

Hi!
Let's get started.

HTML5 && OTT WebApp

为 Web 应用而生

关键特性

DOM/CSS

HTML 传统的显示、交互方式

Canvas

2D 图形接口，位图、矢量图形原语

WebGL

OpenGL ES 2.0 兼容 3D 图形接口

Application Cache

离线应用程序

IndexedDB

NoSQL 应用数据存储，大容量缓存

FileSystem API*

可以使用 IndexedDB 模拟实现

*Storage

各种不同功能的存储空间

PIXI、**CAAT** 提供了基于多 **Backend** 的一致性接口

Cocos2d.js 提供了兼容原始 **Cocos2d** 接口的 **API**

别人都是怎么做的？



iOS 上的魔法

```
<meta name="apple-mobile-web-app-capable" content="yes">  
<meta name="apple-mobile-web-app-status-bar-style" content="black">  
<meta name="format-detection" content="telephone=no">
```

UIWebView 会对使用了特殊 **Meta Tag** 的 WebApp 进行优化，使其更加类似于原生应用程序

当前 Tag 的标记仅仅用来控制浏览器的外观，使 Web 页面在全屏模式下更加符合 iOS 系统的设计风格

通过 Meta Tag 通知浏览器进行针对性的优化何尝不是一种很好的方式



用你最熟悉的方式操作 **DOM/CSS**
将 **Native UI API** 的设计风格带入 **WebApp**
更提供了 **Angular.JS** 的绑定

Famo.us

```
Engine  
Context  
Renderable  
  + Surface 映射到 <div> 元素  
Modifier  
  + StateModifier  
Layout  
  + HeaderFooterLayout  
  + GridLayout  
  + FlexibleLayout  
  + SequentialLayout
```

```
var Engine = require('famous/core/Engine');  
var Surface = require('famous/core/Surface');  
  
var mainContext = Engine.createContext();  
  
var firstSurface = new Surface({  
  content: '<h3>Hi!</h3>\n<p>I\'m a surface!\n<br>I live inside a context.</p>\n<p>You can add <b>HTML</b>\ncontent to me and style me with<br><b>CSS!</b></p>',  
  size: [200, 200],  
  properties: {  
    backgroundColor: 'rgb(240, 238, 233)',  
    textAlign: 'center',  
    padding: '5px',  
    border: '2px solid rgb(210, 208, 203)',  
    marginTop: '50px',  
    marginLeft: '50px'  
  }  
});  
  
mainContext.add(firstSurface);
```



FASTER HTML5

with **CocoonJS**

特殊运行时环境，提供

HTML5 API 级运行时接口

专为全屏 **Game** 优化

Canvas+

JavaScriptCore + Cordova

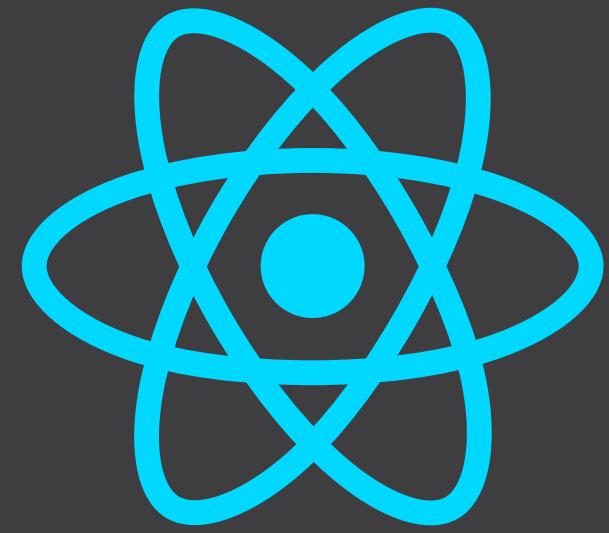
HTML5 API subset <Canvas and WebGL ...>

WebView+

Crosswalk + Cordova

WebView

Android WebView



React.**Native**

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

特殊运行时环境提供 **Library** 级一致性接口

Facebook 的 **React Library** 提供了一整套 **MVC UI** 框架，在此基础上 **React.Native** 通过在 **Native** 层实现一整套一致性接口，将基于 **React** 的应用程序运行在原生系统上 在 **JavaScript** 中用 **React** 对象抽象操作系统原生的 **UI** 组件，代替 **DOM** 元素来渲染，如：以 **View** 取代 `<div>`，以 **Image** 替代 `` 等

Asm.js && Emscripten



C/C++

将 **OpenGL ES API** 映射为 **WebGL**

各种牛逼的功能

还能搞出更多花样么？！

未来 Github 上还会出现什么？

基于 **Canvas**、**WebGL** 的 **UI Framework**

Backbone、**Angular.js**、**Ember.js** 将能使用 **DOM/CSS** 之外的展示技术

基于 **HTML5** 的 **Android**、**iOS Framework**

针对 **WebApp** 特性优化的 **Browser** 分支版本

...

```
console.log('Thanks!')
```