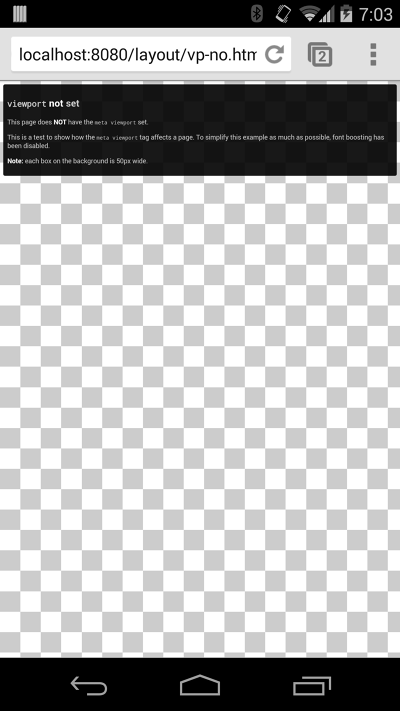
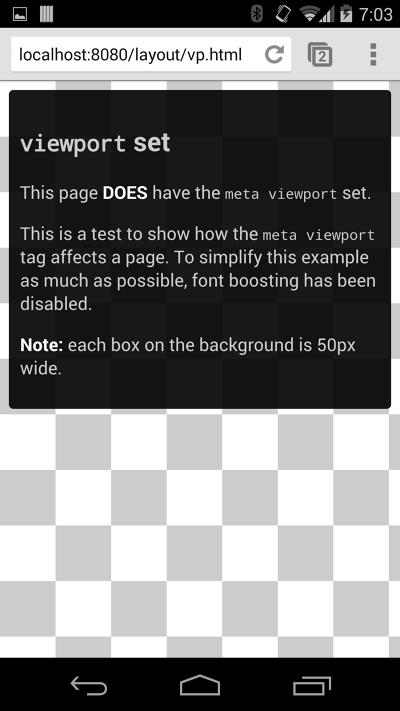
△需要巩固 ※不曾了解，需要知道 ☺仅为了解

## Responsive design

**△添加meta标签**

<meta name="viewport" content="width=device-width, initial-scale=1">

是否添加的区别如下（左边没有添加）：

**△为栅格添加media query样式**

@media screen and (max-width: 48rem) {

.container .col {

width: 95%;

}

}

@media (query) {  
  /\* CSS Rules used when query matches \*/  
}

| 属性 | 结果 |
| --- | --- |
| min-width | 当任意浏览器宽度大于查询中定义的值时适用的规则。 |
| max-width | 当任意浏览器宽度小于查询中定义的值时适用的规则。 |
| min-height | 当任意浏览器高度大于查询中定义的值时适用的规则。 |
| max-height | 当任意浏览器高度小于查询中定义的值时适用的规则。 |
| orientation=portrait | 高度大于或等于宽度的任意浏览器适用的规则。 |
| orientation=landscape | 宽度大于高度的任意浏览器适用的规则。 |

不推荐使用min-device-width去获取设备宽度，因为结果可能不准确。

**※添加浏览器兼容的flex**

.container {

display: -webkit-box; */\* OLD - iOS 6-, Safari 3.1-6 \*/*

display: -ms-flexbox; */\* TWEENER - IE 10 \*/*

display: flex; */\* NEW, Spec - Firefox, Chrome, Opera \*/*

background: *#eee;*

overflow: auto;

}

.container .col {

flex: 1;

padding: 1rem;

}

**※添加浏览器兼容且使用media query的flex**

@media screen and (max-width: 48rem) {

.container {

display: -webkit-box;

display: -ms-flexbox;

display: flex;

flex-flow: column;

}

}

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

使用modernizr-custom.js检测浏览器是否支持flex，并添加.flex/.no-flex样式。

## Responsive images

**※使用max-width让图片不会溢出它的父容器**

img {

max-width: 100%; /\* or use 100vw \*/

}

**※根据实时宽度通过media query改变图片尺寸**

@media screen and (max-width: 700px) {

img*#sfo {*

max-width: 90vw;

width: 90vw;

}

}

**※添加srcset属性让浏览器基于视口宽度加载正确的图片**

srcset="images/sfo-1600\_large.jpg 1600w,

images/sfo-1000\_large.jpg 1000w,

images/sfo-800\_medium.jpg 800w,

images/sfo-500\_small.jpg 500w"

如果你需要用CSS修改图片的尺寸，使用srcset仍然只会按照原始尺寸获取对应的源图片，因此需要一个新的属性sizes="50vw" 去告诉浏览器即将显示的图片大小

<img id="sfo" src="images/sfo-500\_small.jpg" srcset="images/sfo-1600\_large.jpg 1600w, images/sfo-1000\_large.jpg 1000w, images/sfo-800\_medium.jpg 800w, images/sfo-500\_small.jpg 500w" sizes="50vw" alt="View from aircraft window near San Francisco airport">

Media query也可以加入这个属性中

sizes="(max-width: 700px) 90vw, 50vw"

☺该picture元素使我们可以使用source标签定义多个源文件。这与仅使用img带有srcset属性的标签不同，这是因为source标签使我们可以将媒体查询之类的内容添加到每组来源中。可以给浏览器定义图像大小，而不是让浏览器确定图像大小，而是定义每种窗口大小使用的图像。

<figure>

<picture>

<source media="(min-width: 750px)"

srcset="images/horses-1600\_large\_2x.jpg 2x,

images/horses-800\_large\_1x.jpg" />

<source media="(min-width: 500px)"

srcset="images/horses\_medium.jpg" />

<img src="images/horses\_small.jpg" alt="Horses in Hawaii">

</picture>

<figcaption>Horses in Hawaii</figcaption>

</figure>

## Video

**添加视频**

<video controls>  
  <source src="https://storage.googleapis.com/webfundamentals-assets/videos/chrome.webm" type="video/webm">  
  <source src="https://storage.googleapis.com/webfundamentals-assets/videos/chrome.mp4" type="video/mp4">  
  <p>This browser does not support the video element.</p>  
</video>

**添加封面**

<video poster="poster.jpg" ...>  
  ...  
</video>

**正确调整视频大小**

使用 CSS 媒体查询来根据视口尺寸指定元素大小；max-width: 100% 是您的好帮手。

**控制内容的全屏模式**

以全屏模式显示某元素，如视频：

elem.requestFullScreen();

以全屏模式显示整个文档：

document.body.requestFullScreen();

您还可以侦听全屏状态变化：

video.addEventListener("fullscreenchange", handler);

或者，查看元素当前是否处于全屏模式：

console.log("In full screen mode: ", video.displayingFullscreen);

您也可以使用 CSS :fullscreen 伪类来更改全屏模式下元素的显示方式。

为视频添加字幕

<video controls>  
  <source src="https://storage.googleapis.com/webfundamentals-assets/videos/chrome.mp4" type="video/mp4" />  
  <track src="chrome-subtitles-en.vtt" label="English captions"  
         kind="captions" srclang="en" default>  
  <p>This browser does not support the video element.</p>  
</video>

Video 元素属性

| 属性 | 可用性 | 描述 |
| --- | --- | --- |
| src | 所有浏览器。 | 视频的地址（网址）。 |
| poster | 所有浏览器。 | video 元素显示后无需下载视频内容浏览器就可立即显示的图片文件的地址（网址）。 |
| preload | 所有移动浏览器均忽略 preload。 | 提示浏览器：播放前预加载元数据（或某个视频）十分重要。选项包括 none、metadata 或 auto（请参阅[预加载](https://developers.google.cn/web/fundamentals/media/video" \l "preload)部分了解详情）。 |
| autoplay | iPhone 和 Android 设备均不支持；所有桌面版浏览器、iPad 以及 Android 版 Firefox 和 Opera 均支持。 | 尽快开始下载和播放（请参阅[自动播放](https://developers.google.cn/web/fundamentals/media/video" \l "autoplay)部分了解详情）。 |
| loop | 所有浏览器。 | 循环播放视频。 |
| controls | 所有浏览器。 | 显示默认视频控件（播放、暂停等）。 |

## 响应用户界面

与固定宽度的版式相比，自适应设计的主要概念基础是流畅性和比例可调节性。使用相对衡量单位有助于简化版式，并防止无意间创建对视口来说过大的组件。

在顶级 div 上设置 width: 100% 可以确保其横跨视口的宽度，对视口来说既不会太大也不会太小。

div.fullWidth {  
  width: 100%;  
}

**以从小屏幕开始、不断扩展的方式选择主要断点**

<link rel="stylesheet" href="weather.css">  
<link rel="stylesheet" media="(max-width:600px)" href="weather-2-small.css">  
<link rel="stylesheet" media="(min-width:601px)" href="weather-2-large.css">

**优化文本，提高可读性**

传统的可读性理论建议：理想栏目的每一行应该包含 70 到 80 个字符（大约 8 到 10 个英文单词），因此，每次文本块宽度超过 10 个单词时，就应考虑添加断点。

**绝不能完全隐藏内容**

在根据屏幕大小选择要隐藏或显示的内容时请务必谨慎。不要只是因为内容无法适合屏幕而将其隐藏。屏幕大小并非确定用户需求的决定性因素。

## 为网站添加触摸功能

**响应元素状态**

DOM 元素可继承下列任何状态：default、focus、hover 和 active。 要在上述每一种状态下改变 UI，我们需要对下列伪类 :hover、:focus 和 :active 应用样式

**添加事件侦听器**

在 Chrome（版本 55 及更高版本）、Internet Explorer 和 Edge 中，PointerEvents 是建议的自定义手势实现方法。

在其他浏览器中，TouchEvents 和 MouseEvents 是正确的方法。

使用这些事件需要对 DOM 元素调用 addEventListener() 方法，使用的参数为事件名称、回调函数和一个布尔值。布尔值决定是否应在其他元素有机会捕获并解释事件之前或之后捕获事件。（true 表示想要先于其他元素捕获事件。）

// Check if pointer events are supported.  
if (window.PointerEvent) {  
  // Add Pointer Event Listener  
  swipeFrontElement.addEventListener('pointerdown', this.handleGestureStart, true);  
  swipeFrontElement.addEventListener('pointermove', this.handleGestureMove, true);  
  swipeFrontElement.addEventListener('pointerup', this.handleGestureEnd, true);  
  swipeFrontElement.addEventListener('pointercancel', this.handleGestureEnd, true);  
} else {  
  // Add Touch Listener  
  swipeFrontElement.addEventListener('touchstart', this.handleGestureStart, true);  
  swipeFrontElement.addEventListener('touchmove', this.handleGestureMove, true);  
  swipeFrontElement.addEventListener('touchend', this.handleGestureEnd, true);  
  swipeFrontElement.addEventListener('touchcancel', this.handleGestureEnd, true);  
  // Add Mouse Listener  
  swipeFrontElement.addEventListener('mousedown', this.handleGestureStart, true);  
}

this.handleGestureStart = function(evt) {  
  evt.preventDefault();  
  if(evt.touches && evt.touches.length > 1) {  
    return;  
  }  
  // Add the move and end listeners  
  if (window.PointerEvent) {  
    evt.target.setPointerCapture(evt.pointerId);  
  } else {  
    // Add Mouse Listeners  
    document.addEventListener('mousemove', this.handleGestureMove, true);  
    document.addEventListener('mouseup', this.handleGestureEnd, true);  
  }  
  initialTouchPos = getGesturePointFromEvent(evt);  
  swipeFrontElement.style.transition = 'initial';  
}.bind(this);

this.handleGestureMove = function (evt) {  
  evt.preventDefault();  
  if(!initialTouchPos) {  
    return;  
  }  
  lastTouchPos = getGesturePointFromEvent(evt);  
  if(rafPending) {  
    return;  
  }  
  rafPending = true;  
  window. requestAnimationFrame(onAnimFrame);  
}.bind(this);

function getGesturePointFromEvent(evt) {  
    var point = {};  
  
    if(evt.targetTouches) {  
      // Prefer Touch Events  
      point.x = evt.targetTouches[0].clientX;  
      point.y = evt.targetTouches[0].clientY;  
    } else {  
      // Either Mouse event or Pointer Event  
      point.x = evt.clientX;  
      point.y = evt.clientY;  
    }  
  
    return point;  
  }

function onAnimFrame() {  
  if(!rafPending) {  
    return;  
  }  
  
  var differenceInX = initialTouchPos.x - lastTouchPos.x;  
  
  var newXTransform = (currentXPosition - differenceInX)+'px';  
  var transformStyle = 'translateX('+newXTransform+')';  
  swipeFrontElement.style.webkitTransform = transformStyle;  
  swipeFrontElement.style.MozTransform = transformStyle;  
  swipeFrontElement.style.msTransform = transformStyle;  
  swipeFrontElement.style.transform = transformStyle;  
  
  rafPending = false;  
}

## Fetch

基本用法

fetch('examples/animals.json')

.then(validateResponse)

.then(logResult)

.catch(logError);

function validateResponse(response) {

if (!response.ok) {

throw Error(response.statusText);

}

return response;

}

function logResult(result) {

console.log(result);

}

function logError(error) {

console.log('Looks like there was a problem:', error);

}

获取图片

function fetchImage() {

fetch('examples/fetching.jpg')

.then(validateResponse)

.then(readResponseAsBlob)

.then(showImage)

.catch(logError);

}

function readResponseAsBlob(response) {

return response.blob();

}

function showImage(responseAsBlob) {

const imgElem = document.createElement('img');

document.getElementById('img-container').appendChild(imgElem);

const imgUrl = URL.createObjectURL(responseAsBlob);

imgElem.src = imgUrl;

}

注意：该URL对象的 createObjectURL()方法被用于生成表示所述Blob数据URL。这一点很重要。您不能将图像的源直接设置为Blob。必须将Blob转换为数据URL。

获取请求头部的数据而不获取整个响应体

function headRequest() {

fetch('examples/words.txt', {

method: 'HEAD'

})

.then(validateResponse)

.then(readResponseAsText)

.then(logResult)

.catch(logError);

}

function logResult(result) {

console.log(result.headers.get('Content-Length'));

}

Post

function postRequest() {

const formData = new FormData(document.getElementById('msg-form'));

fetch('http://localhost:5000/', { //localhost:5000代表某个api

method: 'POST',

body: formData

})

.then(validateResponse)

.then(readResponseAsText)

.then(showText)

.catch(logError);

}

使用non-cros获取不透明的服务器返回，你无法对返回的数据做显示以外的任何操作

function postRequest() {

const formData = new FormData(document.getElementById('msg-form'));

fetch('http://localhost:5001/', {

method: 'POST',

body: formData,

mode: 'no-cors'

})

.then(logResult)

.catch(logError);

}

发送body为json的请求

function postRequest() {

fetch('http://localhost:5000/', {

method: 'POST',

body: JSON.stringify({ lab: 'fetch', status: 'fun' })

})

.then(validateResponse)

.then(readResponseAsText)

.then(showText)

.catch(logError);

}

发送含有自定属性的请求

function postRequest() {

var myHeaders = new Headers();

myHeaders.append('Content-Type', 'application/json');

myHeaders.append( 'X-CUSTOM', 'hello world');

myHeaders.append( 'Y-Custom',"new");

fetch('http://localhost:5000/', {

method: 'POST',

headers: myHeaders,

body: JSON.stringify({ lab: 'fetch', status: 'fun' })

})

.then(validateResponse)

.then(readResponseAsText)

.then(showText)

.catch(logError);

}

Header interface 使得能够创建和修改请求头，例如Content-Type。但是像 Content-Length因为安全原因是受保护的不能被修改

服务器定义了它所能接受的请求头中自定义的字段，例如本例中接受X-Custom标头，但不接受标Y-Custom头。只要设置了自定义标头，浏览器就会执行预检检查。这意味着浏览器首先向服务器发送OPTIONS请求，以确定服务器允许哪些HTTP方法和标头。如果将服务器配置为接受原始请求的方法和标头，则将其发送，否则将引发错误。

## Accessibility

隐式可聚焦元素

文本字段、按钮和选择列表等内置的交互式 HTML 元素是隐式可聚焦元素，这意味着它们是自动插入到 Tab 键顺序中，并且内置了键盘事件处理，无需开发者进行干预。

语义简介

辅助技术：辅助技术是一个涵盖性术语，泛指有助于任何残障人士完成任务的设备、软件和工具。

功能可见性：功能可见性是指任何提供或给予其用户操作执行机会的对象。功能可见性设计得越好，用途就越明显或越直观。

屏幕阅读器：一种流行的辅助技术是“屏幕阅读器”，视力残障人士可以利用该程序以合成语音朗读屏幕上的文字，从而达到使用计算机的目的。

原生 HTML 中的语义

浏览器可以将 DOM 树转变成无障碍树，因为 DOM 的大部分内容具有隐式语义含义。

可见标签：所有用户都使用它们将含义与元素关联起来；

文本替代项：仅在不需要视觉标签时使用，为任何非文本内容提供文本替代项。

<label>

<input type="checkbox">Receive promotional offers?</input>

</label>

或者

<input id="promo" type="checkbox"></input>

<label for="promo">Receive promotional offers?</label>

图像的替代文本

<img src="/160204193356-01-cat-500.jpg" alt="一只目光汹汹凝视远方的猫">

所有图像都应有 alt 属性，但它们无需都包含文本。 重要的图像应使用描述性替代文本简洁地说明图像内容，而装饰性图像应使用空的 alt 属性，即 alt=""。使用空的 alt 属性就可让屏幕阅读器将图像整个跳过

ARIA 标签和关系

ARIA 是唯一一种可以添加可访问帮助或说明文本的方式。

**aria-label**

aria-label 允许我们指定一个用作可访问标签的字符串。这将替换任何其他原生标记机制，例如 label 元素 —例如，如果 button 同时具有文本内容和 aria-label，将仅使用 aria-label 值。

<button id=”buttonId” aria-label=”menu” class=”btn-primary”></button>

**aria-labelledby**

aria-labelledby 允许我们将 DOM 中另一个元素的 ID 指定为当前元素的标签。相当于借用另一个元素作为自己的aria-label，此元素由另一个元素标记。

值得注意的2点：1. 只有一个标签元素与可标记元素关联，但是 aria-labelledby 可以利用一组 IDREF 从多个元素构建标签。标签将按照 IDREF 的提供顺序串联。

2.您可以使用 aria-labelledby 引用隐藏和不在可访问性树中的元素。 例如，您可以在想要标记的元素旁添加一个隐藏的 span，然后使用 aria-labelledby 引用该元素。

<div aria-labelledby=”buttonId” class=”btn-second”></div>

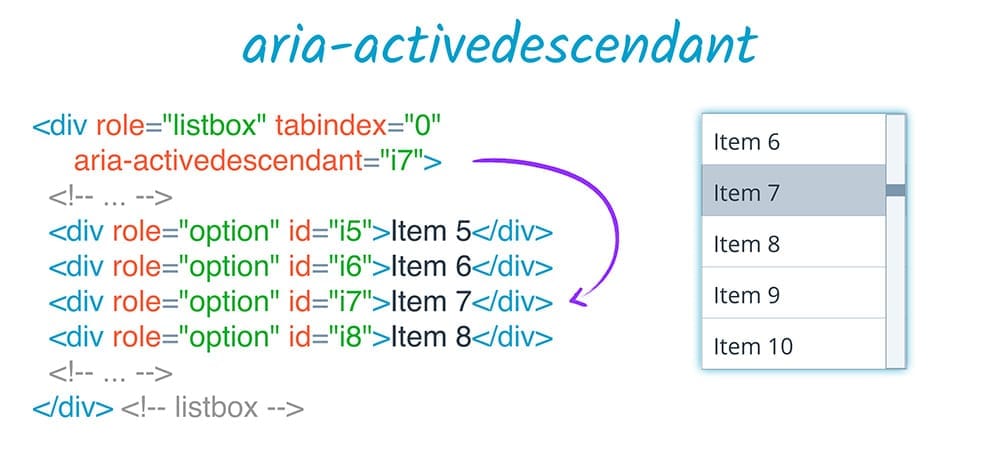
**aria-owns**

使用 aria-owns 将子菜单作为父菜单的子项呈现给屏幕阅读器。

<div aria-owns=”submenu”>car</div>

**aria-activedescendant**

在列表框中，您可能希望将页面焦点停留在列表框容器上，但对当前选中的列表项持续更新列表框的 aria-activedescendant 属性。这样会让当前选定项以焦点项的形式显示给辅助技术。



**aria-describedby**

我们暂时认为它和**aria-labelledby**是一样的

**aria-posinset 和 aria-setsize**

aria-setsize 可以指定实际集大小，aria-posinset 可以指定元素在集中的位置。例如，在一个可能包含 1000 个元素的集中，您可以指定特定元素的 aria-posinset 为 857（即使其在 DOM 中位于首位），然后使用动态 HTML 技术确保用户可以根据需要查看完整列表。

多设备自适应设计

确保始终使用正确的 viewport 元标记

使用自适应网格进行设计

为文本大小之类的项目使用诸如 em 或 rem 的相对单位

对于正确设置移动视口的网站，建议的最小触摸目标大小约为 48 个设备无关像素。

建议为所有文本使用 4.5:1 的 AA（最低）对比度比率



**aria-hidden**

利用 aria-hidden 属性将非视觉隐藏内容排除在辅助技术访问范围之外的机制。如果对元素应用该属性，实际上是将元素及其所有子项从无障碍树中移除。

任何向 DOM 显式隐藏的内容同样不会包含在无障碍树中。 因此，任何 CSS 样式为 visibility: hidden 或 display: none 或者使用 HTML5 hidden 属性的内容同样会向辅助技术用户隐藏。

**aria-live**

aria-live 允许开发者将某个页面部分标记为“活动”

## Progressive Web Apps

在你的页面上添加 service worker 支持

if ('serviceWorker' in navigator) {

window.addEventListener('load', () => {

navigator.serviceWorker.register('service-worker.js')

.then(registration => {

console.log('Service Worker is registered', registration);

})

.catch(err => {

console.error('Registration failed:', err);

});

});

}

添加service worker的逻辑

self.addEventListener('install', event => {

console.log('Service worker installing...');

self.skipWaiting();

});

self.addEventListener('activate', event => {

console.log('Service worker activating...');

});

skipWaiting();跳过等待阶段来立即激活新的service worker

serviceworker的工作范围是它所在的目录为它的scope

使用serviceworker缓存文件

const filesToCache = [

'/',

'style/main.css',

'images/still\_life\_medium.jpg',

'index.html',

'pages/offline.html',

'pages/404.html'

];

const staticCacheName = 'pages-cache-v1';

self.addEventListener('install', event => {

console.log('Attempting to install service worker and cache static assets');

event.waitUntil(

caches.open(staticCacheName)

.then(cache => {

return cache.addAll(filesToCache);

})

);

});

拦截http请求，对404和offline做特殊处理

self.addEventListener('fetch', event => {

console.log('Fetch event for ', event.request.url);

event.respondWith(

caches.match(event.request)

.then(response => {

if (response) {

console.log('Found ', event.request.url, ' in cache');

return response;

}

console.log('Network request for ', event.request.url);

return fetch(event.request)

.then(response => {

If(response.status === 404) {

return caches.match('pages/404.html')

}

return caches.open(staticCacheName).then(cache => {

cache.put(event.request.url, response.clone());

return response;

});

});

}).catch(error => {

return caches.match('pages/offline.html’)

})

);

});

清理这些不使用的缓存

self.addEventListener('activate', event => {

console.log('Activating new service worker...');

const cacheWhitelist = [staticCacheName];

event.waitUntil(

caches.keys().then(cacheNames => {

return Promise.all(

cacheNames.map(cacheName => {

if (cacheWhitelist.indexOf(cacheName) === -1) {

return caches.delete(cacheName);

}

})

);

})

);

});

配置manifest.json

<link rel="manifest" href="/manifest.json">

{

"name": "Space Missions",

"short\_name": "Space Missions",

"lang": "en-US",

"start\_url": "/index.html",

"display": "standalone",

"theme\_color": "#FF9800",

"background\_color": "#FF9800",

"icons": [

{

"src": "images/touch/icon-128x128.png",

"sizes": "128x128"

},

{

"src": "images/touch/icon-192x192.png",

"sizes": "192x192"

},

{

"src": "images/touch/icon-256x256.png",

"sizes": "256x256"

},

{

"src": "images/touch/icon-384x384.png",

"sizes": "384x384"

},

{

"src": "images/touch/icon-512x512.png",

"sizes": "512x512"

}

]

}

编写一个PWA的关键步骤：

1. 添加manifest.json相关配置
2. 添加meta配置
3. 添加serviceworker

一个App Shell的主体

<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="utf-8">  
  <title>App Shell</title>  
  <link rel="manifest" href="/manifest.json">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>App Shell</title>  
  <link rel="stylesheet" type="text/css" href="styles/inline.css">  
</head>  
  
<body>  
  <header class="header">  
    <h1 class="header\_\_title">App Shell</h1>  
  </header>  
  
  <nav class="nav">  
  ...  
  </nav>  
  
  <main class="main">  
  ...  
  </main>  
  
  <div class="dialog-container">  
  ...  
  </div>  
  
  <div class="loader">  
    <!-- Show a spinner or placeholders for content -->  
  </div>  
  
  <script src="app.js" async></script>  
  <script>  
  if ('serviceWorker' in navigator) {  
    navigator.serviceWorker.register('/sw.js').then(function(registration) {  
      // Registration was successful  
      console.log('ServiceWorker registration successful with scope: ', registration.scope);  
    }).catch(function(err) {  
      // registration failed :(  
      console.log('ServiceWorker registration failed: ', err);  
    });  
  }  
  </script>  
</body>  
</html>

## Performance optimization and caching

indexDB

创建数据库，添加数据，查询数据：

<https://github.com/google-developer-training/pwa-training-labs/blob/master/indexed-db-lab/solution/js/main.js>

预加载

<link rel="preload" as="script" href="super-important.js">

预连接

<link rel="preconnect" href="https://example.com">

您可以根据此图表利用以下方式提升您的关键渲染路径CRP：

将关键资源数降至最低：消除关键资源、延迟关键资源的下载并将它们标记为不同步等。

优化关键字节数以缩短下载时间（往返次数）。

优化其余关键资源的加载顺序：尽早下载所有关键资产，以缩短关键路径长度。

## ES2015 concepts and syntax

Promise

const promise = new Promise((resolve, reject) => {

*// do a thing, possibly async, then...*

if (*/\* everything turned out fine \*/*) {

resolve("Stuff worked!");

}

else {

reject(Error("It broke"));

}

});