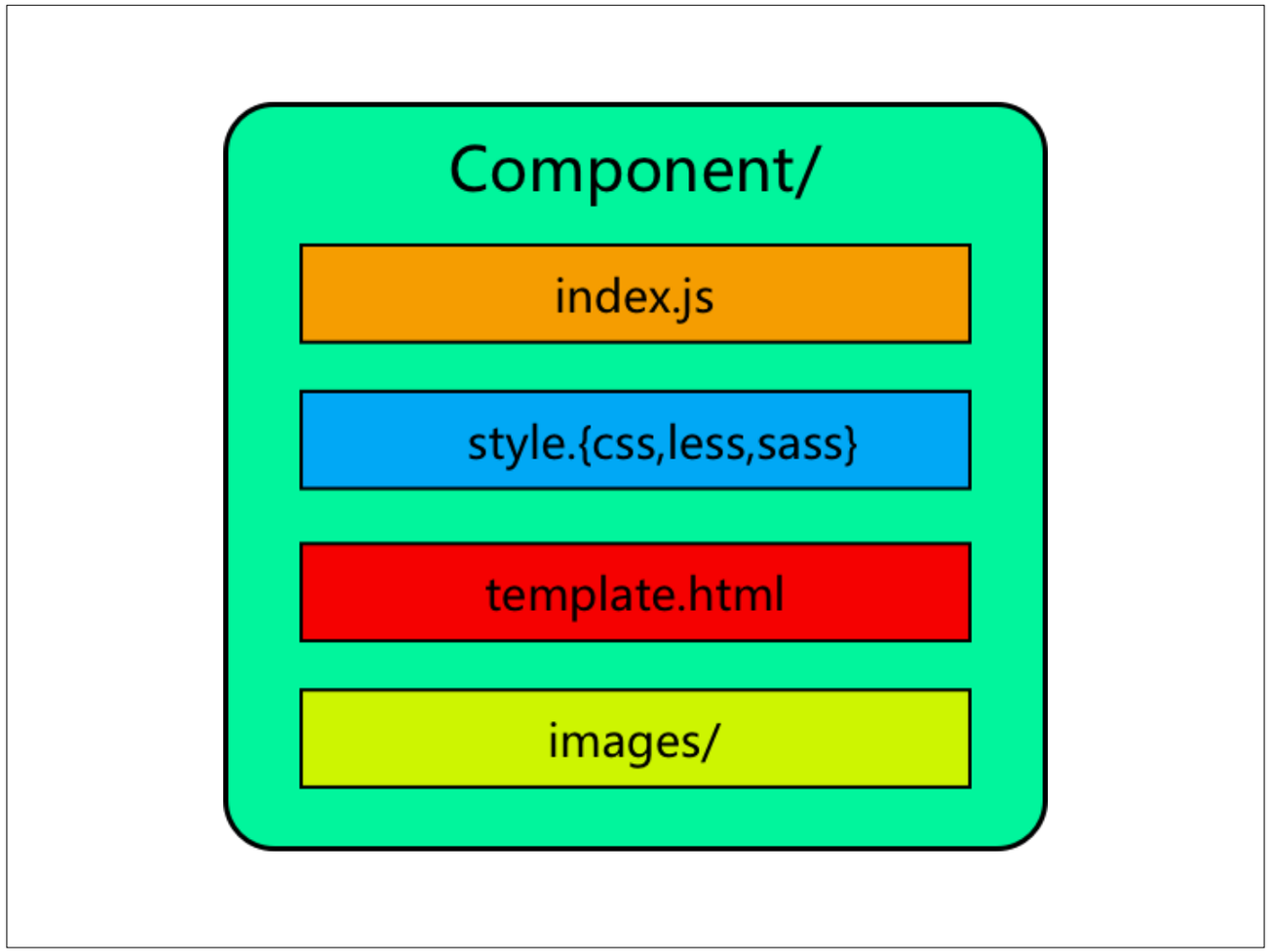
```
initialize: function (options) {
   this.options = Object.assign({
     checked: false,
     onChange: function() {}
   }, options)
   this.state =
    new Backbone.Model({ checked: options.checked || false })
   this.state.on('change:checked', this.render.bind(this))
   this.render()
 },
 toggle: function () {
   this.state.set({checked: !this.state.get('checked')})
   this.options.onChange(this.state.get('checked'))
 }
```

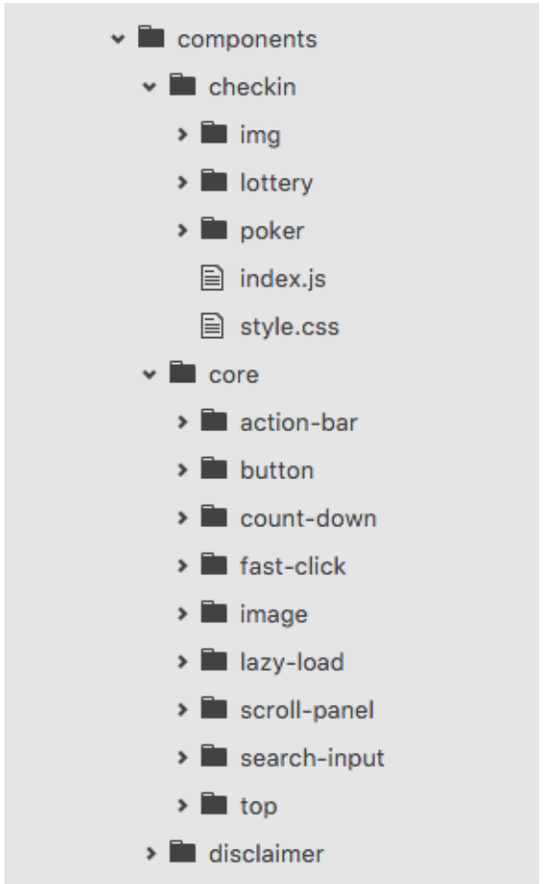
```
{{checkbox checked=checked
  onChange=(action 'onCheckChange')}}
```

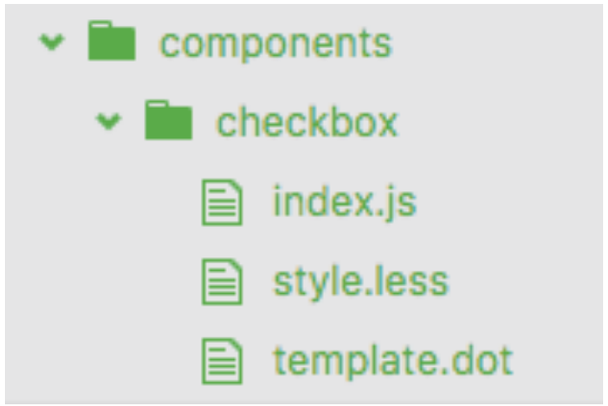
```
<div class="Todos">
  <todos-header @add-todo=“addTodo"
   :all-done.sync="allDone"></todos-header>
  <todo-list :todos="filteredTodos"></todo-list>
  <todo-footer :todos="filteredTodos"
    :visibility="visibility"></todo-footer>
</div>
```

第三式：

把模块的 Template、Style、JavaScript、Images 放到一个目录下！


Component/
index.js
style.{css,less,sass}
template.html
images/


components
checkin
img
lottery
poker
index.js
style.css
core
action-bar
button
count-down
fast-click
image
lazy-load
scroll-panel
search-input
top
disclaimer


components
checkbox
index.js
style.less
template.dot

组件和样式约定

```
<span class="Checkbox">
    <i class=“fake"
      :class="{checked: checked}”
      @click="toggle"></i>
    <input type=“checkbox"
       v-model="checked">
</span>
```

```
.Checkbox {
  .fake {
    text-align: center;
    width: 40px;
    height: 40px;
  }
  input {
    display: none;
  }
}
```

Checkbox / checkbox-component

1. 使用组件名作为样式的命名空间
2. 样式使用预处理器，采用嵌套的方式编写

组件和样式约定

```
.Todo {
  position: relative;
  font-size: 24px;
  border-bottom: 1px solid #ededed;
  .Checkbox {
    position: absolute;
    top: 0;
    bottom: 0;
    margin: auto 0;
  }
}
```

保证单个组件 CSS 的纯度，与组件间搭配的、比如间距、定位等信息，写在父组件中！

如何把散沙捏在一起？

代码拆散，就像散沙，了放在一个组件文件夹中，那该如何打到一起运行放在浏览器中运行呢?

1. 首先模块化地编写这些组件
2. 通过浏览器端的 Loader 或者 Build 工具来将模块组合起来
3. 下面是一些可选的方案和技巧
4. 当然，目前来看，最好的就是祭出大 Webpack

```less
// app.css
@import "./components/todos.less";
@import "./components/todos-header.less";
@import "./components/check-box.less";
@import "./components/todo-footer.less";
@import "./components/todo-list.less";
@import "./components/todo.less";
```

把所有 CSS 文件登记在一个文件中，通过预处理器合并。

```
import Template from './template.dot'
import './style.less'

export default Backbone.View.extend({
  tagName: 'span',
  className: 'Checkbox',
...
```

将其他文件都 Import 到 index.js 中!

祭出 Webpack！

browserify/grunt/gulp 其实也能做

Webpack

不是 JavaScript module Loader 而是 Component module loader

常用的插件介绍

Checkbox.vue

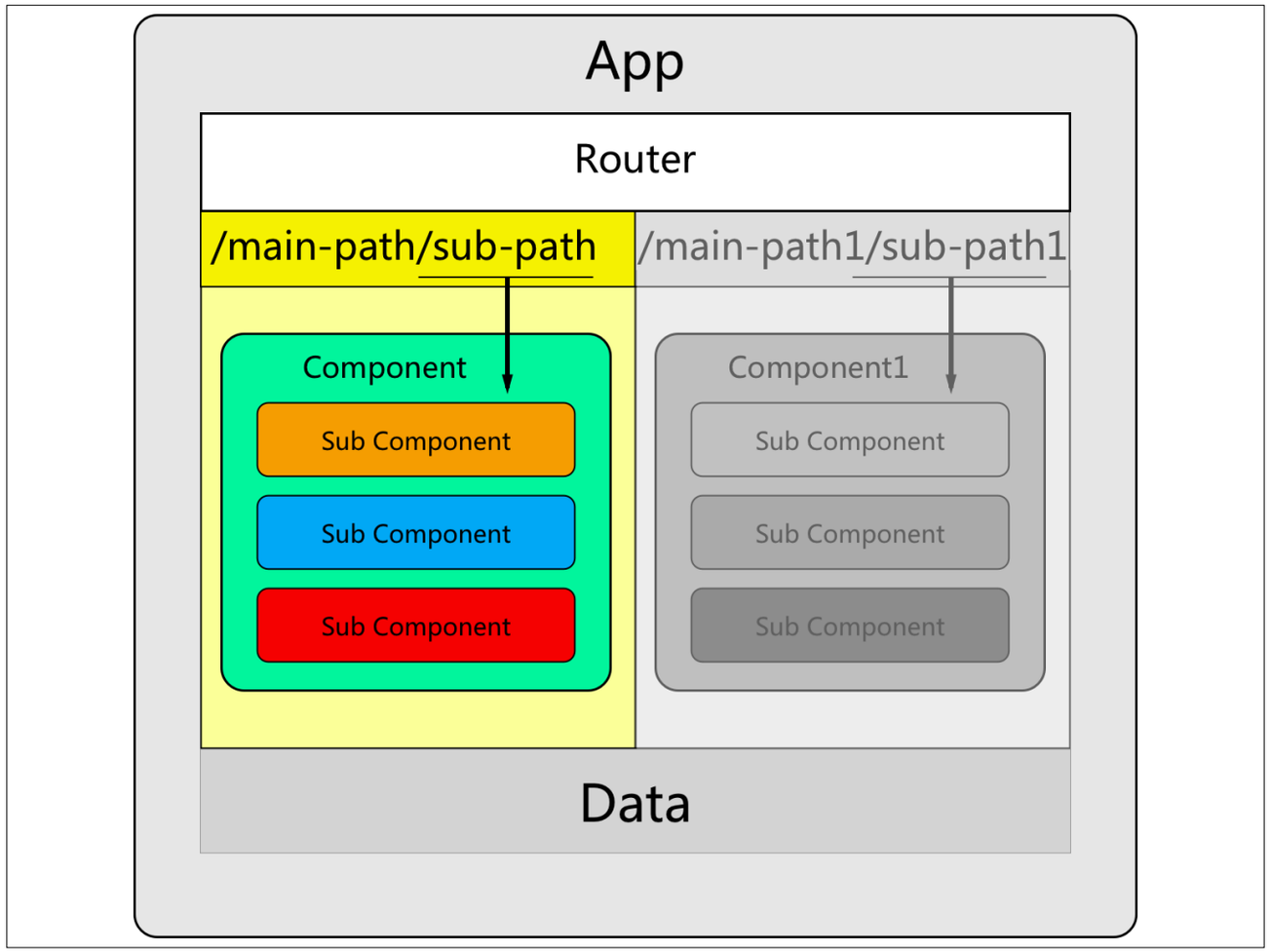
```
<template>
    <span class="Checkbox">
    </span>
</template>

<script>
export default {
…
}
</script>

<style lang="less">
.Checkbox {
…
}
</style>
```

Tips:

开发运行时和线上运行时分开！

第四式：

用路由把组件链成应用


App
Router
/main-path/sub-path
/main-path1/sub-path1
Component
Sub Component
Sub Component
Sub Component
Component1
Sub Component
Sub Component
Sub Component
Data

两层 MVC 的结构。

常用的 Route 模块

Backbone	Backbone.Router
Ember.js	App.Router
Angular	ng-router/ui-router angular 2 router
React	react-router
Vue	vue-router

```
app.get('/app/*', function (req, res, next) {
  res.render('app')
})
```

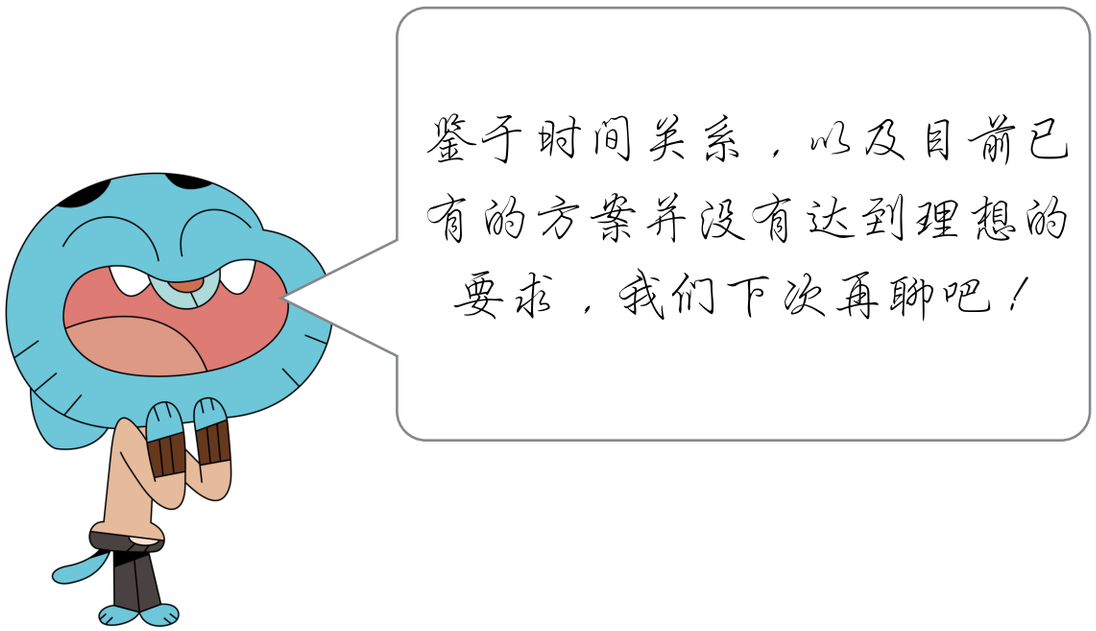
为了避免应用在使用过程中刷新，服务端让所有该页面承载的路由，都返回 APP 页

业务复杂时，按照业务切分为多个独立的单页应用

第五式：

聪明、准确的数据层

请求
缓存
同步
触发


鉴于时间关系，以及目前已有的方案并没有达到理想的要求，我们下次再聊吧！

Q&A