

前端服务化之路

2015.07.12 Herman

who is this guy



- ◆ 前端 - 赫门 (Herman)
- ◆ 来自臺灣
- ◆ 阿里巴巴 - CXDC团队



什么是前端服务化？

先来聊聊前端

从2005年，的AJAX开始

前端有了一个新的定义

来看看我们经历了哪些时代

从 frontpage 开始

Server Side Solution

Client Side MV*

Node.js

Web 2.0

Responsive Design

Hybrid App

HTML5

CSS3

ES5 & ES6

CommonJS

CMD & AMD

.....

dojo

mootools

JQUERY

prototypejs

ejs

mass

ember

knocko
ut

yuí

backbone

angular.js

dojo

mootools

JQUERY

ejs

react

backbone

angular.js

摩尔定律同样适用于前端

“每18个月~24个月，前端都会难一倍”

干

前端的技术发展，可以分成两类

一、空间上的提升

“我们被限制在客户端做开发，

能做些什么，取决于客户端给的权力”

二、质量上的提升

“这么多的前端框架，

其实就是为了页面做得更快，做得更好”

“跳脱不了组件两个字”

前端难度越变越高

“是否象徵前端的价值越变越高？”

高端的前端团队开始思考转型

- ◆ 全栈工程师 ~ 前端需要具备后端的能力
- ◆ 产品工程师 ~ 前端需要具备产品思维
- ◆ 跨端工程师 ~ 前端需要有跨端原生开发的能力
- ◆ 体验工程师 ~ 前端需要具备UX的技能

“真实世界里，符合标准的前端却越来越少”

所有的产品都需要前端

来看看某知名互联网公司的现状



A Venn diagram consisting of two overlapping circles. The larger circle on the left is dark brown and contains the text '公司所有的產品'. The smaller circle on the right is green and contains the text '有前端支持的產品'. The two circles overlap on the right side of the brown circle.

公司所有的產品

有前端支持的產品

并不是所有的产品都面向消费者

重视功能胜于体验的产品

面向特定族群使用者的产品

快速迭代的实验性产品

后端与前端的比例



10



1

“只要让十分之一的后端，做前端的事”

“前端的生产力可以提升 100%”

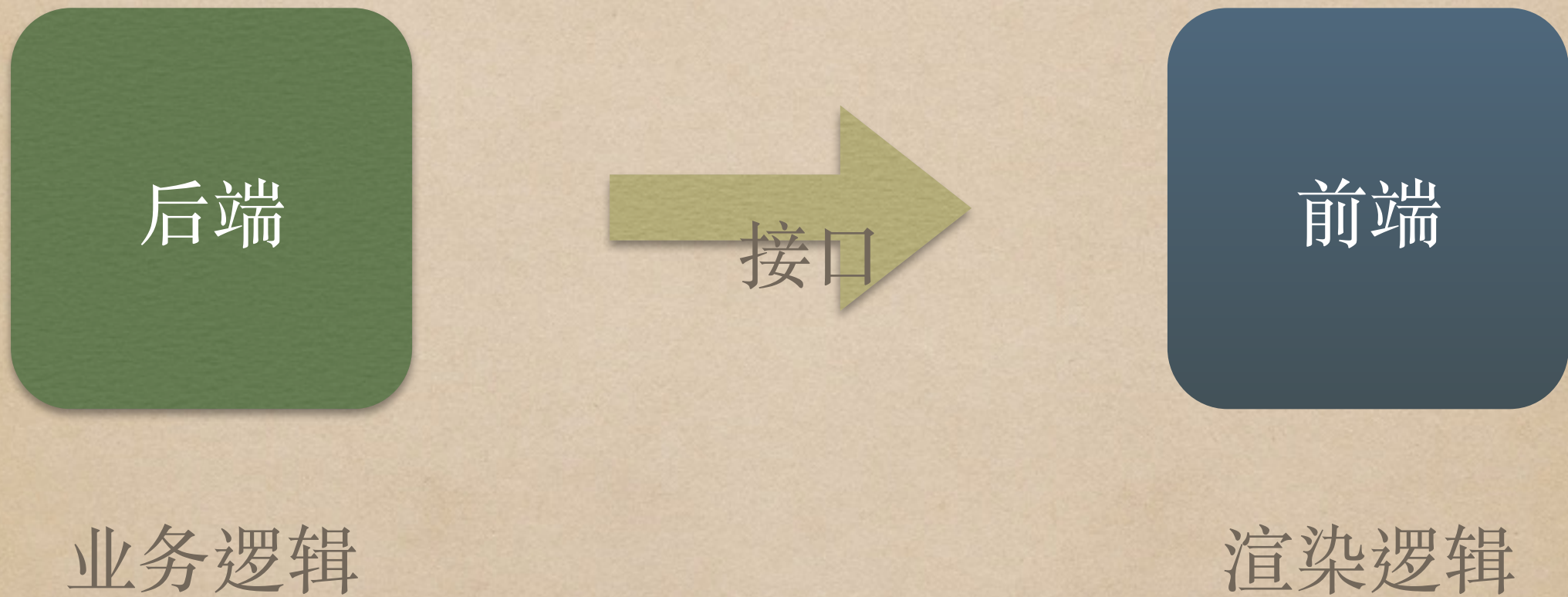
—小馬

我们的思考



过去我们推动前后端分离

接口先行的前后端分离



“后端的业务逻辑，以服务方式提供给前端”

“前端的渲染逻辑”

“能不能以服务方式输出给后端？”



业务逻辑



前端服务



渲染逻辑

前端服務化之路

早就有人做过了啊？

Bower Components

jQuery Plugins

XXX Gallery

YUI

Bootstrap

NPM


React Components

“没有最完美的方案，只有相对适合的方案”

服务化 \neq 组件化

看看这个案例

一个典型的后台



David Williams
Art Director ▾

Dashboards

<

Layouts

Graphs

<

Mailbox

16/24

Metrics

NEW

Widgets

Forms

▾

Basic form

Advanced Plugins

Wizard

File Upload

Text Editor

App Views

SPECIAL

Other Pages

<

Miscellaneous

NEW

Search for something...

Welcome to INSPINIA+ Admin Theme.

16

8

Log out

Basic Form

Home / Forms / Basic Form

All form elements With custom checkbox and radion elements.

Normal

Help text

A block of help text that breaks onto a new line and may extend beyond one line.

Password

Placeholder

placeholder

Disabled

Disabled input here...

Static control

email@example.com

Cancel

Save changes

有哪些方案

1. Bootstrap

找个系统拖曳一下，不就出来了吗

不然看著文档写也很简单吧

1. Bootstrap

```
<link href="font-awesome/css/font-awesome.css" rel="stylesheet">
<link href="css/plugins/iCheck/custom.css" rel="stylesheet">
<link href="css/animate.css" rel="stylesheet">
<link href="css/style.css" rel="stylesheet">
<link href="css/plugins/awesome-bootstrap-checkbox/awesome-bootstrap-checkbox.css" rel="stylesheet">

</head>

<body>

<div id="wrapper">

<nav class="navbar-default navbar-static-side" role="navigation">
<div class="navbar-collapse">
<ul class="nav metismenu" id="side-menu">
<li class="nav-header">
<div class="dropdown profile-element"> <span>

</span>
<a data-toggle="dropdown" class="dropdown-toggle" href="#">
<span class="clear"> <span class="block m-t-xs"> <strong class="font-bold">D<
strong>
</span> <span class="text-muted text-xs block">Art Director <b class="caret">
span> </a>
<ul class="dropdown-menu animated fadeInRight m-t-xs">
<li><a href="profile.html">Profile</a></li>
<li><a href="contacts.html">Contacts</a></li>
<li><a href="mailbox.html">Mailbox</a></li>
<li class="divider"></li>
<li><a href="login.html">Logout</a></li>
</ul>
</div>
<div class="logo-element">
IN+
</div>
</li>
<li>
<a href="index.html"><i class="fa fa-th-large"></i> <span class="nav-label">Dashboards<
class="fa arrow"></span></a>
<ul class="nav nav-second-level collapse">
<li><a href="index.html">Dashboard v.1</a></li>
<li><a href="dashboard_2.html">Dashboard v.2</a></li>
<li><a href="dashboard_3.html">Dashboard v.3</a></li>
<li><a href="dashboard_4_1.html">Dashboard v.4</a></li>
</ul>
</li>
<li>
<a href="layouts.html"><i class="fa fa-diamond"></i> <span class="nav-label">Layouts<
</li>
<li>
<a href="#"><i class="fa fa-bar-chart-o"></i> <span class="nav-label">Graphs</span><
arrow"></span></a>
<ul class="nav nav-second-level collapse">
</ul>
</li>
</ul>
</div>
</div>
</div>
```

```

        </div>
      </div>
    </div>
    <div class="hr-line-dashed"></div>
    <div class="form-group">
      <div class="col-sm-4 col-sm-offset-2">
        <button class="btn btn-white" type="submit">Cancel</button>
        <button class="btn btn-primary" type="submit">Save changes</button>
      </div>
    </div>
  </form>
</div>
</div>
</div>
</div>
</div>
<div class="footer">
  <div class="pull-right">
    10GB of <strong>250GB</strong> Free.
  </div>
  <div>
    <strong>Copyright</strong> Example Company &copy; 2014-2015
  </div>
</div>
</div>
</div>

```



```


<script src="js/jquery-2.1.1.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/plugins/metisMenu/jquery.metisMenu.js"></script>
<script src="js/plugins/scroll/jquery.scroll.min.js"></script>


<script src="js/inspinia.js"></script>
<script src="js/plugins/pace/pace.min.js"></script>


<script src="js/plugins/iCheck/icheck.min.js"></script>
<script>
$(document).ready(function () {
  $('i-checks').iCheck({
    checkboxClass: 'icheckbox_square-green',
    radioClass: 'iradio_square-green',
  });
});
</script>

```


Problems

- ◆ 这些代码并不是人类能阅读的。
- ◆ 页面结构产生出来后就不可逆了。
- ◆ 在难以理解的HTML上叠加依赖HTML结构的JS，是场悲剧
- ◆ 如果页面内容还要从后端动态吐出，更是场悲剧。

2. 解偶 ~ 用接口 + 组件

```
<header id="header"></header>

<div class="wrapper">
  <nav id="nav"></nav>

  <div class="panel">
    <form id="form"></form>

    <button id="btn_cancel">Cancel</button>
    <button id="btn_submit">Save changes</button>
  </div>
</div>
```

```
<script>
  require(['header', 'nav', 'form'], function(Header, Nav, Form) {
    Header.init({
      container: "#header",
      config: './queryHeader.do'
    });

    Nav.init({
      container: "#nav",
      config: './queryNav.do'
    });

    window.formModule = Form.init({
      container: "#form",
      config: './queryForm.do'
    });
  })
</script>
```


2. 解偶 ~ 用接口 + 组件

```
<header id="header"></header>
```

```
<div class="wrapper">
```

```
<script>
```

```
$(function() {
```

```
    $('#btn_submit').on('click', function() {
```

```
        if (window.formModule.validate()) {
```

```
            $.ajax({
```

```
                url: './submitForm.do'
```

```
                data: window.formModule.getValue()
```

```
            });
```

```
        }
```

```
    });
```

```
});
```

```
</script>
```

```
...  
window.formModule = Form.init({
```

```
    container: "#form",
```

```
    config: './queryForm.do'
```

```
});
```

```
})
```

```
</script>
```


Problems

- ◆ HTML 形同虚设，而且难以理解
- ◆ 一大段的Script只是为了初始化HTML
- ◆ 其实写HTML跟执行JS，是同一件事，缺一不可。却要分开异步进行，就是种难以理解的约定耦合。
- ◆ 表单组件，与原生的按钮组件，是完全不同的概念，调用方式也完全不一样

如果这样如何

```
<my-header config="./queryHeader.do"></my-header>

<div class="wrapper">
  .....
  <my-nav config="./queryNav.do"></my-nav>

  <div class="panel">
    .....
    <my-form id="form"
      .....
      config="./queryForm.do"
      action="./submitForm.do"></my-form>

    <button id="btn_cancel">Cancel</button>
    <button id="btn_submit">Save changes</button>
  </div>
</div>
.....
```


“写上一个标签，就在标签的位置把初始化的事情
给做好，该发请求拿数据的就自己发请求”

要加上事件也很直觉

```
<script>
  documnt.getElementById('btn_submit')
    .addEventListener('click', function() {
      var form = documnt.getElementById('form');
      if (form.validate()) {
        form.post();
      }
    });
</script>
```


“如果我们的组件使用起来跟原生的dom组件一样，该有多好”

WEB COMPONENTS

“服务化并不是组件化”

“服务化的背后，是标准化”

“从来没有一个组件方案，可以统一市场”

“但是我们都在用HTML标签跟CSS”

“我们也脱离不了DOM API去操作元素”

“标准加入后，不管你用不用都存在著”

“标准化的意义是整体概念的一致性”

“于其上，让使用者自由扩展”

技术的提升推动标准的制定

看看其他的案例

- ◆ 有了flash，然后有了HTML5
- ◆ jQuery曾经是公认的标配。现在有了vanilla.js
- ◆ 每个框架都定了类的写法。模块标准也有了业界的共识
- ◆ 而未来类的写法，模块的写法，又会是什么样？

“技术迭适中，标准化是唯一的出路”

那 Web Components 如何呢？

其实一点也不适合前端用

- ◆ 各位都能跟上最新最强的技术。Web Components其实没有解决你们的任何问题
- ◆ Web Components太多标准都太理想化，不贴近现实
- ◆ Web Components无法与现在的开发流程做整合，只会绑手绑脚

“但倒是挺适合，作为 前端服务 提供给后端用”

“没有最完美的技术，只有相对适合的技术”

对于前端服务化来说

- ◆ 要求每个人都有这么强的前端技术太过理想化，先做出产品比较重要
- ◆ 最新最炫的前端技术，跟项目是否成功没有什么直接关系
- ◆ 所谓的“服务”。就是让使用者可以用最小成本合理的使用我们的技术。

我们做了什么



基于Web Componets的组件方案



Flipper.js

Live Demo

定义一个组件

```
<!-- 宣告一个组件: ec-series -->
<web-component name="ec-series">

  <!-- 定义组件的样式 -->
  <style>
    header {
      color: black;
    }
  </style>

  <!-- 定义组件的模版 -->
  <template>
    <header><h2>Baidu - Echarts</h2></header>
    <div class="chart" style="height: 530px; width: 600px"></div>
  </template>

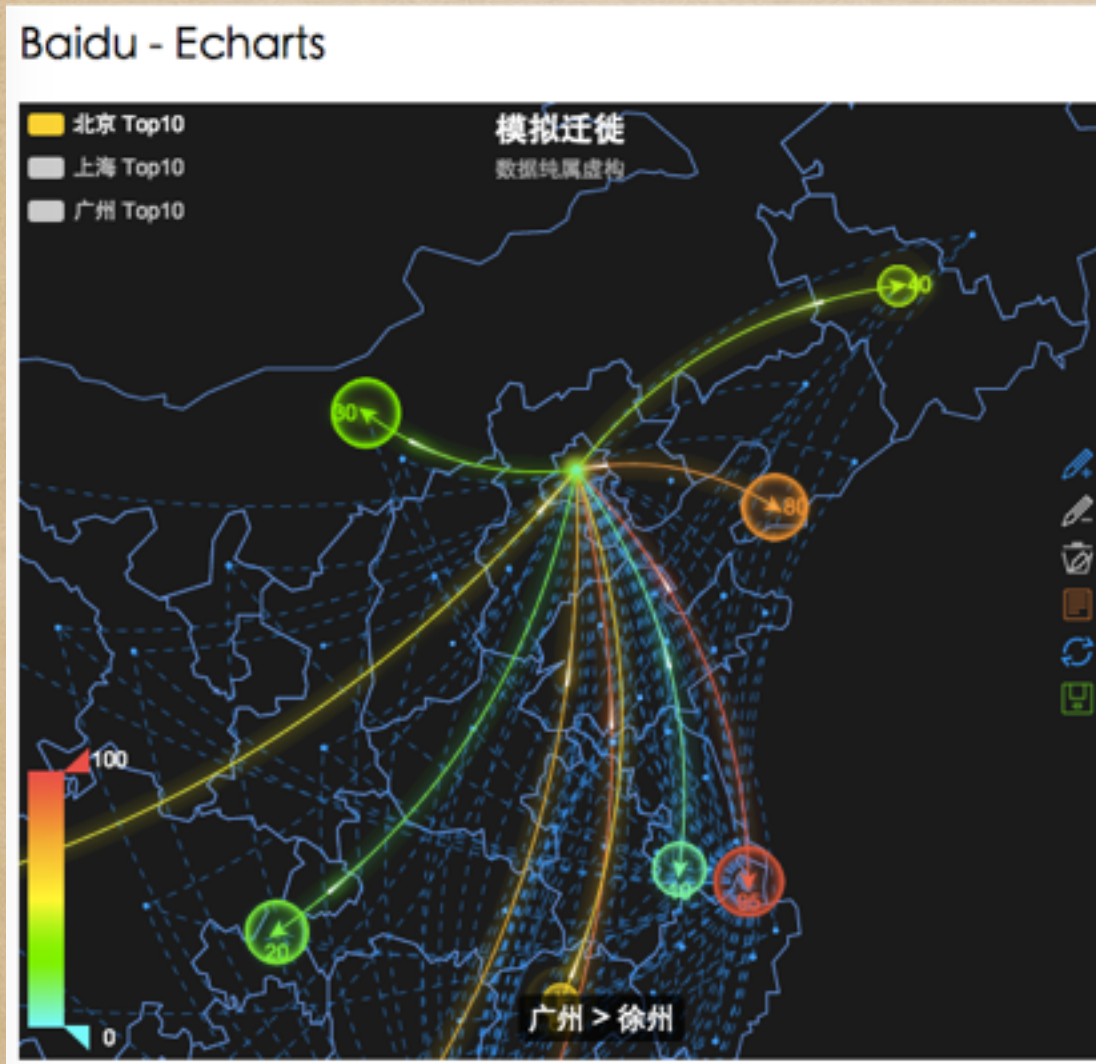
  <script>
    /* 定义组件的依赖: echarts, 可使用AMD或CMD标准 */
    Flipper.register([ '../lib/echarts-all'], function(echarts) {

      /* 回传组件的生命周期配置 */
      return {
        fetch: function() { /* 获取数据 */
          var dataUri = this.getAttribute('data-uri');
          return $.getJSON(dataUri);
        },
        ready: function() { /* echarts初始化 */
          var chart = echarts.init(this.querySelector('.chart'));
          chart.setOption(this.model);
        }
      };
    });
  </script>
</web-component>
```

借用百度的echarts做演示

在页面上直接使用

```
<ec-series data-uri="./data.json"></ec-series>
```



```
data.json  
{  
  1  {"color":["gold","aqua","lime"],"title":{"text":  
      模拟迁徙,"subtext":"数据纯属虚构","x":"center",  
      "textStyle":{"color":"#fff"}},  
      "tooltip":{"  
        trigger:"item","formatter":"{b}"},"  
        legend:  
        "orient":"vertical","x":"left",  
        "data":[[{"  
          name:"北京",{"name":"包头"}], [{"name":"北京",  
          {"name":"北海"}], [{"name":"北京",{"name":"  
          广州"}], [{"name":"北京",{"name":"郑州"}], [{"  
          name":"北京",{"name":"长春"}], [{"name":"北京",  
          {"name":"长治"}], [{"name":"北京",{"name":"  
          重庆"}], [{"name":"北京",{"name":"长沙"}], [{"  
          name":"北京",{"name":"成都"}], [{"name":"北京",  
          {"name":"常州"}], [{"name":"北京",{"name":"  
          丹东"}], [{"name":"北京",{"name":"大连"}], [{"  
          name":"北京",{"name":"东营"}], [{"name":"北京",  
          {"name":"延安"}], [{"name":"北京",{"name":"  
          福州"}], [{"name":"北京",{"name":"海口"}], [{"  
          name":"北京",{"name":"呼和浩特"}], [{"name":"  
          北京",{"name":"合肥"}], [{"name":"北京",{"name":
```


也可以加上事件

```
<ec-series data-uri="./data.json"></ec-series>  
  
<button> Refresh </button>  
  
<script>  
    $('button').on('click', function() {  
        $('ec-series').trigger('refresh');  
    });  
</script>
```

用起来就像原生节点一样，隐藏了组件细节

并且在操作上与原生api保持一致

可搭配jQuery或其他方案使用

Flipper有什麼特性？

兼容性

- ◆ All Modern Browsers
- ◆ IE8+ ，已可应付于实际的业务场景
- ◆ gzip后8k ，可于mobile上使用

易用性

- ◆ 写上标签，等于使用服务。就跟我们写 `<video>` 标签一样
- ◆ 使用者不需关注组件内部实践细节。可搭配任何外部方案使用。就跟我们用一般的html一样
- ◆ 未来组件的升级，或技术更换，完全是透明的

整合性

- ◆ 可原生与AMD、CMD 等模块规范整合
- ◆ 可自行配置渲染引擎。取代原有的HTML注入
- ◆ 提供工程化的打包工具，整合既有流程

扩展性

- ◆ Web Component只是个容器，Flipper也是。
- ◆ 开发者可以选择任何自己熟悉的技术。并透过Flipper注册成为一个Web Component

隔离性

- ◆ 透过module loader进行JS的隔离。
- ◆ 透过CSS预编译避免CSS对外的污染。。
- ◆ 可透过配置决定是否使用shadow dom。进行完整的样式隔离

性能

- ◆ 调试过程中使用html import。上线前透过gulp工具打包于合并组件成单一的js与css
- ◆ 减少安全性的顾虑，并提高载入性能
- ◆ 组件档案亦可内置app中，进行本地存取

没使用的标准

- ◆ HTML Import仅做调试用。线上打包后仍旧使用requirejs或sea.js等loader
- ◆ ShdowDom预设关闭，由组件开发者透过配置决定是否使用

为什么不用Polymer

- ◆ Polymer是一个完整且强大的组件方案，但灵活性相对少了点
- ◆ 我们希望任何组件方案开发出来的组件，都能选择透过web-component方式输出给后端用
- ◆ Polymer的浏览器兼容性与体积，不太符合国内互联网产业的要求

“Flipper做的是服务化，而不是组件方案。”

“优秀的组件方案太多了，未来只会更多”

“之前是angular，今年是react，未来会是什么？”

“作为前端服务提供者的前端，透过Flipper，将不同的技术转换成统一的标准”

“而未来的技术迭迨跟升级，就是完全透明的”

“就跟我们今天调一个restful接口一样”

“解决一个小问题，把它做好，就够了”

使用场景

React的组件

- ◆ 使用react开发了一个组件。
- ◆ 给前端用的时候，直接使用react。
- ◆ 给后端用的时候，透过flipper输出成web component

其他方案的组件

- ◆ flipper本身是个统一的组件注册机制
- ◆ 内部可以用任何的技术方案。angular, backbone, ember, jquery plugin, 都可以
- ◆ 当透过web components输出，使用者只会看到一个标签

如何使用

直接写在页面上

```
<bootstrap-tab> <!-- 使用bootstrap 做的 tab组件 -->

  <div class="content">
    <!-- 使用react 做的 form组件 -->
    <react-form></react-form>
  </div>

  <div class="content">
    <!-- 使用jQuery 做的 calendar 组件 -->
    <jquery-claendar></jquery-claendar>
  </div>

  <div class="content">
    <!-- 使用angular 做的 todolist 组件-->
    <angular-todolist></angular-todolist>
  </div>

</bootstrap-tab>
```


或是用在angular里

```
<div ng-controller="TodoListController as todoList">
  <ul class="unstyled">
    <li ng-repeat="todo in todoList.todos">
      <my-todolist class="done-{{todo.done}}">
        {{todo.text}}
      </my-todolist>
    </li>
  </ul>
</div>
```

当然也可以用在其他MV*方案

“因为是标准化的组件，可以搭配任何方案使用”

“只要最终这个标签被写到页面上，就会透过
Flipper转交到实际的组件工厂中”

不同的方案共处一室

“邪魔歪道？”

“效能很差？载入无效资源？”

关注点分离后的解耦

- ◆ 不局限组件开发者的怎么去开发一个组件
- ◆ 不局限组件使用者的怎么去使用一个组件
- ◆ 透过标准化的web component对接
- ◆ 在大型架构中，先求代码结构清晰，再做效能调优。会发现事半功倍。

“没有技术可以历久不衰，永不更换”

“但是我们可以从统一的标准层，隐藏技术细节”

以 `<video>` 来举例

- ◆ 这个标签背后做了什么事情，背后用了什么技术。使用者通常不关注
- ◆ 使用者只需在页面上哪个位置需要影片，有哪些API可以控制
- ◆ 背后升级的时候，使用者完全无感
- ◆ 可以搭配任何我们已知的框架使用，例如ember、angular、或是react来用

“走在标准上，扩充业务组件服务”

“提高公司整体产品生产力”

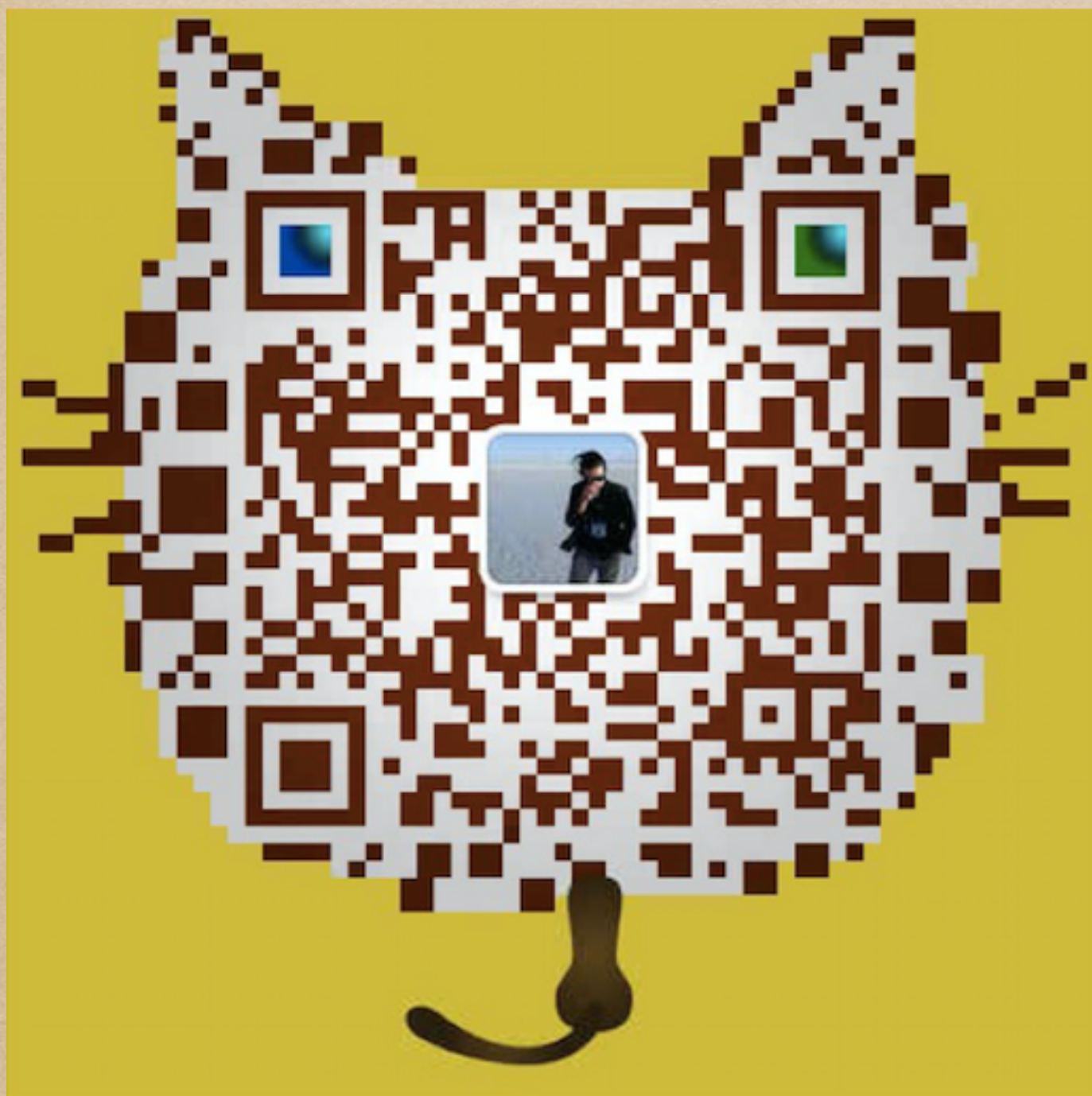
“下一代的前端不是要会最新最好的技术”

“而是能把最新最好的技术沉淀成实战服务，
简单的让他人使用”

“服务化的背后是标准化”

Q & A

即将开源，徵求合作夥伴



Thank You