高级node第十三单元（6.16）

2020年6月16日 8:59

fetch的使用

关于前端路由

Fetch是什么

1. 是浏览器提供的原生的api

```
> fetch
< ƒ fetch() { [native code] }
```

2. 作用：就是想服务端发起http请求
3. 与XMLHttpRequest相比，接口更加友好，采用了promise的形式
4. 同样有跨域的问题

fetch如何使用

```
fetch('/login', {
  method: 'POST',
  body: JSON.stringify({username, password}),
  headers: {
    'Content-Type': 'application/json'
  }
}).then(response => {
  window.response = response;
 } );
```

Fetch(url, {
Method: 请求方法
Body: post请求的参数 <string> <formData> <Blob> <arrayBuffer>
Headers: 配置各种请求头
})
Return Promise<Response>
如果是get方式，但是想传递参数，需要自己在url地址中拼接

response的用如下

关于前端路由

1. 传统的前端界面发生变化
   a. 改变URL地址
   b. 浏览器会向服务器发送对应的请求
   c. 加载对应的页面，达到视图变化的效果
2. 在单页面时代
   a. 改变url地址
   b. 不向服务器发请求
   c. 通过前端js的逻辑检索对应的视图（组件）
   d. 进行组件的切换

url地址发生改变的方式

1. 手动在地址栏输入
   会发起请求
2. 点击a标签
   href的值是正常的路径 -会发起请求
   href的值是锚点链接-不会发请求
3. 使用location.href 或者 window.open
   会发起请求
4. 使用location.search
   会发起请求
5. 使用location.hash
   不会发起请求
6. history的相关方法 go back forward
   不会发起请求
7. 使用history.pushState()方法
   不会发起请求

```
> history.pushState('one', {name: 'one'}, '/one')
< undefined
> history.pushState('two', {name: 'two'}, '/two')
```

hash路由和history路由的实现细节

1. hash路由的实现
   通过location.hash改变url地址的锚点链接
   通过window.onhashchange事件去监听路由发生变化

2. history路由的实现
   通过调用history.pushState或者history.replaceState方法让路由发生变化的
   通过某一种方式去监听url地址发生了变化（）；
   1. 改变url地址同时向服务器发请求的方式不需要监听的
   2. History.go/back/forward可以通过window.onpopstate来监听
   3. 当调用pushState和replaceState的时候可以主动触发popstate事件

```
<div id="box">abc</div>
<script>
  const oBox = document.getElementById('box')
  const views = {
    '/abc': 'abc',
    '/bbc': 'bbc'
  }
  // histor.go back forward的情况
  window.onpopstate = showView
  function showView(){
    oBox.innerHTML = views[location.pathname];
  }
  function pushState(str, obj, pathname){
    history.pushState(str, obj, pathname)
    showView();
  }
  function replaceState(str, obj, pathname){
    history.replaceState(str, obj, pathname)
    showView();
  }
</script>
```

前端单页面应用部署到服务器

1. publicPath问题

```
// We interpret the `public path` (su
publicPath: paths.publicUrlOrPath,
// Point sourcemap entries to origin
```

控制的是url地址的前缀

2. 使用了history模式的路由出现的问题
   前端的路由一定要加publicPath前缀

3. 使用history模式路由，解决单页面刷新找不到页面的问题
   解决思路： 服务端接收到请求，判断是否是前端单页面配置的路由，如果是，则永远返回该页面，如果不是就按照后端正常的路由规则执行

   根据前端的publicPath去鉴定是否是前端单页面配置的路由

```
import url from 'url';
import fs from 'fs';
import path from 'path';
const publicPath = '/build';
export default () => {
  return async (ctx, next) => {
    const {pathname} = url.parse(ctx.url);
    if('/' + pathname.split('/')[1] === publicPath){
      // 以为是单页面的路由
      ctx.response.set('content-type', 'text/html');
      ctx.body = fs.readFileSync(path.join(__dirname, '../public/build/index.html'));
    }else{
      await next();
    }
  };
};
```