DAY28 promise封装ajax及应用

2023年5月25日 18:38

promise封装ajax

promise封装ajax

1.前面的封装利用回调函数获取异步的数据

2.按照顺序获取多条接口数据产生回调地狱

3.采用promise的来实现。

插件封装的步骤

1.根据ajax的四步曲，约定默认参数和配置参数 - Object.assign

2.限定用户的输入，提示相关的错误 - 判断

3.将回调函数改写成promise，通过promise获取异步数据 - 传参

4.返回promise对象，利用then获取对应的接口数据，利用catch捕获错误

瀑布流

一.瀑布流布局，又称瀑布流式布局。

是比较流行的一种网站页面布局，视觉表现为参差不齐的多栏布局，随着页面滚动条向下滚动，这种布局还会不断加载数据块并附加至当前尾部。

针对多图片展示

接口数据必须包括图片的高度

固定列的实现过程

1.绘制布局,包括样式的设定

2.获取数据接口，拿到数据

3.找到最小高度的那一列插入对应的图片或者文字，不断进行中

4.拖动滚动条加载下一页，直到所有的数据完成

```
<body>
  <div id="main">
    <div class="list"></div>
    <div class="list"></div>
    <div class="list"></div>
    <div class="list"></div>
    <div class="list"></div>
  </div>
</body>

<script src="./ajax.js"></script>
```

二.获取接口数据，渲染到对应的li元素中

```
const render = function () {
  const lists = document.querySelectorAll('.list');//5个list
  ajax_promise({
    url: './pbl.json',
```

```
    dataType: 'json'
  }).then(res => {
    console.log(res);
    // 渲染数据
    for (let item of res) {//渲染一张，追加一张
      let strHtml = '';
      strHtml += `
      <div>
        <img src="${item.src}" style="height:${item.height}"/>
        <p>图片的描述</p>
      </div>
    `;
      // 拼接完成，插入最小高度的list中。
      getMinList(lists).innerHTML += strHtml;
    }
  });
}
render();//执行函数
```

三.求最小高度的list，将渲染的结构插入

offsetHeight

思路：假设第一个是最小的，剩余的四个和假设的最小值进行比较，求得真正的最小高度的list

```
function getMinList(list) {//list参数：传入的5个div
  let minHeight = list[0].offsetHeight;//假设第一个是最小的高度
  let minIndex = 0;//假设最小高度div的索引。通过索引找对应的list
  for (let i = 1; i < list.length; i++) {
    if (list[i].offsetHeight < minHeight) {
      minHeight = list[i].offsetHeight;
      minIndex = i;//存储索引
    }
  }
  return list[minIndex];//最小高度的list
}
```

四.加载更多

如果高度最小的那一列最后的内容都显示在可视区(即将看到下面的空白)，加载第二页，依此类推

思路

如果存在可视区的高度 + 滚动条的top值 > 其中一个list的高度，说明准备加载

```
window.onscroll = function () {
  const lists = document.querySelectorAll('.list');//5个list
  let clientH = document.documentElement.clientHeight;//可视区的高度
  let scrollT = document.documentElement.scrollTop;//滚动条的卷去的尺寸(滚动条的top值)
  let minHeightList = getMinList(lists).offsetHeight;// 最小高度的list
  if (minHeightList < clientH + scrollT) {
    render();
```

```
  }
};
```

事件轮询

1.同步和异步

浏览器相关，JavaScript依赖浏览器进行解析，JavaScript单线程，只有一个主线程，一次只能干一件事情。

同步：阻塞模式，同步代码进入主线程，按照顺序执行，后一个任务等前一个任务完成才执行。

异步：非阻塞模式，异步代码先进入任务队列，等待主线程的通知，进入主线程执行。

梳理JavaScript的异步语法

定时器

回调函数，事件包含异步

ajax异步接收响应

promise

async await

2.队列里面的异步代码分为：宏任务和微任务

宏任务：整个script代码(注意)，定时器，I/O(input/output)输入和输出

微任务：primise...then...（promise里面的代码是同步的，then里面的代码是异步的)

3.事件轮询的概念

重点：因为整个script代码是宏任务，必须从宏任务开始，由此开始实现宏任务和微任务的划分。

概念：执行一次宏任务，清空所有的微任务，循环往复，直到所有的任务都被清空(执行)

通俗的理解：事件轮询，轮询循环宏任务和微任务，直到所有的任务都被清空(执行)

```
// 案例:
setTimeout(function () {
  console.log('1111')
}, 0);
new Promise(function (resolve, reject) {//同步
  console.log("2222");
  resolve();
}).then(function () {
  console.log('3333')
})
console.log("4444");//同步
// 解析:
// 1.宏任务script开始执行，同步先走，进行宏任务和微任务的划分
// 同步先走
// console.log("2222");//同步
// console.log("4444");//同步
// 2.进行宏任务和微任务的划分
// 宏任务
```

```
// console.log('1111')
// 微任务  比上面的宏任务要快
// console.log('3333')
// 3.结果:
// 2222, 4444, 3333, 1111



// 案例
new Promise(function (resolve, reject) {
  resolve();
}).then(function () {
  console.log("111");
  return new Promise(function (resolve, reject) {
    resolve();
  })
}).then(function () {
  console.log("222");
})
new Promise(function (resolve, reject) {
  resolve();
}).then(function () {
  console.log("333");
})
// 解析:
// 1.宏任务script开始执行，同步先走，进行宏任务和微任务的划分
// 2.进行宏任务和微任务的划分
// 宏任务
// 微任务
// console.log("111");   存在微任务 console.log("222");
// console.log("333");
// 3.结果:
// 111,333,222



// 案例:
new Promise((resolve) => {
  resolve(
    (() => {
      console.log(1);
      return 2;
    })()
  );
  console.log(3);
}).then((arg) => {
  console.log(arg);
});
setTimeout(() => {
  console.log(4);
}, 0);
console.log(5);
```

```
// 解析:
// 1.宏任务script开始执行，同步先走，进行宏任务和微任务的划分
// console.log(1);
// console.log(3);
// console.log(5);
// 2.进行宏任务和微任务的划分(划分完成，立刻执行微任务)
// 宏任务
// console.log(4);
// 微任务
// console.log(2);

// 3.结果:
// 1, 3, 5, 2, 4



// 案例:
console.log(1);
setTimeout(() => {
  console.log(2);
}, 0);
new Promise((resolve) => {
  console.log(3);
  resolve();
}).then(() => {
  console.log(4);
});
setTimeout(() => {
  console.log(5);
  new Promise((resolve) => {
    console.log(6);
    setTimeout(() => {
      console.log(7);
    });
    resolve();
  }).then(() => {
    console.log(8);
  });
}, 500);
new Promise((resolve) => {
  console.log(9);
  resolve();
}).then(() => {
  console.log(10);
  setTimeout(() => {
    console.log(11);
  }, 0);
});
console.log(12);
// 解析:
// 1.宏任务script开始执行，同步先走，进行宏任务和微任务的划分
// console.log(1);
// console.log(3);
```

```
// console.log(9);
// console.log(12);
// 2.进行宏任务和微任务的划分(划分完成，立刻执行微任务)
// 宏任务
// 微任务
// 3.结果:
// 1, 3, 9, 12, 4, 10, 2, 11, 5, 6, 8, 7

// 3.结果:
// 第一次执行微任务的结果：1,3,9,12,4,10
// 第二次执行宏任务的结果：1,3,9,12,4,10,2    目前执行了一次宏任务，继续清空微任务，但是没有微任务
// 第三次执行宏任务的结果：1,3,9,12,4,10,2,11    目前执行了一次宏任务，继续清空微任务，但还是没有微任务
// 第四次执行宏任务的结果：1,3,9,12,4,10,2,11,5    目前执行了一次宏任务，存在微任务，继续微任务
// 第五次执行微任务的结果：1,3,9,12,4,10,2,11,5,6,8    目前执行了一次微任务，清空微任务，继续执行宏任务。
// 第六次执行宏任务的结果：1,3,9,12,4,10,2,11,5,6,8,7  目前执行了一次宏任务，继续清空微任务，但还是没有微任务，继续执行宏任务，没有宏任务，继续清空微任务才结束。
```