Service Worker实现浏览器端页面渲染或CSS,JS编译

这篇文章发布于 2018年04月24日，星期二，22:49，归类于 JS实例。 阅读 12320 次, 今日 4 次 14 条评论

Service Worker提供了一种能力，可以fetch请求的资源，然后后Service Worker中进行编译或转化，返回处理后的其他资源，这种特性可以用来实现各种资源的在线的客户端编译，本文就将抛砖引玉，通过两个应用案例，展示未来web开发可能的面貌。

一、Service Worker与直接数据和HTML模板渲染页面

现在很多网站的页面是基于Node.js直接渲染输出的，以实现完全的前后端分离，而这种渲染本质上还是在服务端进行的，只是渲染的语言是JavaScript。

实际上，有了Service Worker，我们可以直接在客户端，也就是浏览器里面直接进行渲染，优点是……乍一想，其实没什么特别的优点，无非是传输的HTML页面的体积小了，不过这点优化的体积就像是隔靴搔痒，杯水车薪，不值一提。但后来有一想，不对啊，这HTML编译全在客户端执行，岂不是省了很多服务器，为公司省了很多钱，因为烧的都是用户的电。

而且，毕竟是另外的一种解决思路，保不准某些场景下能大放异彩呢，抛砖引玉嘛，重在能力展示。

我们先看实际例子，您可以狠狠地点击这里：Service Worker与客户端渲染HTML渲染页面demo

打开一看，就是个列表，而且数据还是2012年北京蔬菜价格什么鬼？首先，数据什么的不是重点，只从为了节约时间从这篇文章“基于HTML模板和JSON数据的JavaScript交互”弄过来的，

这个页面玄机的要看开发文件源代码才行（右键页面源代码都不行）：https://github.com/zhangxinxu/https-demo/tree/master/online-render

可以看到，`index.html`文件源代码列表明明是模板语法：

```
<ul class="fix cho_list_title">
    <li class="cho_list_li">名称</li>
    <li class="cho_list_li tc">价格(元)</li>
    <li class="cho_list_li tc">数量/操作</li>
</ul>
<%for (var i=0;i<goods.length;i++) {%>
    <%if (goods[i].num == 0){%>
    <ul class="fix cho_list">
        <li class="cho_list_li"><%=goods[i].name%
        <li class="cho_list_li tc g6"><%=goods[i]
        <li class="cho_list_li tc">
            <a href="javascript:" class="cho_btn"
        </li>
    </ul>
    <%} else {%>
    <ul class="fix cho_list cho_list_on">
        <li class="cho_list_li"><%=goods[i].name%
        <li class="cho_list_li tc g6"><%=goods[i]
        <li class="cho_list_li tc">
            <span class="cho_nums">
```

但是我们右键页面，查看HTML源代码的时候确是真实的列表内容：

view-source:https://zhangxinxu.github.io/https-demo/online-render/index.html

```
<!-- 右侧主列表 -->
<div id="listOut" class="cell">
    <ul class="fix cho_list_title">
        <li class="cho_list_li">名称</li>
        <li class="cho_list_li tc">价格(元)</li>
        <li class="cho_list_li tc">数量/操作</li>
    </ul>

        <ul class="fix cho_list cho_list_on">
            <li class="cho_list_li">白萝卜</li>
            <li class="cho_list_li tc g6">0.4/斤</li>
            <li class="cho_list_li tc">
                <span class="cho_nums">
                    <a href="javascript:" class="cho_desc" name="choOpt" title="减少一份"
                    </a><input class="cho_input" name="choInput" value="1" id="000010" t
                    href="javascript:" class="cho_asc" name="choOpt" title="增加一份" re
                </span>
            </li>
        </ul>


        <ul class="fix cho_list">
            <li class="cho_list_li">花生</li>
            <li class="cho_list_li tc g6">2.25/斤</li>
            <li class="cho_list_li tc">
                <a href="javascript:" class="cho_btn" name="choOpt" title="购买" rel="1"
            </li>
        </ul>
```

为什么会发生这么神奇的事情？玄机就是Service Worker在背后做了HTML的数据渲染。

首先，注册Service Worker，代码如下：

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('./sw-reader.js', {
        scope: './'
    });
}
```

然后关键就是`sw-reader.js`（访问点击这里），做了两件事情：

1. 内置极简模板渲染引擎，采用的是老版本的artTemplate，大小仅2.3K。截图如下：

2. 捕获 `.html` 的请求，并对获取到的的HTML字符内容进行模板渲染，并返回输出。完成JS代码如下：

```
self.addEventListener('fetch', function(event) {
  event.respondWith(async function () {
    if (/\.html/.test(event.request.url)) {
      let res = await fetch(event.request);
      let text = await res.text();

      var data = {};
      // 过滤JSON数据，使不暴露给用户
      text = text.replace(/([\w\W]*?)<\/script>/, function (matchs, $1) {
        // 获取页面渲染的JSON数据
        data = JSON.parse($1);
        return '';
      });
      // 模板text和数据data进行渲染并重新返回给浏览器
      return new Response(template.compile(text)(data), {
        headers: {
          'content-type': 'text/html; charset=utf-8'
        }
      });
    }
    return fetch(event.request);
  }());
});
```

也就是变量 `text` 是原始的HTML代码，而在Service Worker中返回的是 `template.compile(text)(data)` 渲染后的HTML代码，于是，浏览器中呈现的就是最终的列表数据而非原始模板HTML。

在本demo中，我是将页面需要的JSON数据放在了模板页面头部，并使用 `<script type="text/json">` 进行标示，截图如下：

`sw-reader.js` 中就是匹配 `<script type="text/json">` 获取JSON数据来进行渲染的。当然，你也可以使用其他标识，例如，前后都3个美元符号包起来进行识别，例如：

```
$$${
   list: []
}$$$
```

就是不怎么雅观。OK，小例子，抛砖引玉，下面看另外一个对静态资源进行编译的例子。

二、Service Worker与浏览器直接CSS编译

CSS一些预编译工具，例如，Less，Stylus都是使用JavaScript进行编译的，而这些编译工具都是有web版的，因此，理论上我们是可以把Less，Stylus的编译直接搬到浏览器客户端里面，然后我们就不用加载CSS文件了，直接加载 `less` 文件，Service Worker中编译成CSS并返回，页面也是可以正常访问的。

然后，问题在于，Less，Stylus的web版解析器体积有些大，压缩后还有150多K，个人觉得成本有点高了。

为了快速书写，我自己弄了个名为Qcss的编译工具，可以讲CSS声明自定义为缩写，以实现写CSS代码如马踏飞燕，例如：

```
.class-a { dib; w100; tc; }
```

可以实时转换成：

```
.class-a {
  display: inline-block;
  width: 100px;
  text-align: center;
}
```

而这个编译的JS方法提及非常小，仅2K多一点，和less的JS代码量相比简直就是毛毛雨，因此非常使用用在Service Worker实现一个可以在线编译CSS的功能。

先看例子，您可以狠狠地点击这里：Qcss和Service Worker在浏览器中编译成CSS demo

我们打开控制台查看网络请求，会发现，页面请求的是qcss文件，而不是传统的css文件：

而这个qcss文件原始代码是这样的：

全是一些CSS属性和属性值的缩写，但是，我们在demo页面查看qcss.test的请求数据，却是正统的CSS代码，截图示意如下：

同样的，也是Service Worker在浏览器背后编译的功劳，注册Service Worker的JS文件的核心代码如下：

```
var qcss=function(b){ /*qcss编译为css方法* /};

self.addEventListener('fetch', function(event) {
  event.respondWith(async function () {
    if (/\.qcss/.test(event.request.url)) {
      let res = await fetch(event.request);
      let text = await res.text();
      return new Response(qcss(text), {
        headers: {
          'content-type': 'text/css; charset=utf-8'
        }
      });
    }
    return fetch(event.request);
  }());
});
```

对吧，通俗易懂，一路了然。

这种在线编译有什么好处呢？例如省流量，test.qcss文件为7K多，而test.css文件则14K多，相比传统CSS文件传输体积小了50%，注意，这已经是压缩后的CSS再减小50%体积，根据自己测试，减小的体积在30%~50%之间（选择器越简洁压缩率越高）。

想想看，要是公司产品所有CSS都变成Qcss，传输小30%~50%，估计可以给公司省不少流量钱。

为此，我特意又写了个Node.js工具，可以把CSS文件转换成QCSS文件：https://github.com/zhangxinxu/gulp-qcss/blob/master/css2qcss.js

一个压缩器，一个解析器，外加Service Worker，就可以颠覆目前的CSS资源加载策略，对于一些无关紧要的小项目活动页面之类的，大家其实可以试试这种新技术，验证其价值。

关于QCSS解析及压缩，以后时机成熟我会专门介绍，这里，抛砖引玉，为大家的技术选项实践提供更开阔的思路。

三、Service Worker与浏览器直接JS编译

例如，我们可以把CoffeeScript直接搬到浏览器中，与上面类似，这里不进一步展开。

四、结束语

理论上，有了Service Worker，我们可以把很多很多Node.js的能力搬到客户端搬到浏览器中，不仅仅是离线缓存，不仅仅是模拟并伪造ajax请求数据，可谓是有了Service Worker，真的可以为所欲为。

有了技术能力，剩下的就是发挥我们的创造力和想象力，让我们的web开发策略更加繁荣昌盛。

还是那句话，本文旨在抛砖引玉，本文提供的两个demo的技术策略并不一定是最优解，也并不一定适合实际项目，尤其大型项目，可能会有风险。也欢迎大家集思广益，提供更好更佳的相关事件，我会在文中及时进行更新。

以上~

感谢阅读！

（本篇完） // 想要打赏？点击这里。有话要说？点击这里。

« 简单了解ES6/ES2015 Symbol() 方法

文章文档标题自动生成导航jQuery小插件titleNav.js »

猜你喜欢

- 万岁，浏览器原生支持ES6 export和import模块啦！
- 借助Service Worker和cacheStorage缓存及离线开发
- 了解JS中的全局对象window.self和全局作用域self
- Stylus-NodeJS下构建更富表现力/动态/健壮的CSS
- 高富帅seajs使用示例及spm合并压缩工具露脸
- JS一般般的网页重构可以使用Node.js做些什么

- 使用electron构建跨平台Node.js桌面应用经验分享
- windows系统下批量删除OS X系统.DS_Store文件

分享到： 1

标签： async, await, CSS压缩工具, fetch, nodejs, QCSS, Service Worker

发表评论（目前14条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. 小书匠说道：
2018年06月26日 02:08

666.学到了.

回复

2. 张 鑫旭说道：
2018年05月2日 17:47

兼容性貌似还存在点问题

回复

3. 你今天真好看说道：
2018年04月28日 16:42

嗯，看着你右上角的广告轮播图超烦人的，又不能关掉。
于是F12审查元素，把那一段代码删掉了。
嗯，这下干净了，继续看。

回复

4. 小媒体说道：
2018年04月27日 16:10

文章很不错，之前也想过，不过美中不足的是，SEO依然不行，百度不会解析JS部分，按照渲染完的内容去收录建立索引。

回复

5. kingShen说道：
2018年04月27日 14:27

get

回复

6. **im333说道：**
2018年04月27日 09:38

用着东西渲染，会影响百度收录吗

回复

7. **jiangwei说道：**
2018年04月26日 18:58

就怕移动端跑不动

回复

8. **da li说道：**
2018年04月26日 16:22

了解了….

回复

9. **iamakn说道：**
2018年04月26日 10:14

留下，学一会

回复

10. **GoStop说道：**
2018年04月25日 11:21

关注下，万一火了呢

回复

11. **郭子说道：**
2018年04月25日 10:30

https://zhangxinxu.github.io/https-demo/online-render/
https://zhangxinxu.github.io/https-demo/online-render/index.html

为什么第一个链接没有渲染。。

回复

> **张 鑫旭说道：**
> 2018年04月25日 23:17
>
> https://zhangxinxu.github.io/https-demo/online-render/index.html 完整URL访问，因为demo是判断的.html后缀处理的。
>
> 回复

12. **渣渣罗说道：**
2018年04月25日 00:32

赞

回复

13. **ferrinweb说道：**

2018年04月24日 23:22

错别字：
为了快速书写，我自己弄了个名为Qcss的编译工具，可以→讲←CSS声明自定义为缩写

回复

最新文章

» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门

» 常见的CSS图形绘制合集 (178)
» 粉丝群第1期CSS小测点评与答疑 (112)
» 未来必热：SVG Sprite技术介绍 (111)
» HTML5终极备忘大全（图片版+文字版） (85)
» 让所有浏览器支持HTML5 video视频标签 (83)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (80)
» CSS3下的147个颜色名称及对应颜色值 (78)
» 小tips: 纯CSS实现打字动画效果 (72)
» 写给自己看的display: flex布局教程 (69)
» 分享三个纯CSS实现26个英文字母的案例 (69)

今年热议

» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)
» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- 万岁，浏览器原生支持ES6 export和import模块啦！
- 借助Service Worker和cacheStorage缓存及离线开发
- 了解JS中的全局对象window.self和全局作用域self
- Stylus-NodeJS下构建更富表现力/动态/健壮的CSS
- 高富帅seajs使用示例及spm合并压缩工具露脸
- JS一般般的网页重构可以使用Node.js做些什么
- 使用electron构建跨平台Node.js桌面应用经验分享
- windows系统下批量删除OS X系统.DS_Store文件