# HTML|HTML5

为兼容XHTML和其他浏览器

<script type=”text/javascript”>

//<![CDATA[

//]]>

</script>

# CSS|CSS3

### 绝对定位

绝对定位的元素的位置相对于最近的已定位祖先元素，如果元素没有已定位的祖先元素，那么它的位置相对于最初的包含块。

# Bootstrap

<http://v3.bootcss.com/css/>

* 脚手架
* css
* 组件
* JS插件
* 定制

问题：<meta name="viewport" content="width=device-width,initial-sc ale=1.0,maximum-scale=1.0,user-scalabel=no">

各参数什么意思？

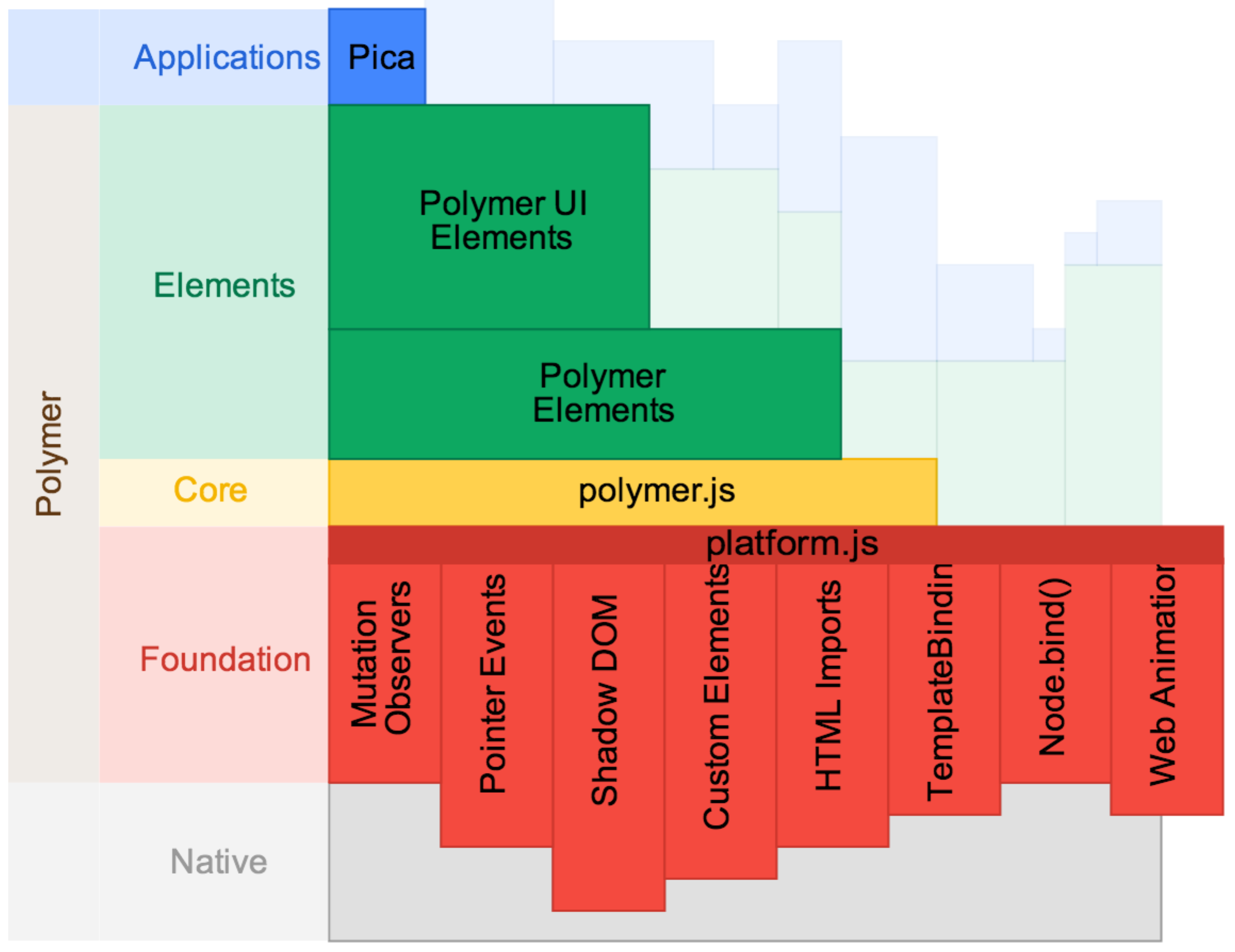
什么是media query？

# Polymer

## 概述

Polymer基于Shadow DOM、Custom Elements、MDV等最新浏览器特性。

代表了下一代Web框架的方向：一切皆组件、尽量减少代码量、尽量减少框架限制。



Polymer由以下几层组成：

* 基础层（Foundation）——platform.js：基本构建块，其中大部分API最终将成为原生浏览器API。
* 核心层（Core）——polymer.js：基础层实现的辅助工具。
* 元素层（Elements）：包括构建于核心层之上的UI以及非UI组件。

## 基础层（platform.js）

其中，基础层使用了以下技术：

1. [DOM Mutation Oberservers](http://dom.spec.whatwg.org/" \l "mutation-observers" \t "_blank)和 [Object.observe()](http://updates.html5rocks.com/2012/11/Respond-to-change-with-Object-observe" \t "_blank)：用于监视DOM元素和简单JavaScript对象的改变。该功能可能会在ECMAScript 7中正式标准化。
2. [Pointer Events](http://www.polymer-project.org/platform/pointer-events.html" \t "_blank) ：在所有的平台上以同样的方式处理鼠标和触摸操作。
3. [Shadow DOM](http://www.polymer-project.org/platform/shadow-dom.html" \t "_blank)：将结构和样式封装在元素内（比如定制元素）。
4. [Custom Elements](http://www.polymer-project.org/platform/custom-elements.html" \t "_blank)：定义自己的HTML5元素。自定义元素的名字中必须包括一个破折号，它的作用类似于命名空间，为了将其与标准元素区分开来。
5. [HTML Imports](http://www.polymer-project.org/platform/html-imports.html" \t "_blank)：封装自定义元素，包中包括HTML、CSS、JavaScript元素。
6. [Model-Driven Views](http://www.polymer-project.org/platform/mdv.html" \t "_blank)（MDV）：直接在HTML中实现数据绑定。仍没有标准化的计划。
7. [Web Animations](http://www.polymer-project.org/platform/web-animations.html" \t "_blank)：统一Web动画实现API。

以上第3-5个API都是[Web Components](http://www.w3.org/TR/components-intro/" \t "_blank)的一部分。

## Getting started

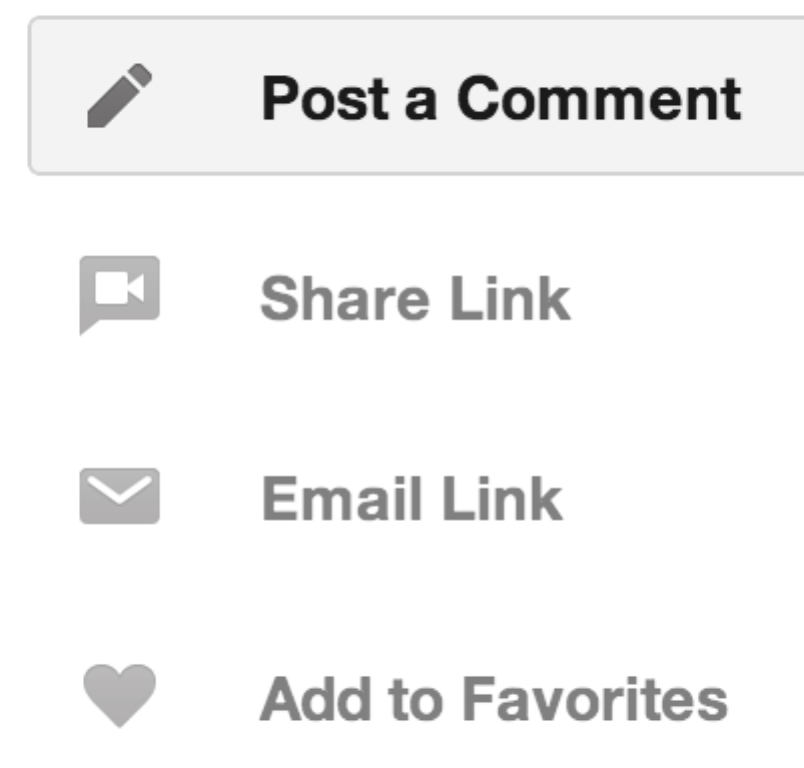
下载源码三种方法：

1. 直接下载zip压缩包，里面包含所有文件。
2. install using Bower，只包含polymer.min.js和platform.min.js编译文件。
3. 运行./pull-all.sh，下载一些库，还可以更新。

测试环境：

将压缩包解压后，目录名为polymer，放到tomcat/webapps中，启动tomcat，访问

<http://localhost:8080/polymer/toolkit-ui/workbench/menu.html>



## Core

处于效率考虑，如果<polymer-element>在document外创建，则不会创建Shadow DOM、事件监听、属性监听。与<img>和<video>行为类似。

Polymer element在以下情况会完整创建：

1. 在document中创建；
2. 被enteredView回调；
3. 在另个元素的shadowRoot中创建；

通过设置alwaysPrepare为true，在任何情况下都创建

Polymer(‘element’, { alwaysPrepare: true});

### polymer attribute

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| name | required | 元素名称，必须包含"-". |
| attributes | optional | 发布属性 |
| extends | optional | 继承元素. |
| noscript | optional | 简单元素，不需要调用Polymer()方法 |
| lightdom | optional | 创建light dom，默认创建shadow dom，  这是测试中的特性，在其他shadow dom中使用lightdom，可能导致绑定错误 |
| constructor | optional | 构造函数，设置该函数则可通过new 创建元素(e.g. var tagName = new TagName()). |

### 默认属性

<polymer-element name="tag-name" class="active" mycustomattr> </polymer-element>

每个tag-name实例都包含mycustomattr属性。

### 其他创建元素的方式

1. 引入script文件

<polymer-element name=”tag-name”>

<template>…</template>

<script src=”path/to/tagname.js”></script>

</polymer-element>

1. 在元素定义之前设置script

<script src=”path/to/tagname.js”></script>

<polymer-element name=”tag-name”>

<template>…</template>

</polymer-element>

1. 没有script的简单元素

<polymer-element name=”tag-name” constructor=”TagName” noscript>

<template>

</template>

</polymer-element>

### 发布公共属性和方法

可通过Polymer()方法的第二个参数，设置元素的属性和方法，第二个参数用于定义元素模型。

<polymer-element name=”tag-name”>

<template></template>

<script>

Polymer(‘tag-name’, {

//定义message属性和foo方法

message: “Hello”,

foo:function(){}

});

</script>

</polymer-element>

this指代当前元素自身，this.localName = “tag-name”

this.$.id=通过id找到节点对象

### 初始化

最好在ready()方法中设置初始值，而不是在prototype中，否则可能导致同一元素多个实例间状态不同。

Polymer('x-foo', {

ready: function() {

this.list = [];

this.person = {};

}

});

而不是

Polymer('x-foo', {

list: [],

person: {}

});

### 创建私有或静态变量

采用匿名方法调用

<polymer-element name=”tag-name”>

<template></template>

<script>

(function(){

var foo\_ = new Foo();

Polymer(‘tag-name’, {

get foo() { return foo\_;}

});

})();

</script>

</polymer-element>

### 元素生命周期方法

Polymer('tag-name', {

created: function() { ... },

ready: function() { ... },

enteredView: function () { ... },

leftView: function() { ... },

attributeChanged: function(attrName, oldVal, newVal) {

//var newVal = this.getAttribute(attrName);

console.log(attrName, 'old: ' + oldVal, 'new:', newVal);

},

});

|  |  |  |
| --- | --- | --- |
| Spec | Polymer | Called when |
| createdCallback | created | 创建元素实例时 |
| - | ready | <polymer-element> 完全准备好时 (比如 Shadow DOM 创建完成, 设置好属性观察, 事件绑定完成等) |
| enteredViewCallback | enteredView | 实例加入document时 |
| leftViewCallback | leftView | 实例从document删除时 |
| attributeChangedCallback | attributeChanged | 增加、删除、更新属性时 |

### WebComponentsReady

polyfill解析元素定义和创建元素是异步的。如果在元素创建前获取它，则会返回HTMLUnknownElement，因此应在WebComponentsReady事件中获取元素。

<head>

<link rel=”import” href=”path/to/x-foo.html”>

</head>

<body>

<x-foo></x-foo>

<script>

window.addEventListener(“WebComponentsReady”, function(e){

var xFoo = document.querySelector(“x-foo”);

xFoo.barProperty = “baz”;

});

</script>

</body>

### 发布属性

两种方式发布属性：

1. <polymer-element>的<attributes>属性（推荐）；
2. 在prototype中用publish方法。

attributes中的属性默认设为null。可通过prototype设置默认值。

<polymer-element name="x-foo" attributes="foo">

<script>

Polymer('x-foo'); // x-foo has a foo property with null value.

</script>

</polymer-element>

<polymer-element name="x-foo" attributes="bar">

<script>

Polymer('x-foo', {

// x-foo has a bar property with default value false.

bar: false

});

</script>

</polymer-element>

Polymer默认会将属性类型转换为默认值类型，bar为boolean型。

<!-- Same, but using the alternate "publish" object. -->

<polymer-element name="x-foo">

<script>

Polymer('x-foo', {

publish: {

bar: false

}

});

</script>

</polymer-element>

### 数据绑定

Polymer的data-binding通过以下技术实现：

1. HTML template元素<template>；
2. template 绑定，利用<template>如何创建实例片段；
3. Node.bind()描述DOM绑定到data values；
4. Object.observe()；
5. DOM Mutation Observers。

通过数据绑定，修改model的值能直接反射到DOM上，修改DOM的值能直接赋给model。

<script src="platform.min.js"></script>

<ul> //repeat遍历salutations中的元素

<template id="greeting" repeat="{{ salutations }}">

<li>{{ what }}: <input type="text" value="{{ who }}"></li>

</template>

</ul>

<script>

var t = document.getElementById('greeting');

var model = { //创建model

salutations: [

{ what: 'Hello', who: 'World' },

{ what: 'GoodBye', who: 'DOM APIs' },

{ what: 'Hello', who: 'Declarative' },

{ what: 'GoodBye', who: 'Imperative' }

] };

t.model = model;

// Needed to detect model changes if Object.observe() is not available.

Platform.performMicrotaskCheckpoint();

</script>

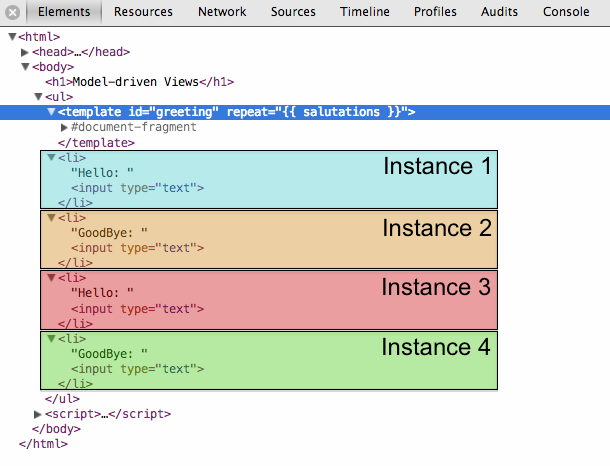
</body>

<template>是新元素，浏览器正在逐步实现它。Chrome Inspector可以查看template的内容。

如果没有引入<script src="platform.min.js"></script>，则内容如下：



否则内容如下：



Polymer能绑定对象和数组。

<polymer-element name="name-tag" attributes="person">

<template>

Hello! <span style="color:{{person.nameColor}}">{{person.name}}</span>

</template>

<script>

Polymer('name-tag', {

ready: function() {

this.person = {

name: "Scott",

nameColor: "orange"

}

}

});

</script>

</polymer-element>

<polymer-element name="visitor-creds">

<template>

<name-tag person="{{person}}"></name-tag>

</template>

<script>

Polymer('visitor-creds', {

ready: function() {

this.person = {

name: "Scott2",

nameColor: "red"

}

}

});

</script>

</polymer-element>

visitor-creds和name-tag间共享person对象

### 声明事件映射

<polymer-element name="g-cool" on-keypress="{{keypressHandler}}">

<template>

<button on-click="{{buttonClick}}"></button>

</template>

<script>

Polymer('g-cool', {

keypressHandler: function(event, detail, sender) { ...},

buttonClick: function(event, detail, sender) { ... }

});

</script>

</polymer-element>

keypress绑定keypressHandler，button的click绑定buttonClick方法。

事件处理函数能有以下参数：

1. inEvent：标准事件对象；
2. inDetail：inEvent.detail；
3. inSender：声明处理函数的节点引用，不同于inEvent.target和inEvent.currentTarget。

### 属性观察

Polymer利用polyfill（observe-js）定期调用Platform.flush()检查数据变化。

何时直接调用Platform.flush()

1. 属性变化导致CSS应用到元素；
2. 拖动滑块时另一个控件显示值。

#### Changed watchers

通过实现propertyNameChanged函数

<polymer-element name="g-cool" attributes="better best">

<script>

Polymer('g-cool', {

plain: '',

best: '',

betterChanged:function(oldValue, newValue) {},

bestChanged: function(oldValue, newValue) {}

});

</script>

</polymer-element>

betterChanged和bestChanged监听better和best属性的改变事件。

#### Property Observer

observe方法定义一个或多个property和observer映射关系。如果有observe则不会再调用propertyNameChanged。

Polymer('x-element', {

foo: '',

bar: '',

observe: {

foo: 'validate',

bar: 'validate'

},

ready: function() {

this.foo = 'bar';

this.bar = 'foo';

},

validate: function(oldValue, newValue) {},

});

当foo和bar改变时，调用validate()

Polymer('x-element', {

observe: {

'a.b.c': 'validateSubPath'

},

ready: function() {

this.a = {

b: {

c: 'exists'

}

};

},

validateSubPath: function(oldValue, newValue) {

var value = Path.get('a.b.c').getValueFrom(this);

// oldValue == undefined

// newValue == value == this.a.b.c === 'exists'

}

});

### 调用事件

<polymer-element name="ouch-button">

<template>

<button on-click="{{onClick}}">Send hurt</button>

</template>

<script>

Polymer('ouch-button', {

onClick: function() {

this.fire('ouch', {msg: 'That hurt!'}); // fire(inType, inDetail, inToNode)

}

});

</script>

</polymer-element>

<ouch-button></ouch-button>

<script>

document.querySelector('ouch-button').addEventListener('ouch', function(e) {

console.log(e.type, e.detail.msg); // "ouch" "That hurt!"

});

</script>

### 表达式

内嵌表达式

以{{}}括起来，可以为简单表达式或对象

只包含简单值和关系，不要写复杂逻辑；

不能通过全局变量（如window）访问

|  |  |  |
| --- | --- | --- |
| Feature | Example | Explanation |
| Identifiers& paths | foo, foo.bar.baz | 相对于局部model，当model改变时会相应变化 |
| Logical not operator | ! |  |
| Unary operators | +foo, -bar | 转为Number |
| Binary operators | foo + bar,foo - bar,foo \* bar | 支持: +, -, \*, /, % |
| Comparators | foo < bar,foo != bar,foo == bar | 支持: <, >, <=, >=, ==, !=, ===, !== |
| Logical comparators | foo && bar || baz | 支持: ||, && |
| Ternary operator | a ? b : c |  |
| Grouping (parenthesis) | (a + b) \* (c + d) |  |
| Literal values | numbers, strings,null, undefined | 不支持转义字符和非十进制数字。 |
| Array & Object initializers | [foo, 1],{id: 1, foo: bar} |  |

例如

<div>Jill has {{ daughter.children.length + son.children.length }} grandchildren</div>

结果为<div> Jill has 100 grandchildren</div>

名字空间，用于引用外部model的值。

<template repeat="{{ user in users }}">

{{ user.name }}

<template repeat="{{ file in user.files }}">

{{ user.name }} owners {{ file.name }}

</template>

</template>

如果<template>中包含子<template>，则所有祖先都可见。

<template bind="{{ foo as foo }}">

<!-- foo.\* available -->

<template bind="{{ foo.bar as bar }}">

<!-- foo.\* & bar.\* available -->

<template bind="{{ bar.bat }}">

<!-- only properties of bat are available -->

<template bind="{{ boo as bot }}">

<!-- bot.\* and properties of bat are available. NOT foo.\* or bar.\* -->

</template>

</template>

</template>

</template>

### 继承/重置外部样式

Polymer('x-foo', {

applyAuthorStyles: true,

resetStyleInheritance: true

...

});

### 避免FOUC

FOUC(a flash of unstyled content)在没加载完CSS文件之前，显示没有样式的web页面的问题。由于在所有信息获取到之前web browser engine就渲染页面，加载完css才纠正样式。

可为元素添加polymer-veiled类，隐藏该元素

<x-foo class="polymer-veiled">If you see me, elements are upgraded!</x-foo> <div class="polymer-veiled"></div>

或者通过Polymer.veiledElements，其中的元素在启动时会自动设为polymer-veiled类。

Polymer.veiledElements = ['x-foo', 'div'];

|  |  |
| --- | --- |
| Class name | Behavior when applied to an element |
| polymer-veiled | 设置元素的 opacity: 0. |
| polymer-unveil | 设置元素的opacity: 1. |

### 覆盖默认行为

body默认在Polymer.veiledElement数组中。当WebComponentsReady调用时，Polymer移除polymer-veiled类，添加polymer-unveil。为覆盖这种行为（页面初始化时不可见），可设Polymer.veiledElements = null;

### 启动后显示元素

为避免FOUC，加载页面时为元素增加polymer-veiled类，当元素需要显示时调用Polymer.unveilElement()。

element.classList.add('polymer-veiled');

// ... some time late ...

Polymer.unveilElements();

### Polyfill样式指令

@polyfill指令用于将原生CSS选择器替换成polyfill。

在CSS样式上面的注释内设置@polyfill，@polyfill后面的内容（.bar）指明下面的样式用polyfill处理。

/\* @polyfill .bar \*/

::content .bar {

color: red;

}

/\* @polyfill .container > \* \*/

::content > \* {

border: 1px solid black;

}

### Shadow DOM

Shadow DOM通过Shadow root下的DOM子树封装标签，为元素样式提供作用域。

Shadow DOM允许单节点表示3种子树：light DOM，shadown DOM和composed DOM。

Polymer消息传递

有两种方式：

1. 数据绑定（data binding）

优点：不需要写代码；双向绑定

缺点：只在Poylmer element内部或之间有效

1. 变化观察者（changed watchers）

优点：不需要发布属性；允许element观察属性变化

缺点：只在Poylmer element内部有效；需要写代码

1. 常规事件（custom event）

优点：在Poylmer element内部或之间都有效

缺点：少量代码

## 注意事项

created：控件初始化时调用，此时属性还没有赋值

ready：控件初始化完成时调用，此时属性已赋值，但对于对象或数组，无法正确赋值。

xxxChanged：属性值变更时调用。

注意attributes属性有s

<polymer-element name="mt-radio" **attributes**="name value checked">

### 其他

**问题：自定义标签不能简写**

标签格式为<mt-radio></mt-radio>，而不能为<mt-radio/>。

**问题：控件初始化时，值无法绑定**

<script>

var rs=[];

</script>

<mt\_checkbox id=” checkboxgroup1” rs=”{{}}”/>

以上写法，rs是无法绑定到控件中的。

必须在WebComponentReady时，手动绑定rs

document.addEventListener('WebComponentsReady', function() {

var checkboxgroup1 = document.getElementById('checkboxgroup1');

checkboxgroup1.rs = rs;

});

### 控件特性

#### checkbox

必须给checked属性赋初始值，可正确显示。在初始化时，checkbox只要有checked属性，就是选中，不管其值为多少。

#### 控件设置id

**问题：id如果用local，在创建多个控件时会冲突**

<template>

<input type=”checkbox” id=”local”/><label for=”local”></label>

</template>

因此须采用不同id。

**问题：如果id直接设为{{id}}，则label无法绑定前面的input**

因为<mt-checkbox id=”xxxx”/>中id已经出现过，在template中重复出现。

解决方法：可以给id加个后缀。

<template>

<input type=”checkbox” id=”{{id}}**\_checkbox**”/><label for=” {{id}}**\_checkbox**”></label>

</template>

 Platform.performMicrotaskCheckpoint();什么意思？

为什么id修改了，还可以通过this.$.id访问到？

初始化

# JavaScript

JavaScript包括ECMAScript，DOM，BOM。

## ECMAScript

由ECMA-262定义，提供核心语言功能。

严格模式，在顶部添加编译指标(pragma)”use strict”，对不安全行为抛出异常

function doSomething(){

“use strict”;

…

}

数据类型：

基本类型：Undefined、Null、Boolean、Number、String

引用类型：Object

1. 基本类型在内存中占据固定大小的空间，**保存在栈内存**中，值传递；
2. 引用类型是指针，值是对象，**保存在堆内存中**，**引用传递**
3. 基本类型用typeof，引用类型用instanceof

var message; //未赋值时=undefined

typeof(message); //返回undefined

typeof(value) ; //未声明，返回undefined

var car = null;

typeof(car); //返回null

typeof(null); //返回object

var message = “hello”;

var messageboolean = Boolean(message); //Boolean()函数转换为Boolean值

如果值为0、-0、null、""、false、undefined 或者 NaN， 则转换为false，否则为true。

with(a){ //所有with中的对象前面限定为a

href //在href前面加上a.

}

### arguments

函数内部arguments为参数数组

function say(){

alert(arguments.length);

}

function add(num1, num2){

arugments[1]=10; //num2的值会同步为10

num1 =20; //arguments[0]的值不会同步为20

}

修改arguments的值会同步对应的命名参数，但修改命名参数不会影响arguments的

arguemtns的长度由传入参数个数决定，不是命名参数个数决定

没有传递值的命名参数赋为undefined

如 add(10)则arguemtns.length等于1，num2为undefined

ECMAScript没有函数签名，函数参数以数组形式传递，用arguments访问。不能重载，后面的定义会覆盖前面的。

函数无需返回值，默认返回undefined

### 执行环境及作用域

执行环境分为全局环境和函数执行环境；

进入新执行环境会创建一个作用域链

局部环境可访问函数作用域、父环境、全局环境的变量

解析标识符是沿着作用域链的前端开始，逐级向后解析

当前作用域的对象在作用域最前端。全局执行环境的变量在作用域链最后。

手动解除全局环境的引用

try-catch的catch块：创建新变量（抛出异常）

with语句：将制定对象添加到作用域链中

会延长作用域链，在前端临时增加一个变量对象

JS没有块级作用域

### Object对象

两种创建方式：

var person = new Object();

var person = { name : ‘dwt’}; //对象字面符

persion[‘name’]

persion.name

### Array对象

两种创建方法：

var colors = new Array();

var colors = [‘red’, ‘blue’]; //数组字面符

通过数组length属性动态变化，可以增加或删除项

## 文档对象模型DOM

提供访问和操作网页内容的方法和接口。

DOM1：DOM core和DOM HTML。

DOM2：DOM Views、DOM Events、DOM Style、DOM Traversal and Range。

绝不要使用在文档加载之后使用 document.write()。这会覆盖该文档。

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p>My First Paragraph.</p>

<button onclick="myFunction()">点击这里</button>

<script>

function myFunction() {

document.write("糟糕！文档消失了。");

}

</script>

</body>

</html>

## 浏览器对象模型BOM

提供与浏览器交互的方法和接口。

## <script>

6个属性：

charset：通过src引用文件的字符集，大多数浏览器会忽略此值。

asysc：在不妨碍页面其他操作的情况下，立刻下载外部脚本。不能保证脚本按照它们在页面中的顺序执行。

defer：当文档完全解析后执行脚本。

language：已废弃。

src：引入外部脚本。可跨域引用。

type：脚本的内容类型，默认为text/javascript。

浏览器会先解析没有defer属性的<script>代码，再解析后面的内容。为避免页面出现空白，应把<script>放在页面最后</body>之前。

<noscript>在不支持脚本的浏览器中显示其内容。

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量（在函数外声明的）会在页面关闭后被删除。

如果把值赋给尚未声明的变量，该变量将被自动作为全局变量声明。

1. 访问对象属性方式

objectName.propertyName

objectName[‘propertyName’]

for(var i in objectName){

objectName[i]

}