fetch & promise

今日目标

1. 熟练使用 fetch api 发送请求
2. 完成头像上传，理解 content-type
3. 能够使用 Promise 封装异步 api
4. 完成图书管理项目

fetch api

提供了一种简单，合理的方式来跨网络异步获取资源，相比 xhr 更加简单

fetch 请求流程

1. 调用 `fetch` 函数，传入 url 作为第一个参数，`{headers, body}` 作为第二个参数
2. `.then(cb)` 传入函数获取请求结果
3. `.catch(cb)` 传入函数处理请求异常

```
// url, 请求配置
fetch("http://ajax-api.itheima.net/api/login", {
  // 请求方法
  method: "POST",
  // 请求数据
  body: JSON.stringify({ username: "admin", password: "123456" }),
})
  .then(() => {
    alert("登录成功");
  })
  .catch(() => {
    alert("登录失败");
  });

// 默认为 GET 请求
fetch("http://ajax-api.itheima.net/api/news")
  .then((res) => {
    // 请求成功
    console.log(res);
```

```
  })
  .catch((err) => {
    // 请求失败
    console.err(err);
  });
```

头像上传

头像上传地址修改为：https://autumnfish.cn/api/form/upload

利用 res.data 获取结果

```
<div id="result">请上传头像</div>
<input
  type="file"
  id="fileInput"
  placeholder="上传头像"
  accept="image/*"
  style="display: none"
/>
<button id="upload">上传文件</button>
<script>
  const fileInput = document.querySelector("#fileInput");
  const upload = document.querySelector("#upload");
  const result = document.querySelector("#result");
  upload.onclick = () => {
    fileInput.click();
  };
  fileInput.onchange = (e) => {
    console.log(e.target.files);
    const file = e.target.files[0];
    const formData = new FormData();
    formData.append("avatar", file);
    e.target.value = null;
    fetch("https://autumnfish.cn/api/form/upload", {
      method: "POST",
      body: formData,
    })
      .then((res) => res.json())
      .then((res) => {
        result.innerHTML = `<img src="${res.data}"/>`;
      });
  };
</script>
```

content-type

用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件

常见的媒体格式类型如下：

- text/html： HTML 格式
- text/plain：纯文本格式
- text/xml： XML 格式
- image/gif：gif 图片格式
- image/jpeg：jpg 图片格式
- image/png：png 图片格式

以 application 开头的媒体格式类型：

- application/xhtml+xml：XHTML 格式
- application/xml： XML 数据格式
- application/atom+xml：Atom XML 聚合格式
- application/json： JSON 数据格式
- application/pdf：pdf 格式
- application/msword： Word 文档格式
- application/octet-stream： 二进制流数据（如常见的文件下载）
- application/x-www-form-urlencoded： `<form encType=””>`中默认的 encType，form 表单数据被编码为 key/value 格式发送到服务器（表单默认的提交数据的格式）

另外一种常见的媒体格式是上传文件之时使用的：

- multipart/form-data： 需要在表单中进行文件上传时，就需要使用该格式

Promise

Promise-前提知识

> 为了更好的理解`Promise`的作用,咱们需要先**同步 2 个概念**,**异步函数**和**回调函数**,以及多个**回调函数嵌套**带来的问题

异步函数

1. **异步函数**的执行，由于是异步的，不会阻塞**主线程代码**的执行
2. 常见异步函数(触发时机分别是?)
    i. `setTimeout`
    ii. `setInterval`
    iii. `XMLHttpRequest`

回调函数

1. 把一个**函数当成参数**传递, **将来**特定的时机**调用**, 这个函数就叫**回调函数。**

    i. 把函数当作参数传递,被传递的那个函数--->**回调函数**

2. 指出下列代码中是否使用了**回调函数**,如果**是**,那么哪个是**回调函数**?

```
setInterval(() => {
  console.log("123");
}, 1000);

// load 事件，请求完成时触发
xhr.addEventListener("load", () => {
  console.log(xhr.response);
});
```

3. 大部分有**回调函数**的地方,都会涉及到**异步函数**

回调函数嵌套

多个异步操作**彼此依赖**,所产生的**嵌套代码**

比如:

1. 1 秒之后打印 1
2. 1 打印之后,等待 2 秒打印 2
3. 2 打印之后,等待 3 秒打印 3
4. .....(写出来看看)

```
setTimeout(() => {
  console.log(1);
  setTimeout(() => {
    console.log(2);
    setTimeout(() => {
```

```
      console.log(3);
    }, 3000);
  }, 2000);
}, 1000);
```

函数能够访问当前函数变量以及更高层级变量

回调地狱：嵌套太多之后，变量冲突，变量的引用（可以异步变化）和值（不可以异步变化）无法区分

Promise-概念及基本使用

> `Promise`可以用来解决上一节**回调函数嵌套**的问题,咱们来看看如何**使用它**

概念

1. `Promise` 是一个对象，它代表了一个**异步操作**的**最终完成或者失败。**
2. `fetch`就是基于`Promise`
3. 基于**Promise**之后的约定:
   i. 通过`then`获取成功结果
   ii. 通过`catch`获取失败结果
   iii. 可以用**链式的方式**处理多个彼此依赖的异步操作
      a. 一路点下去
      b. `arr.map().filter().map().filter().join()`
      c. 上一个方法的返回值,可以继续点出后续的方法
4. 大部分异步函数最新的`api`都有`Promise`的版本,有一些可能需要自己封装,比如(**setTimeout**)

使用步骤

```
// 1. 创建 并传入回调函数
const p = new Promise((resolve, reject) => {
  // 内部一般封装异步的操作
  // 成功执行 resolve
  // 失败执行 reject
});
// 2. 使用
p.then((res) => {}) // resolve的值
  .catch((err) => {}); // reject的值
```

代码解析:

```
// 原生Promise的写法
// const p = new Promise(function (resolve, reject) {})
// new Promise时传入的回调函数中的 2个参数
//  resolve reject
// Promise会传入2个具体的函数进来
// 内部根据异步执行的结果 触发对应的函数 即可 -->成功/失败
//  resolve   reject (形参) 名字可以改,但是建议别改
const p = new Promise((resolve, reject) => {
  console.log("resolve:", resolve);
  console.log("reject:", reject);
  // 写异步的代码
  // 根据成功  resolve then
  setTimeout(() => {
    // resolve('成功 ')
    reject("哎呀,失败");
  }, 1000);
  // 根据失败 reject catch
});

// console.log('p:', p)
p.then((res) => {
  console.log("res:", res);
}).catch((err) => {
  console.log("err:", err);
});
```

Promise-抽取（封装）

> 上一节的链式调用中有大量重复的**创建 Promise 对象**的语法,为了简化调用,咱们抽取一下

需求:

1. 抽取一个方法,调用返回`Promise`对象
2. 接收延迟的**时间**,到时之后`then`中可以获取到`延迟了 xx 秒执行`

```
function wait(delay) {
  return new Promise((resolve, reject) => {
    // 代码略
  });
}
```

```
// 运行效果
wait(2000)
  .then((res) => {
    console.log(res); // 延迟了2秒执行
    return wait(3000);
  })
  .then((res) => {
    console.log(res); // 延迟了3秒执行
  });
```

分析:

1. 延迟执行代码使用`setTimeout`
2. `setTimeout`设置延迟单位是? 毫秒
3. 内部执行哪个方法,对应到外部的`then` resolve
4. `then`中如何获取到数据? 调用 resolve 后，then 中的函数被调用

Promise 案例

```
function wait(n) {
  // setTimeout 没有 reject
  return new Promise((resolve, _) => {
    setTimeout(() => {
      resolve();
    }, n);
  });
}

function requestNews() {
  return new Promise((res, rej) => {
    const xhr = new XMLHttpRequest();
    xhr.open("GET", "http://ajax-api.itheima.net/api/news");
    xhr.send();
    xhr.responseType = "json";
    xhr.onreadystatechange = () => {
      if (xhr.readyState === 4) {
        if (xhr.status === 200) {
          res(xhr.response);
        } else {
          rej(xhr.status);
        }
      }
    };
```

```
  });
}

wait(1000)
  .then(() => {
    return requestNews();
  })
  .then((res) => {
    console.log(res);
  });
```

axios 做了 xhr 的 promise 封装，可做了解，用法与 fetch 相似，但是兼容性更强

图书管理项目

引入 bootstrap cdn:

> cdn 可以直接链接在 html 上的第三方代码，**通过网络 url 引入**

```
<head>
  <link
    href="https://cdn.bootcdn.net/ajax/libs/twitter-
bootstrap/5.2.2/css/bootstrap.css"
    rel="stylesheet"
  />
  <script src="https://cdn.bootcdn.net/ajax/libs/twitter-
bootstrap/5.2.2/js/bootstrap.js"></script>
</head>
```

bootstrap 弹框：

```
const modal = new bootstrap.modal(document.querySelector("#someModal"));

// 显示弹框
modal.show();
// 隐藏弹框
modal.hide();

// 弹框显示时，内容节点必须存在
someNode.innerHTML = "<div>innerHTML修改的内容</div>";
// setTimeout 0 -> 等待视图生效
setTimeout(() => {
  modal.show();
```

```
}, 0);
// 或者用上文中的 wait
wait(0).then(() => {
  modal.show();
});
```

全部代码

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <!-- 引入 bootstrap 样式表 -->
    <link
      href="https://cdn.bootcdn.net/ajax/libs/twitter-
bootstrap/5.2.2/css/bootstrap.css"
      rel="stylesheet"
    />
    <style>
      body {
        padding-top: 50px;
      }
    </style>
  </head>

  <body>
    <!-- 栅格系统 -->
    <div class="container">
      <div class="d-flex justify-content-between align-items-center">
        <h1>图书管理</h1>
        <button
          class="btn btn-success btn-sm"
          data-bs-toggle="modal"
          data-bs-target="#addModal"
        >
          添加
        </button>
      </div>
      <table
        class="table table-bordered table-striped table-dark table-hover text-
center"
```

```
    >
      <thead>
        <!-- 表头行 -->
        <tr>
          <th scope="col">Id</th>
          <th scope="col">书名</th>
          <th scope="col">作者</th>
          <th scope="col">出版社</th>
          <th scope="col">操作</th>
        </tr>
      </thead>
      <tbody>
        <!-- 表格中的每一行 -->
        <tr>
          <th scope="row">xxx</th>
          <td>xxx</td>
          <td>xxx</td>
          <td>xxx</td>
          <td>
            <button type="button" class="btn btn-link btn-sm btn-delete">
              删除
            </button>
            <button type="button" class="btn btn-link btn-sm btn-update">
              编辑
            </button>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
  <!-- add Modal -->
  <div class="modal fade" id="addModal">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">添加图书</h5>
          <button
            type="button"
            class="btn-close"
            data-bs-dismiss="modal"
            aria-label="Close"
          ></button>
        </div>
        <div class="modal-body">
          <form id="addForm" class="p-3">
            <!-- 书名 -->
            <div class="mb-3">
```

```
            <label class="form-label">书名</label>
            <input
              type="text"
              name="bookname"
              class="form-control"
              placeholder="请输入图书名称"
              name="bookname"
            />
          </div>
          <!-- 作者 -->
          <div class="mb-3">
            <label class="form-label">作者</label>
            <input
              type="text"
              name="author"
              class="form-control"
              placeholder="请输入作者名字"
              name="author"
            />
          </div>
          <!-- 出版社 -->
          <div class="mb-3">
            <label class="form-label">出版社</label>
            <input
              type="text"
              name="publisher"
              class="form-control"
              placeholder="请输入出版社名称"
              name="publisher"
            />
          </div>
        </form>
      </div>
      <div class="modal-footer">
        <button
          type="button"
          class="btn btn-secondary"
          data-bs-dismiss="modal"
        >
          取消
        </button>
        <button type="button" class="btn btn-primary" id="addBtn">
          确认
        </button>
      </div>
    </div>
  </div>
```

```
    </div>
    <!-- add Modal -->
    <div class="modal fade" id="editModal">
      <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">编辑图书</h5>
            <button
              type="button"
              class="btn-close"
              data-bs-dismiss="modal"
              aria-label="Close"
            ></button>
          </div>
          <div class="modal-body">
            <form id="editForm" class="p-3">
              <input type="hidden" name="id" />
              <!-- 书名 -->
              <div class="mb-3">
                <label class="form-label">书名</label>
                <input
                  type="text"
                  name="bookname"
                  class="form-control"
                  placeholder="请输入图书名称"
                  name="bookname"
                />
              </div>
              <!-- 作者 -->
              <div class="mb-3">
                <label class="form-label">作者</label>
                <input
                  type="text"
                  name="author"
                  class="form-control"
                  placeholder="请输入作者名字"
                  name="author"
                />
              </div>
              <!-- 出版社 -->
              <div class="mb-3">
                <label class="form-label">出版社</label>
                <input
                  type="text"
                  name="publisher"
                  class="form-control"
                  placeholder="请输入出版社名称"
```

```
                  name="publisher"
                />
              </div>
            </form>
          </div>
          <div class="modal-footer">
            <button
              type="button"
              class="btn btn-secondary"
              data-bs-dismiss="modal"
            >
              取消
            </button>
            <button type="button" class="btn btn-primary" id="editBtn">
              确认
            </button>
          </div>
        </div>
      </div>
    </div>
    <!-- bootstrap的js -->
    <script src="https://cdn.bootcdn.net/ajax/libs/twitter-
bootstrap/5.2.2/js/bootstrap.js"></script>
    <script>
      const baseUrl = "http://ajax-api.itheima.net/api";

      // 1. 获取所有的图书列表数据
      // 2. 循环渲染图书的表格结构
      function initBookList() {
        return fetch(baseUrl + "/books", {
          method: "GET",
        })
          .then((res) => res.json())
          .then((res) => {
            // 渲染图书的列表结构
            renderBookList(res.data);
          });
      }

      // 根据数组循环渲染图书的列表数据
      function renderBookList(list) {
        // 1. 声明一个空数组 rows
        const rows = [];
        // 2. 循环 list 数组，创建每个 tr 行，并 push rows 中
        list.forEach((item) => {
          rows.push(`<tr>
              <th scope="row">${item.id}</th>
```

```
              <td>${item.bookname}</td>
              <td>${item.author}</td>
              <td>${item.publisher}</td>
              <td>
                <button type="button" class="btn btn-link btn-sm btn-delete"
data-id="${item.id}">删除</button>
                <button type="button" class="btn btn-link btn-sm btn-update"
data-id="${item.id}">编辑</button>
              </td>
            </tr>`);
        });
        // 3. 把 rows 渲染到页面的 tbody 中
        document.querySelector("tbody").innerHTML = rows.join("");
      }

      // 调用初始化图书列表数据的方法
      initBookList();

      function getFormData(form) {
        // 快速获取到，要提交给服务器的数据
        const data = new FormData(form);
        return {
          bookname: data.get("bookname"),
          author: data.get("author"),
          publisher: data.get("publisher"),
          id: data.get("id") || undefined,
        };
      }

      // 1. 为 form 表单绑定 submit 提交事件
      // 2. 阻止表单的默认提交行为
      // 3. 把表单采集到的数据，通过 Ajax 提交给服务器
      const addForm = document.querySelector("#addForm");
      const addModal = new
bootstrap.Modal(document.querySelector("#addModal"));
      document.querySelector("#addBtn").addEventListener("click", function (e)
{
        // 发起请求，新增一本图书
        fetch(baseUrl + "/books", {
          method: "POST",
          body: JSON.stringify(getFormData(addForm)),
          headers: {
            "content-type": "application/json",
          },
        })
          .then((res) => res.json())
          .then((res) => {
```

```
        // 添加成功
        // 1. 重新渲染图书列表的数据
        initBookList();
        addForm.reset();
        addModal.hide();
      });
    });

    // 实现删除的功能
    // 1. 通过事件委托，给 tr 行中的删除按钮，绑定 click 事件
    // 2. 获取到点击的这一行的图书的 id
    // 3. 调用删除的接口，根据 id 删除指定的图书
    // 4. 删除成功之后，重新请求列表的数据
    document.querySelector("tbody").addEventListener("click", function (e) {
      if (e.target.classList.contains("btn-delete")) {
        // 获取自定义属性的值
        const id = e.target.dataset.id;
        // 调接口删除指定的图书
        fetch(baseUrl + "/books/" + id, { method: "DELETE" }).then((res) =>
{
          alert("删除成功");
          // 删除成功之后，刷新列表数据
          initBookList();
        });
      }
    });

    // 实现修改功能
    // 1. 打开对话框，回显对应的图书信息
    const eidtForm = document.querySelector("#editForm");
    const editModal = new bootstrap.Modal(
      document.querySelector("#editModal")
    );
    document.querySelector("tbody").addEventListener("click", function (e) {
      if (e.target.classList.contains("btn-update")) {
        fetch(baseUrl + "/books/" + e.target.dataset.id, {
          method: "GET",
        })
          .then((res) => res.json())
          .then((res) => {
            console.log(res);
            for (const key in res.data) {
              eidtForm.querySelector(`[name=${key}]`).value = res.data[key];
            }
          });
        editModal.show();
      }
```

```
      });
      // 2. 绑定点击事件，提交修改信息
      document.querySelector("#editBtn").addEventListener("click", () => {
        // 发起请求，修改图书
        const data = getFormData(editForm);
        fetch(baseUrl + "/books/" + data.id, {
          method: "PUT",
          body: JSON.stringify(data),
          headers: {
            "content-type": "application/json",
          },
        })
          .then((res) => res.json())
          .then((res) => {
            // 添加成功
            // 1. 重新渲染图书列表的数据
            initBookList();
            eidtForm.reset();
            editModal.hide();
          });
      });
    </script>
  </body>
</html>
```