web前端开发技术基本概念（Edited by Labraff)

国内访问：https://gitee.com/labraff/frontend-collection/blob/master/web-fontend-basic.md

外网访问：https://github.com/Labraff/Frontend-Collection/blob/master/web-fontend-basic.md

基础知识

1. 用户场景： 用户使用Web浏览器访问某购物网站，在地址栏里输入：http://x.com/index.php ，然后按回车键，看到了琳琅满目的商品首页
2. 用户场景： 用户通过浏览器访问某网站的用户注册页面。如：http://x.com/users/new.php
    - 当用户未填写用户名，提交后，浏览器立即弹出对话框，提示用户名为必填
    - 当用户填写了一个常见的用户名如：tom，提交后，浏览器弹出对话框，提示该用户名已被占用
    - 当用户终于按要求正确填写所有字段，提交后，页面跳转至网站首页

Q：请基于上述用户场景，从Web浏览器和Web服务器、HTTP请求和响应、客户端检验与服务端校验、客户端渲染和服务端渲染等多个技术角度，说明在浏览器背后，经历了哪些技术阶段和发生了什么?

A：

1.用户使用Web浏览器访问某购物网站，依次经历Web浏览器本地解析http请求，转发dns服务器解析地址，将实际ip地址返回给浏览器，浏览器通过ip请求商品首页，web服务器接收浏览器发送的请求后，将相关资源返回给浏览器，浏览器将资源解析后就用户就能看到琳琅满目的商品首页。

2.用户通过浏览器访问某网站的用户注册页面，基本资源请求过程和上述相似。但在“当用户未填写用户名，提交后，浏览器立即弹出对话框，提示用户名为必填”发生的是客户端校验，只是通过js检测基本的格式是否正确。“当用户填写了一个常见的用户名如：tom，提交后，浏览器弹出对话框，提示该用户名已被占用”发生的服务端校验，这里通过将注册信息打包后请求服务器，服务器接收到注册信息后进入数据库校验信息，发现用户已存在，则让浏览器弹出提示框，这里用到的是服务端渲染。“当用户终于按要求正确填写所有字段，提交后，页面跳转至网站首页”同样是服务端校验，流程与上述相似。

语义

Q：什么是语义标记？请举一实例，然后至少结合三个应用场景(搜索引擎、视障人群等)说明为什么要使用语义标记?

A：在html中是一系列表示文章结构或者功能的标签，比如`<head><nav><aside><article><section><footer>` `<font>功能类标签与<p>文章结构语义标签`。在搜索引擎方面，以一个文章网站为例，使用语义标记可以有利于SEO和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息，爬虫依赖于标签来确定上下文和各个关键字的权重。在视障人群方面，如果使用的是含语义的标记，屏幕阅读器就会“逐个拼出”单词，而不是试着去对它完整发音。在跨设备渲染方面，PDA、手机等设备可能无法像普通电脑的浏览器一样来渲染网页（通常是因为这些设备对CSS的支持较弱）。使用语义标记可以确保这些设备以一种有意义的方式来渲染网页。理想情况下，观看设备的任务是符合设备本身的条件来渲染网页。语义标记为设备提供了所需的相关信息，就省去了你自己去考虑所有可能的显示情况（包括现有的或者将来新的设备）。

Q：基本的语义类标记，请举出一个实例(包括标记名称与主要属性)，说明下列各类语义元素的语义和典型用法：

A：

- 列表元素

- `<ul>、<li>`
- 主要用于无序列表的渲染

```
<html>

<body>

<h4>一个嵌套列表：</h4>
<ul>
  <li>咖啡</li>
  <li>茶
    <ul>
    <li>红茶</li>
    <li>绿茶
      <ul>
      <li>中国茶</li>
      <li>非洲茶</li>
      </ul>
    </li>
    </ul>
  </li>
  <li>牛奶</li>
</ul>

</body>
</html>
```

一个嵌套列表：

- 咖啡
- 茶
  - 红茶
  - 绿茶
    - 中国茶
    - 非洲茶
- 牛奶

- 表格元素
  - `<table>、<tr>、<td>`
  - 主要用于表格渲染，以及用于页面分块

```
<html>

<body>

<h4>这个表格有一个标题，以及边框：</h4>

<table border="1">
<caption>我的标题</caption>
<tr>
  <td>100</td>
  <td>200</td>
  <td>300</td>
</tr>
<tr>
  <td>400</td>
  <td>500</td>
  <td>600</td>
</tr>
</table>

</body>
```

这个表格有一个标题，以及边框：

我的标题

100	200	300
400	500	600

- 块元素 block Element
  - ◎ address - 地址 ◎ blockquote - 块引用 ◎ center - 居中对齐块 ◎ dir - 目录列表 ◎ div - 常用块级容器，也是css layout的主要标签 ◎ dl - 定义列表 ◎ fieldset - form控制组 ◎ form - 交互表单 ◎ h1 - 大标题 ◎ h2 - 副标题 ◎ h3 - 3级标题 ◎ h4 - 4级标题 ◎ h5 - 5级标题 ◎ h6 - 6级标题 ◎ hr - 水平分隔线 ◎ isindex - input prompt ◎ menu - 菜单列表 ◎ noframes - frames可选内容，（对于不支持frame的浏览器显示此区块内容） ◎ noscript - 可选脚本内容（对于不支持script的浏览器显示此内容） ◎ ol - 排序表单 ◎ p - 段落 ◎ pre - 格式化文本 ◎ table - 表格 ◎ ul - 非排序列表（无序列表）
  - 依据各自的语义有各自的功能，块级元素默认独占一行
- 内联元素 inline Element
  - ◎ a - 锚点 ◎ abbr - 缩写 ◎ acronym - 首字 ◎ b - 粗体(不推荐) ◎ bdo - bidi override ◎ big - 大字体 ◎ br - 换行 ◎ cite - 引用 ◎ code - 计算机代码(在引用源码的时候需要) ◎ dfn - 定义字段 ◎ em - 强调 ◎ font - 字体设定(不推荐) ◎ i - 斜体 ◎ img - 图片 ◎ input - 输入框 ◎ kbd - 定义键盘文本 ◎ label - 表格标签 ◎ q - 短引用 ◎ s - 中划线(不推荐) ◎ samp - 定义范例计算机代码 ◎ select - 项目选择 ◎ small - 小字体文本 ◎ span - 常用内联容器，定义文本内区块 ◎strike - 中划线 ◎ strong - 粗体强调 ◎ sub - 下标 ◎ sup - 上标 ◎ textarea - 多行文本输入框 ◎ tt - 电传文本 ◎ u - 下划线 ◎ var - 定义变量
  - 依据各自的语义有各自的功能，内联元素默认会接在上一内联元素后面
- 其它类型

表单(代码)

Q：请编写代码（HTML文件与Javascript文件分开编写），实现符合如下要求的表单

1. 接收表单的服务器地址为:"posts/save.php"
2. 表单数据将被写入到服务器端的数据库
3. 表单有一个单行文本框，且预置的文本为：“欢迎光临”
4. 表单有一个多行文本控件，且预置的文本为：“welcome！”
5. 表单有三个单选按钮，预置第二个单选按钮被选中
6. 表单有三个复选框，预置第一个和第二个复选框被选中
7. 表单有一个列表框控件，预置第三个列表项被选中
8. 表单有一个列表框控件，预置第一个和第三个列表项被选中
9. 表单有一个隐藏的控件，其值为“1”
10. 表单有一个文件上传的控件
11. 表单有一个提交表单按纽
12. 当表单被提交时，弹出一个对话框，展示以上各文本输入框、单选框、复选框、下拉列表框等控件的值

A:

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Labraff</title>
</head>
<body>
  <form id="form" action="posts/save.php" method="post">
    <input type="text" name="" placeholder="欢迎光临" />
    <textarea name="">welcome!</textarea>
    <input type="radio" name="day" value="day1" />day1
    <input type="radio" name="day" value="day2" checked />day2
    <input type="radio" name="day" value="day3" />day3
    <input type="checkbox" name="week" value="week1" checked />week1
    <input type="checkbox" name="week" value="week2" checked />week2
    <input type="checkbox" name="week" value="week3" />week3
    <select name="select">
      <option value="volvo">Volvo</option>
      <option value="saab">Saab</option>
      <option value="opel" selected>Opel</option>
    </select>
    <select name="selectList" multiple="multiple">
      <option value="volvo" selected>Volvo</option>
      <option value="saab">Saab</option>
      <option value="opel" selected>Opel</option>
    </select>
    <input type="hidden" name="hiddenVal" value="1">
    <input type="file">

    <input type="submit" value="提交" onclick="getAll()" />
  </form>
  <button onclick="getAll()">Btn</button>
</body>
<script>
  let getAll = () => {
    let f = document.forms[0].children
```

```
    console.log(f)
    let out = ''
    for (let item of f) {
      out = out + 'localName:' + item.localName + '      '
      if (!item.value) {
        out = out + 'value:' + 'empty\n'
      } else {
        out = out + 'value:' + item.value + '\n'
      }
    }
    alert(out)
  }
</script>
</html>
```

布局

Q：请按如图所示布局：

- 1.使用CSS的浮动(float)属性，编写代码（HTML文件与CSS文件分开编写），给予实现
- 2.请列举除浮动之外的至少三种列式布局技术，并一一说明其各自的适用场景、技术优势和不足之处

A：

1.

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Labraff</title>
  <style>
    .row{
      min-height: 120px;
      width: 100%;
      border: solid #000000 1px ;
      text-align: center;
    }
    .side{
      height: 400px;
      width: 20%;
      border: solid #000000 1px ;
      text-align: center;
      margin: auto;
    }
    #main{
      width: 50%;
      border: solid #000000 1px ;
    }
    .box{
      border: solid #000000 1px ;
      width: 40%;
      height: 140px;
      margin: auto;
      margin-bottom: 140px;
    }
```

```
  </style>
</head>
<body>
  <div class="row">顶部</div>
  <div class="row" style="display: flex;">
    <div class="side">左侧栏</div>
    <div id='main'>
      <h3>主内容区域</h3>
      <div id="innerbox" style="display: flex;">
        <div class="box">主内容区第一列</div>
        <div class="box">主内容区第二列</div>
      </div>
    </div>
    <div class="side">右侧栏</div>
  </div>
  <div class="row">第二行</div>
  <div class="row">第三行</div>
  <div class="row">底部</div>
</body>
</html>
```

2.

- 浮动布局（Float Layout）
  - 适用场景：多列盒子浮动时可以采用弹性盒，条目尺寸未知或动态时。
  - 技术优势：使用em或rem单位进行布局，相对%更加灵活，同时可以支持浏览器的字体大小调整和缩放等的正常显示。理想状态下所有屏幕的高宽比和最初的设计高宽比一样，或者相差不多，完美适应。
  - 不足之处：这种rem+js只不过是宽度自适应，高度没有做到自适应。
- 静态布局（Static Layout）
  - 适用场景：pc网站。
  - 技术优势：这种布局方式对设计师和CSS编写者来说都是最简单的，亦没有兼容性问题。
  - 不足之处：不能根据用户的屏幕尺寸做出不同的表现。
- 流式布局（Liquid Layout）
  - 适用场景：Web前端开发的早期历史上，用来应对不同尺寸的PC屏幕。
  - 技术优势：使用%百分比定义宽度，高度大都是用px来固定住，可以根据可视区域 (viewport) 和父元素的实时尺寸进行调整，尽可能的适应各种分辨率。
  - 不足之处：如果屏幕尺度跨度太大，那么在相对其原始设计而言过小或过大的屏幕上不能正常显示。因为宽度使用%百分比定义，但是高度和文字大小等大都是用px来固定，所以在大屏幕的手机下显示效果会变成有些页面元素宽度被拉的很长，但是高度、文字大小还是和原来一样（即，这些东西无法变得“流式”），显示非常不协调。
- 响应式布局（Responsive Layout）
  - 适用场景：所有终端上（各种尺寸的PC、手机、手表、冰箱的Web浏览器等等）都能显示出令人满意的效果。
  - 技术优势：适应pc和移动端，效果完美
  - 不足之处：1.媒体查询是有限的，也就是可以枚举出来的，只能适应主流的宽高。2.要匹配足够多的屏幕大小，工作量不小，设计也需要多个版本。

名词解释

- DOM
  - 文档对象模型（Document Object Model）它是一种与平台和语言无关的应用程序接口(API),它可以动态地访问程序和脚本,更新其内容、结构，是一种基于树的API文档。
- BOM
  - 浏览器对象模型（Browser Object Model）它提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。代表浏览器窗口的Window对象是BOM的顶层对象，其他对象都是该对象的子对象。常见子对象有window、navigator、screen、history、location、document、event
- CSS伪类
  - 无法由设计师定义,而是由浏览器根据**实时运行状态**定义的类，如

```
a:link {color: blue;} /* 未访问的链接 */
a:visited {color: red;} /* 已访问的链接 */
a:hover {color: black;} /* 鼠标划过链接 */
a:active {color: yellow;} /* 已选中的链接 */
```

- CSS伪元素
  - 伪元素选择器是对**内容元素**进行筛选的类，如

```
<p>This is a example!</p>
p:first-letter {
  color:#ff0000;
  font-size:xx-large;
}
p:first-line { color:#ff0000; }
```

- jquery
  - jQuery是一个快速、简洁的JavaScript框架，倡导写更少的代码，做更多的事情。它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优化HTML文档操作、事件处理、动画设计和Ajax交互。它具有独特的链式语法和短小清晰的多功能接口；具有高效灵活的css选择器，并且可对CSS选择器进行扩展；拥有便捷的插件扩展机制和丰富的插件。

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
//使p元素被点击后隐藏
```

- SCSS
  - SCSS是CSS预处理工具，采用类CSS的语言风格，但增加了诸如变量、循环、函数和继承等编程语言特性。

```
$bgColor:#fff
.container{
    height:200px;
    .box{
        height:100px;
        color:$bgColor;
    }
}
```

- 响应式设计
  - 响应式网页设计是伴随移动设备平台兴起而出现的技术，主要解决同一个Web页面在不同设备和环境中均能获得一致性用户体验的问题，支持各种尺寸与分辨率的设备，支持种种尺寸缩放的行为。
- 媒体查询
  - 通过查询当前属于哪种设备, 让网页能够在不同的设备下正常的预览。针对不同设备进行优化的，尤其是分辨率信息，从而可以针对不同的分辨率设置不同的布局方式。
- cookies
  - Cookie是保存在客户端的纯文本文件，一般用于存储用户登录信息，用于服务器验证用户在线状态。
- node.js
  - 是一个基于 Chrome V8 引擎的 JavaScript运行环境，能让 JavaScript 运行在服务端的开发平台。它使用事件驱动， 非阻塞I/O模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。
- 跨域访问
  - 跨域访问是指从一个域名的网页去请求另一个域名的资源，只要 协议、域名、端口有任何一个的不同就被认为是跨域。但在一个域与其他域建立了信任关系后，两个域之间不但可以按需要相互进行管理，还可以跨网分配文件和打印机等设备资源，使不同的域之间实现网络资源的共享与管理。
- 同源策略
  - 同源策略，是一个著名的安全策略，所谓同源是指，域名，协议，端口相同，浏览器会对不同源的脚本或者文本的访问方式进行的限制。比如源a的js不能读取或设置引入的源b的元素属性。那么先定义下什么是同源，所谓同源，就是指两个页面具有相同的协议，主机（也常说域名），端口，三个要素缺一不可。
- 事件驱动
  - 事件驱动，简单地说就是根据事件的发生情况来执行操作（调用函数）。事件驱动程序的基本结构是由一个事件收集器、一个事件发送器和一个事件处理器组成。事件收集器专门负责收集所有事件来源（如鼠标、键盘事件、时钟事件等）。事件发送器负责将收集器收集到的事件分发到目标对象中。事件处理器做具体的事件响应工作，它往往要到实现阶段才完全确定，因而需要运用虚函数机制（函数名往往取为类似于HandleMsg的一个名字）。
- 定时器
  - js运行环境提供的一种执行代码的机制。通过定时器，开发者可以不依赖于用户的操作而自动调用执行某些代码，且支持异步执行
- URL
  - 统一资源定位系统（uniform resource locator）是因特网服务程序上用于指定信息位置的表示方法。

项目实践

~~这部分瞎yy的，结合项目根据个人项目来说明即可~~

请结合你的个人项目案例，举例说明：

Q：什么是Web页面的内容、结构、样式、行为？这种分层的设计模式有什么好处？前端开发技术中还有哪些设计模式？请列举出至少一种并说明。

A：内容，具体展示给用户浏览的，具有实际语义的文本或媒体；结构，通过一定的布局来表示文档叙述逻辑；样式，用于美化结构与内容的视觉设计；行为，通过一定的交互事件来形成一定的反馈，或者根据事件触发某些函数以实现某些功能。

这样的分层设计让开发逻辑更加清晰简洁，分工更加明确，任务可以并行；比如可以一部分开发者进行结构与样式的设计（UI）、一部分开发者进行事件脚本的编写。

常见的前端开发设计模式：MVC设计模式、MVVM设计模式

MVC是一种设计模式，它将应用划分为3个部分：数据（模型）、展现层（视图）和用户交互（控制器）。换句话说，一个事件的发生是这样的过程：用户和应用产生交互->控制器的事件处理器被触发->控制器从模型中请求数据，并将其交给视图->视图将数据呈现给用户。V层是视图层，控制界面显示，将界面与数据连接。M层存放数据，处理逻辑，比如处理从数据库调用的数据，还有业务逻辑处理。C层用于连接M和V，Controller 负责显示界面、响应用户的操作以及与 Model 交互，比如操作DOM、对事件的监听。这就导致了Controller和 View 紧耦合、逻辑复杂，难以维护。MVVM与MVC最大的区别就是：它实现了View和Model的自动同步，也就是当Model的属性改变时，我们不用再自己手动操作Dom元素，来改变View的显示，而是改变属性后该属性对应View层显示会自动改变。由此和view层分开来，只是注重数据的改动即可。**MVVM设计模式中最重要的就是实现了View和Model的自动同步，所以可以不用频繁地手动操作DOM元素。**

比如说Vue框架，Vue实例中的data相当于Model层，而ViewModel层的核心是Vue中的双向数据绑定，即Model变化时VIew可以实时更新，View变化也能让Model发生变化。 整体看来，MVVM比MVC精简很多，不仅简化了业务与界面的依赖，还解决了数据频繁更新的问题，不用再用选择器操作DOM元素。

Q：什么是Web前端技术、Web后端技术？两者的优缺点是什么？

A：Web前端技术是创建Web页面或app等前端界面呈现给用户的过程，通过HTML，CSS及JavaScript以及衍生出来的各种技术、框架、解决方案，来实现互联网产品的用户界面交互。Web后端技术用于前端与服务端进行的数据交互，通常是从数据库或其他数据源写入、读取和处理数据。主要区别体现在文档的渲染代码生成操作是通过前端实现还是后端实现，两者都是以前后端混合开发为基础的。

Web前端技术优点主要体现在运行流畅度好，用户交互性好，但单独的前端无法实现多用户数据共享等，Web后端优点主要体现在能支持多用户信息交互，而存在的缺点主要体现在网络通信延迟上，容易带来流畅度低 的问题。

Q：你的下一版本会考虑使用哪些新的开发模式、技术或工具？譬如前后端分离开发模式、前端组件化、反应式编程等。说说你对这些新模式或技术的理解，给出升级的思路和代码实例。

A：一直采用前后端分离开发模式、同时也注重前端组件化开发。

前后端分离开发模式具有一个很明显的优点，它能让前后端各自独立开发，降低开发耦合度，前后端的信息通过协议来传递，代码耦合度也被极大降低，复用性变得极强。同时前端组件化开发同样是种低耦合高复用的开发思路，让一些非常常用的前端组件代码优化封装，只需几行代码引入即可轻松实现相应功能，让开发者专注于业务逻辑的开发，而非拘泥于一些样式结构的设计。

升级的思路无非将之前的复用度高的代码封装，模板化。这里的模板化是指聚合了结构、样式、事件的综合组件，只需要设置相应参数即可实现功能。同时对于一些计算型函数也加以封装，丰富个人的utils.js。同时根据各自框架的特点设计各自框架的包调用模型，将程序代码开发规范化，流程化。最common的栗子，request封装。

```
request(opts){
    if (opts.login==true) {
        let promise  = new Promise(function(resolve,reject){
            uni.navigateTo({
                url: "/pages/login/login",
            })
        })
        return promise
    } else {
        let baseinfo = {
            url: opts.url,
            method: opts.method,
            header: opts.token ? {
                Authorization: 'Bearer ' + opts.token,
                shopId: 0,
            } : {
                shopId: 0,
            },
            dataType: 'application/json',
            data:opts.data,
        }
        let promise = new Promise(function(resolve, reject) {
            uni.request(Object.assign(baseinfo, {
                success: function(res) {
                    resolve(JSON.parse(res));
                },
                fail: function(res) {
                    reject(res);
                    uni.showToast({
                        duration:1000,
                        icon:'loading',
                        title:'网络开小差~'
                    })
                }
            }))
        })
        return promise;
    }
}
```

辨析题

请说明下列各概念，并解释它们的区别(正反例)

- 安全字体、网络字体、图标字体
    - 安全字体：在Web编码中，CSS默认应用的Web字体，使用安全字体不容易出现字体相关的错误。
    - 网络字体：Web浏览者可能本地不存在Web开发者设定的字体，那么就需要从网络上获取这些字体资源，CSS有一种标记叫做`@font-face`，在`@font-face`声明里，你可以声明一种字体，指定这种字体字体库文件从网络中的某个地方下载。

```
@font-face {
  font-family: '徐静蕾手写体';
  src: url('http://www.webhek.com/徐静蕾手写体.eot'); /* IE9 */
  src: url('http://www.webhek.com/徐静蕾手写体.eot?#iefix') format('embedded-opentype'),/* IE6-IE8 */
       url('http://www.webhek.com/徐静蕾手写体.woff') format('woff'),/* 所有现代浏览器 */
       url('http://www.webhek.com/徐静蕾手写体.ttf')  format('truetype'),/* Safari,Android,iOS */
       url('http://www.webhek.com/徐静蕾手写体.svg#svgFontName') format('svg');/* Legacy iOS */
}
```

    - 图标字体：图标字体是使用微小图像而不是字母形式的字体。 与字符一样，每个图标字体都是矢量元素，可根据需要进行伸缩，并可以使用CSS样式进行修改。使用图标字体的主要原因是可以轻松更改大小，颜色，形状。 图标字体本质上是透明的，因此可以将它们放在任何颜色或类型的背景上，还可以添加笔触或更改图标的不透明度。
- Javascript与node.js
    - Javascript：JavaScript 是一种轻量级的编程语言。JavaScript 是可插入 HTML 页面的编程代码。JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。
    - node.js：是一个基于 Chrome V8 引擎的 JavaScript运行环境，能让 JavaScript 运行在服务端的开发平台。它使用事件驱动， 非阻塞I/O模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。
- DOM、HTML DOM 与 SVG DOM
    - DOM：DOM 是 Document Object Model（文档对象模型）的缩写。DOM 是 W3C（万维网联盟）的标准。DOM 定义了访问 HTML 和 XML 文档的标准：“W3C 文档对象模型 （DOM）是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。”W3C DOM 标准被分为 3 个不同的部分：核心 DOM - 针对任何结构化文档的标准模型、XML DOM - 针对 XML 文档的标准模型、HTML DOM - 针对 HTML 文档的标准模型
    - HTML DOM：HTML DOM 是：HTML 的标准对象模型、HTML 的标准编程接口，HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法。换言之，HTML DOM 是关于如何获取、修改、添加或删除 HTML 元素的标准。
    - SVG DOM：使用脚本可以很方便的完成各种复杂的任务，也是完成动画和交互的一种主流方式。由于SVG是html的元素，所以支持普通的DOM操作，又由于SVG本质上是xml文档，所以也有一种特殊的DOM操作，大多称之为SVG DOM。
- 窗口、文档
    - 窗口：浏览器窗口
    - 文档：html中body标签里面的内容
- 客户端本地存储、服务端网络存储
    - （用户登录和缓存方面）
    - 客户端本地存储：把一些信息存储到客户端本地（主要目的有很多，其中有一个就是实现多页面之间的信息共享）
        1. 离线存储（xxxx.manifest），H5处理离线缓存还是存在一些硬伤，所以真实项目中一般还是传统的【NATIVE APP】来完成这件事情。
        2. localStorage / sessionStorage：H5中新增加的API，基于这个API可以把一些数据缓存到客户端本地（常用）。
        3. IndexedDB / webSQL：本地数据库存储。
        4. Cookie：本地信息存储（常用）。
        5. CacheStorage / ApplicationCache：本地缓存存储。
    - 服务端网络存储：Session

1. session是服务器端存储
2. 在服务器端建立session之后，服务器喝当前客户端之间会建立一个唯一的标识（sessionID / sid），而本次存储的session信息都存放到对应的sid下（目的是为了区分不同客户端在服务器上建立session信息，后期查找的时候，可以找到自己当初建立的）。

- 同步调用、异步调用
  - 同步调用：同步调用就是客户端等待调用执行完成并返回结果。
  - 异步调用：异步调用就是客户端不等待调用执行完成返回结果，不过依然可以通过回调函数等接收到返回结果的通知。
- URL绝对地址、URL相对地址
  - URL绝对地址：网站主页上的文件或目录在硬盘上真正的路径。

    绝对路径: `<a href="/sub/news.html" >link</a>`
    如当前域名为`http://x.com`,则浏览器生成的最终地址为`http://x.com/sub/news.html`
  - URL相对地址：指出的位置是以信息提供者的位置为参照的。

    相对路径： `<a href="../news.html" >link</a>`
    如该标记当前所在页面为`http://x.com/bbs/index.html`,则浏览器生成的最终地址为`http://x.com/news.html`
- 请指出C:\www\index.html 、 file:///C:/www/index.html 和 http://x.com/index.html 这三者的区别？以及各自的功能限制？
  - C:\www\index.html ： 本地磁盘文件访问路径，它只能用于访问本地磁盘上的文件
  - file:///C:/www/index.html：一种本地文件传输协议，File协议主要用于访问本地计算机中的文件，就如同在Windows资源管理器中打开文件一样。
  - http://x.com/index.html：通过基于TCP/IP通信的超文本传输协议来传递数据，功能限制主要体现在网络如果不连通就无法访问，容易受外界干扰。
- 静态定位、浮动定位、绝对定位、固定定位
  - 静态定位：static，浏览器的默认定位方式，即按照html文档的代码顺序（即文档流）进行依次定位。
  - 浮动定位：float，浮动严格讲不算是一种定位方式。它的最初用途是实现文字环绕的效果。如同绝对定位，元素将失去其在文档流中的位置和空间，重新按浮动的方向定位。与绝对定位不同的是，它位置坐标的参考原点或浮动范围只能是父元素，且不能与其它元素产生重叠,而是产生环绕的效果。
  - 绝对定位：absolute，一旦设置绝对定位，该元素就好象被从文档流中删除，该元素也就将失去了在正常文档流中所占的位置与空间。该元素的新位置将以距离其最近一个且不是静态定位的祖先元素为坐标原点进行偏移。并且将转型为一个块状盒子，而不论原来它在正常流中属于何种类型。
  - 固定定位：fixed，是绝对定位的一种变形，即其参考原点将是屏幕可视区域，而不是文档区域。常用来实现诸如顶部固定导航等效果。文档区域的长宽度,是由文档的内容和样式设置共同决定的。而屏幕可视区域的高宽度则是物理屏幕及浏览器窗口的大小决定的 当文档区域的长度 > 屏幕可视区域的高度,会出现纵向滚动条 当文档区域的宽度 > 屏幕可视区域的宽度,会出现横向滚动条。
  - (补充)相对定位：relative，先假设元素仍然按静态定位确定位置，然后以此位置为参考原点，根据设置的偏离值获得实际的位置，但所占空间仍然保留。并且可以其它元素重叠。换言之，如果你为元素设置了相对定位，但未设置偏离值,则该元素的显示和静态定位并无区别。但是，该元素的状态将变为已定位元素，可以成为绝对定位的参考原点。
- 前端模板引擎、后端模板引擎
  - 前端模板引擎：主要结合js实现html，一种以handlebar mustache为代表，实现方式为拼字符串。另一种以react为代表，实现方式为virtual Dom。
  - 后端模板引擎：以velocity这种为代表，可结合Java等语言实现，由服务端生成html返回客户端。

- ~~设计时、构建时(build)、编译时、运行时~~
  - ~~设计时：编写代码的时间状态~~
  - ~~构建时(build)：代码编译成可执行项目时间状态~~
  - ~~编译时：代码被编译为指定类型文件的时间状态~~
  - ~~运行时：编译后代码在被执行的时间状态~~
- localStorage 、 sessionStorage 与cookie
  - localStorage ： localStorage 是 HTML5 标准中新加入的技术，它并不是什么划时代的新东西。早在 IE 6 时代，就有一个叫 userData 的东西**用于本地存储**，而当时考虑到浏览器兼容性，更通用的方案是使用 Flash。而如今，localStorage 被大多数浏览器所支持，如果你的网站需要支持 IE6+，那以 userData 作为你的 polyfill 的方案是种不错的选择。
  - sessionStorage：sessionStorage 与 localStorage 的接口类似，但保存数据的生命周期与 localStorage 不同。做过后端开发的同学应该知道 Session 这个词的意思，直译过来是“会话”。而 sessionStorage 是一个前端的概念，它只是可以将一部分数据在当前会话中保存下来，刷新页面数据依旧存在。但**当页面关闭后，sessionStorage 中的数据就会被清空。**
  - cookie：Cookie 是小甜饼的意思。顾名思义，cookie 确实非常小，它的大小限制为4KB左右。它的主要用途有**保存登录信息**，比如你登录某个网站市场可以看到“记住密码”，这通常就是通过在 Cookie 中存入一段辨别用户身份的数据来实现的。
- http协议、https协议
  - http协议：超文本传输协议，是一个基于请求与响应，无状态的，应用层的协议，常基于TCP/IP协议传输数据，互联网上应用最为广泛的一种网络协议,所有的WWW文件都必须遵守这个标准。设计HTTP的初衷是为了提供一种发布和接收HTML页面的方法。
  - https协议：HTTPS是一种通过计算机网络进行安全通信的传输协议，经由HTTP进行通信，利用SSL/TLS建立全信道，加密数据包。HTTPS使用的主要目的是提供对网站服务器的身份认证，同时保护交换数据的隐私与完整性。

简答

- 请举一实例，说明基于ajax的Http请求方式与传统的方式有何区别？其优势是什么？
  - ajax：通过xmlHttpRequest对象请求服务器服务器接受请求返数据实现刷新交互，xmlhttp请求的**发起者可以是页面中的任何元素**，浏览器接收到服务器的响应后传递给对应的处理函数，由该函数决定做什么，其优势主要体现在请求发起者更加精确，请求目的性更强，可以由事件触发请求。
  - Http：通过httpRequest对象请求服务器接受请求返数据需要页面刷新，传统的http请求的发起者对于是当前页面，浏览器接收到服务器的响应后要刷新整个页面。
- 假如 http://x.com/1.json 和 http://x.com/1.php 两个url返回的是相同的json内容，请问，这两者有何不同?
  - http://x.com/1.json：返回的是静态json数据，不会根据情况修改内容，一般作为只读型资源。
  - http://x.com/1.php：返回的是动态生成的json数据，一般根据相应的请求返回相应的json数据。
- 请列举出三种动态创建或修改页面的技术，并各举一实例，并比较其优劣
  - 基于DOM、基于js、基于后端技术ASP、PHP、JSP

```
<!DOCTYPE html>
<html>
<body>
<script>
    //基于DOM
    document.write()
```

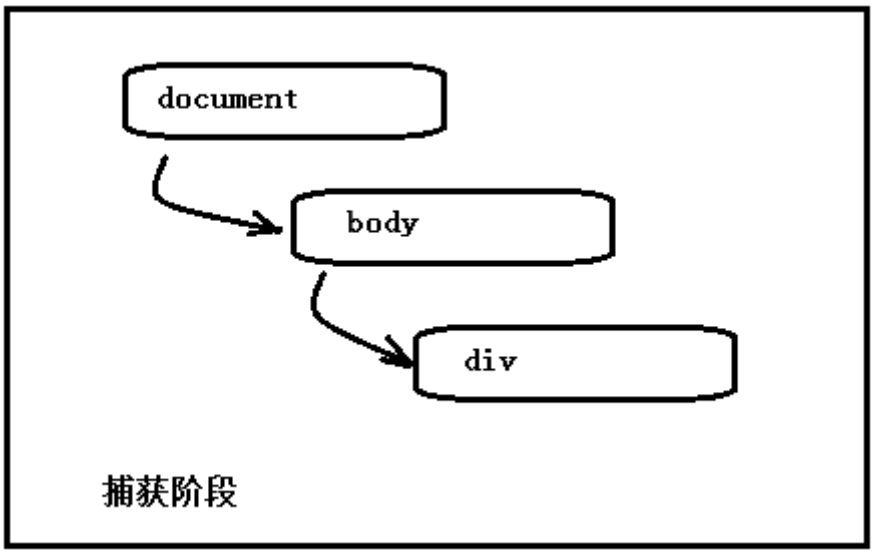
```
    document.getElementById("targetdiv").innerHTML = "<p>This
is<em>my</em>Content.</p>";
    //基于js createElement();  appendChild();
    var para = document.createElement("p");//创建一个p标签
    var info = "nodeName:";
    info += para.nodeName;
    info += "   nodeType:";
    info += para.nodeType;
</script>
<?php
    //基于后端脚本引擎
    echo "<h1>My first JSP script!</h1>";
?>

</body>
</html>
```

- 什么是前端构建，为什么需要前端构建?
  - 使用某些前端构建工具在自动化地将某些前端框架编写的代码打包成浏览器能解析的文件，比如使用Vue编写网页，通过webpack打包成html/css/js文件让浏览器解析。
  - 1.性能优化
    都知道浏览器请求的文件越多越耗时，请求的文件越大越耗时，尤其是在现在很多使用前端MVC, MVVM框架的时候，为了前端代码更清晰，结构更合理，我们就由很多JS文件，无疑又拖慢了网页的速度。
    2.文件合并
    浏览器需要下载多个JS文件，而浏览器是有并发限制，也就是同时并发只能下载几个文件，假如浏览器并发数是5，你有20个JS文件，而每5个需要2S, 那么光下载JS文件都需要8S，那么网页的性能可想而知，所以需要合并多个文件以减少文件的数量。
    3.文件压缩
    文件越大，下载越慢，而针对JavaScript和CSS, 里面的空格，换行这些都是为了让读代码时更容易阅读，但是对机器来说，这些对它没有影响，所以为了减少文件大小，一般的情况都会用工具去掉空格和换行，有时候还会用比较短的变量名(记住这个要让工具最后压缩时做，而源代码一定要保证命名可读性) 来减少文件大小。
    而所有的前端构建工具都具有文件合并和压缩的功能。
- 请指出四种事件类型，并举一实例说明，其用法和用途
  - 文档事件：指由于文档元素(document对象)的变化所引发的事件，如load、change、submit等`<input onclick='submit'>`
  - 窗口事件：指浏览器窗口(window对象)的变化所引发的事件，如load、resize、close等``
  - 输入事件：指键盘、鼠标等所发出的事件，如keyup、keydown、click、mouseover等
  - 系统事件：指来自系统内部的事件，如定时器
- 如果要获取某一个元素对象的引用，有哪三种实现技术？请试举一例，一一说明
  - 通过document对象获取，`getElementById、getElementsByTagName、getElementByName`
  - 通过jquery第三方库获取，`$("li").get(0)`
  - </?>
- 在node.js环境下，能使用jquery库吗？为什么?
  - 一般情况下不可以
- 什么是单页应用？和服务端驱动的多页应用相比？它有什么特点？其优势与劣势是什么?
  - 单页面应用（SPA）通俗点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的html,css,js。所有的页面内容都包含在这个所谓的主页中。但是写的时候，还是会分开写（页面片段），然后在交互的时候由 路由程序动态载入，单页面的页面跳转，仅刷新局部资源。多应用于PC端。多页面应用 （MPA）一个应用中有 **多个页面**，页面跳转的时候是 **整页进行刷新。**

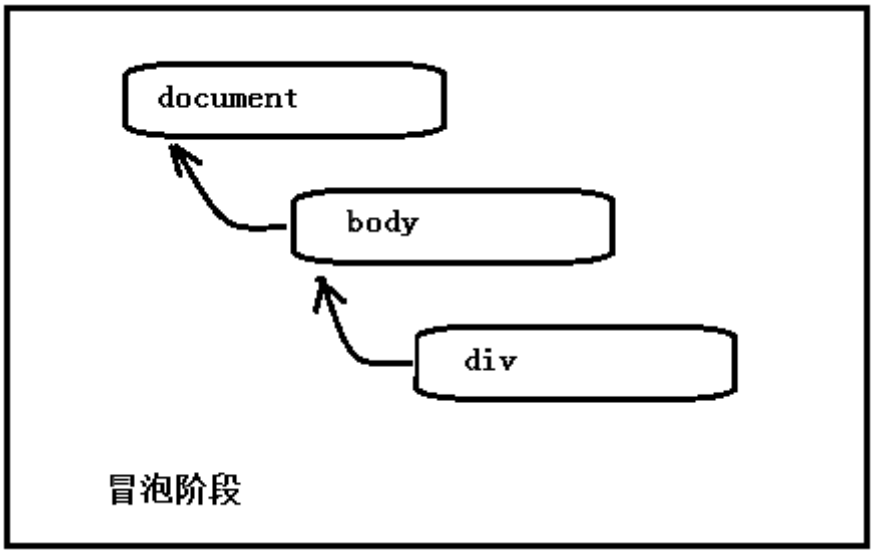
    - 优点：用户体验好，快，内容的改变不需要加载整个页面，基于这一点 SPA 对服务器压力较小。 前后端分离。 页面效果会比较炫酷（比如切换页面内容时的专场动画）。
    - 缺点：不利于 SEO。 导航不可用，如果一定要导航需要自行实现前进、后退，由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理。 初次加载时耗时多。首屏加载速度慢，页面复杂度提高很多。
- 如何实现跨标签页通讯?
    - localstorage：localstorage是浏览器多个标签共用的存储空间，所以可以用来实现多标签之间的通信
    - SharedWorker：可以被多个window共同使用，但必须保证这些标签页都是同源的
    - websocket等
- 什么是Web Worker?
    - 当在 HTML 页面中执行脚本时，页面的状态是不可响应的，直到脚本已完成。web worker 是运行在后台的 JavaScript，独立于其他脚本，不会影响页面的性能。您可以继续做任何愿意做的事情：点击、选取内容等等，而此时 web worker 在后台运行。
- 什么是reflow(回流)和repaint(重绘)优化？如何减少重绘和回流?
    - 回流：对DOM树进行渲染，只要修改DOM或修改元素的形状大小，就会触发reflow，reflow的时候，浏览器会使已渲染好受到影响的部分失效，并重新构造这部分，完成reflow后，浏览器会重新绘制受影响的部分到屏幕中
    - 重绘：对DOM的修改导致的样式变化，但未影响几何属性时，浏览器不需要重新计算元素的几何属性，直接可以为该元素绘制新的样式，跳过了回流环节，这个过程就叫重绘。
    - 减少对DOM的操作，直接改变className，如果动态改变样式，则使用cssText（减少设置多项内联样式），让元素脱离动画流，减少render 树的规模，避免使用table布局
- 浏览器如何阻止事件传播，阻止默认行为
    - 1.事件捕获`event.stopPropagation()`
    - 2.事件冒泡`event.cancelBubble=false`
    - 3.阻止浏览器默认行为：`event.preventDefault()`火狐/谷歌
    - 4.阻止浏览器默认行为：`event.returnValue=false`低版本IE
- 虚拟DOM方案相对原生DOM操作有什么优点，实现上是什么原理?
    - Virtual DOM是对DOM的抽象，本质上是JavaScript对象，这个对象就是更加轻量级的对DOM的描述。从浏览器解析性能上讲，虚拟DOM通过js运算，将一些没必要的重复渲染操作进行删减，提高了渲染效率。从开发便利性角度来讲，通过js操作虚拟DOM比直接操作DOM对象更加方便，而且主流框架采用声明式开发，对虚拟DOM做了很多封装，使用更加方便。
    - 虚拟DOM实现原理主要包括三部分：用 JavaScript 对象模拟真实 DOM 树，对真实 DOM 进行抽象；diff 算法 — 比较两棵虚拟 DOM 树的差异；pach 算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM 树。本质是采用差量计算来最小化增量更新渲染节点数。
- 浏览器事件机制中事件触发三个阶段
    - javaScript事件的三个阶段：捕获阶段 目标阶段 冒泡阶段

      捕获阶段：事件从根节点流向目标节点，途中流经各个DOM节点，在各个节点上触发捕获事件，直到达到目标节点。



目标阶段：事件到达目标节点时，就到了目标阶段，事件在目标节点上被触发。

冒泡阶段：事件在目标节点上触发后，不会终止，一层层向上冒，回溯到根节点。



- 什么是浏览器缓存?
  - 为了节约网络的资源加速浏览，浏览器在用户磁盘上对最近请求过的文档进行存储，当访问者再次请求这个页面时，浏览器就可以从本地磁盘显示文档，这样就可以加速页面的阅览。