Vue 2.X 浏览器兼容方案

问题描述

组件环境

- vue – 2.5.11
- vue-cli 使用模板 `webpack-simple`
- http请求：axios

Vue 官方对于 ie 浏览器版本兼容情况的描述是 ie9+，即是 ie9 及更高的版本。经过测试，Vue 的核心框架 `vuejs` 本身，以及生态的官方核心插件（VueRouter、Vuex等）均可以在 ie9 上正常使用。

Vue 作者尤雨溪对于 Vue 的学习建议 中有提及为了将项目更好的生态化/工程化，要尽可能学习及使用新的 ECMAScript 规范。目前 ES6/ES2015 是可用度和稳定度较高的规范，文档齐全，国内还有 阮一峰 《ECMAScript 6 入门》 做了大量的文档翻译，开发环境可谓完善。然而版本较旧的浏览器并不支持 es6 规范，尤其是 ie 浏览器，即使是最高的 ie11 版本，对于 es6 规范也支持得并不全。如此则需要对所有原生不支持 ES6 特性的浏览器做兼容性处理。

本文将针对使用 Vue 生态开发完成的网站，以 ie9 版本为基础兼容目标，实现全功能正常使用的全面兼容解决方案。

ES6兼容

在 ie9 的环境上，es6 的部分新对象、表达式，并不支持，解决方案是使用 `babel-polyfill` 组件，它可以将 es6 的代码翻译成低版本浏览器可以识别的 es5 代码

```
npm i babel-polyfill --save-dev
复制代码
```

安装完成后，在项目的主入口文件 `main.js` 的首行就可以直接引用

```
import 'babel-polyfill';
复制代码
```

在项目使用 `vue-cli` 生成的代码中，根目录有一个 `.babelrc` 文件，这是项目使用 babel 的配置文件。在默认生成的模板内容中，增加 `"useBuiltIns": "entry"` 的设置内容，这是一个指定哪些内容需要被 polyfill(兼容) 的设置

useBuiltIns 有三个设置选项

- `false` – 不做任何操作
- `entry` – 根据浏览器版本的支持，将 polyfill 需求拆分引入，仅引入有浏览器不支持的polyfill
- `usage` – 检测代码中 `ES6/7/8` 等的使用情况，仅仅加载代码中用到的 polyfill

这里推荐设置为 `entry` ，完整的 `.babelrc` 内容如下：

```
{
  "presets": [
    [
      "env",
      {
        "modules": false,
        "useBuiltIns": "entry"
      }
    ],
    "stage-3"
  ]
}
复制代码
```

加入这些代码后，工程里的大部分内容已可兼容到 ie9 版本

Number对象

即使在使用 `babel-polyfill` 做代码翻译后，发现还是有一些 es6 的新特性并没有解决，比如说 `Number` 对象的 `parseInt` 和 `parseFloat` 方法

es6 将全局方法 `parseInt()` 和 `parseFloat()` ，移植到 `Number` 对象上面，行为完全保持不变。

这样做的目的，是逐步减少全局性方法，使得语言逐步模块化。

解决这个问题不需要引入包来解决，同样在项目主入口文件 `main.js` 加入以下代码（代码尽可能靠前，最好是在引用 `babel-polyfill` 之后 ）

```
if (Number.parseInt === undefined) Number.parseInt = window.parseInt;
if (Number.parseFloat === undefined) Number.parseFloat = window.parseFloat;
复制代码
```

requestAnimationFrame方法

`window.requestAnimationFrame` 是浏览器用于定时循环操作的一个接口，类似于 `setTimeout`，主要用途是按帧对网页进行重绘。

`requestAnimationFrame` 的优势，在于充分利用显示器的刷新机制，比较节省系统资源。显示器有固定的刷新频率（60Hz或75Hz），也就是说，每秒最多只能重绘60次或75次，`requestAnimationFrame` 的基本思想就是与这个刷新频率保持同步，利用这个刷新频率进行页面重绘。此外，使用这个API，一旦页面不处于浏览器的当前标签，就会自动停止刷新。这就节省了CPU、GPU和电力。

不过有一点需要注意，`requestAnimationFrame` 是在主线程上完成。这意味着，如果主线程非常繁忙，`requestAnimationFrame` 的动画效果会大打折扣。

`window.requestAnimationFrame()` 方法告诉浏览器您希望执行动画并请求浏览器在下一次重绘之前调用指定的函数来更新动画。该方法使用一个回调函数作为参数，这个回调函数会在浏览器重绘之前调用。

有部分第三方组件就使用了这个方法，例如部分文件上传、图片处理类的组件；那么在这类型的组件在 ie9 下使用时，会报出

```
SCRIPT5007: Expected object.
复制代码
```

`window.requestAnimationFrame()` 的最低兼容 ie 版本为 10，那么在 ie9 上做兼容就需要制作 requestAnimationFrame polyfill

```
(function() {
    var lastTime = 0;
    var vendors = ['ms', 'moz', 'webkit', 'o'];
    for(var x = 0; x < vendors.length && !window.requestAnimationFrame; ++x)
{
        window.requestAnimationFrame =
window[vendors[x]+'RequestAnimationFrame'];
        window.cancelAnimationFrame =
window[vendors[x]+'CancelAnimationFrame']
                                   ||
window[vendors[x]+'CancelRequestAnimationFrame'];
    }

    if (!window.requestAnimationFrame)
        window.requestAnimationFrame = function(callback, element) {
            var currTime = new Date().getTime();
            var timeToCall = Math.max(0, 16 - (currTime - lastTime));
            var id = window.setTimeout(function() { callback(currTime +
timeToCall); },
              timeToCall);
            lastTime = currTime + timeToCall;
            return id;
        };
```

```
    if (!window.cancelAnimationFrame)
        window.cancelAnimationFrame = function(id) {
            clearTimeout(id);
        };
}());
复制代码
```

Gist：requestAnimationFrame polyfill
这部分代码同样是尽可能在网站入口处就执行

http网络请求(跨域)

在大多数的 Web 项目中（以 JavaWeb 为例），网站的页面和服务（至少是 controller 层）在同一个工程进行开发和部署，在大前端的新型模式下，我们建议尽可能对网站的前端和后端进行完全分离，前后端分离的好处和意义这里不再赘述。
既然是前后端分离，那么部署也必然是各自独立部署，不同的访问路径，就会产生跨域访问的问题（同一站点，不同端口号也是跨域）
在此设定背景情况：

- 服务端已完整开启 CROS 跨域支持
- http 组件使用 axios
- axios 设置 `withCredentials` 为 true 开启跨域访问时携带 cookie 数据

> 高版本浏览器（ie10+ 或 chrome, ff）仅需要完成背景情况中的功能，即可支持跨域数据请求功能

axios 进行数据请求时，默认使用 `XMLHttpRequest` 对象，在检测到当前请求是跨域访问时，axios 会测试浏览器是否支持 `XDomainRequest` 对象，若支持则优先使用。
ie8 / ie9 的 `XMLHttpRequest` 对象，不支持跨域访问，该对象在 ie10 后才原生支持跨域访问。微软的解决方案是在 ie8 / ie9 中提供了 XDomainRequest(XDR) 对象来进行解决跨域问题，虽然使用该对象可以跨域访问成功，并返回数据，但它却依然是一个功能不完整的半成品，它的使用有诸多限制：

- XDR 仅支持 GET 与 POST 两种请求方式
- XDR 不支持自定义的请求头，若服务端使用 `header` 的自定义参数进行做身份验证，则不可用
- 请求头的 `Content-Type` 只允许设置为 `text/plain`
- XDR 不允许跨协议的请求，如果网页在 HTTP 协议下，就只能请求 HTTP 协议下的接口，不能访问 HTTPS 接口
- XDR 只接受HTTP/HTTPS 的请求
- 发起请求的时候，不会携带 `authentication` 或 `cookies`

微软虽然提供了解决方案，但却是不折不扣的鸡肋，根本无法胜任系统中各种场景的数据请求需求，至此，axios 对 ie9 的跨域数据请求已无能为力。

完美解决方案：代理(proxy)

虽然 axios 对 ie9 跨域已无能为力，但前端项目打包的解决方案 `webpack` 提供了一个优雅而彻底解决问题的方式：**代理**
devServer.proxy

webpack 的 `devServer.proxy` 的功能是由 http-proxy-middleware 项目来实现的
实现原理是将目标位置的请求代理为前端服务本地的请求，既然是代理成为本地的请求，就不存在跨域的问题，axios 就会用回 `XMLHttpRequest` 对象进行数据请求，一切都恢复正常了，header、cookies、content-type、authentication 等内容都被正确传递到服务端。

项目中 webpack.config.js 的配置

```
devServer: {
    historyApiFallback: true,
    noInfo: true,
    overlay: true,
    proxy: {
        '/api': {
            target: 'http://localhost:8081/myserver',
            pathRewrite: {
                '^/api': ''
            }
        }
    }
}
复制代码
```

配置中指定了将 `http://localhost:8081/myserver` 服务的位置代理为本地前端服务的 `http://localhost:8080/api`。例如需要读取用户信息的原请求是 `http://localhost:8081/myserver/user/zhangsan`，代理后，就变为 `http://localhost:8080/api/user/zhangsan`。

即是 `/api` 的前缀代表了服务端，所以在使用 axios 时，需要对每个服务端请求都增加上 `/api`的前缀；通常在项目开发中，需要对数据请求组件 `axios` 进行二次封装，以达到统一设置默认参数，统一数据请求入口等目的，那么此时就只需要在二次封装的文件里统一调整请求前缀即可。

不过，webpack 的 `devServer.proxy` 仅在开发模式下可用，生产模式下无法使用。开发模式下，调试服务可以读取 `webpack.config.js` 中的配置内容进行实时代理，而项目在部署到生产环境前，需要将工程进行编译转换成静态的 js 文件，没有调试服务的支撑自然是无法进行请求代理的。

nginx 配置

虽然 `devServer.proxy` 的功能仅能工作于开发模式，那么在生产模式下，自然也是有解决方案的；通常 Vue 的项目在编译成最终的 js 文件后，仅需要静态服务器即可，这其中又以 `nginx` 为最优选择方案，轻量、高性能、高并发、反向代理服务等均为其优点，这里需要做的数据请求代理的功能就使用到了 nginx 的 **反向代理** 功能

`conf/nginx.conf` 文件配置增加以下内容

```
location /api/ {
    proxy_pass http://localhost:8081/myserver/;
}
复制代码
```

该配置同样是将 `http://localhost:8081/myserver/` 的目标服务端位置代理为本地服务的 `/api` 路径，如此，生产环境下的数据请求问题也得以解决