如何学习js

- **jsx**
  - 特殊的代码需要babel 进行转换成Jsx,让浏览器进行解析
- **三元运算符**
  - I. 条件（三元）运算符是 JavaScript 唯一使用三个操作数的运算符：一个条件后跟一个问号（?），如果条件为真值，则执行冒号（:）前的表达式；若条件为假值，则执行最后的表达式
    ```
    function getFee(isMember) {
      return isMember ? '$2.00' : '$10.00';
    }
    ```
  - II. condition ? exprIfTrue : exprIfFalse
    - I. condition
      计算结果用作条件的表达式。
    - II. exprIfTrue
      如果 condition 的计算结果为真值（等于或可以转换为 true 的值），则执行该表达式。
    - III. exprIfFalse
      如果 condition 为假值（等于或可以转换为 false 的值）时执行的表达式。
  - 
    处理 null 值
    一个常见的用法是处理可能为 null 的值：
    ```
    const greeting = (person) => {
      const name = person ? person.name : "stranger";
      return `Howdy, ${name}`;
    };

    console.log(greeting({ name: "Alice" })); // "Howdy, Alice"
    console.log(greeting(null)); // "Howdy, stranger"
    ```
    III. 子主题 3
- **javascript 模块化**
  - 模块化的背景
    - JavaScript 程序本来很小——在早期，它们大多被用来执行独立的脚本任务，在你的 web 页面需要的地方提供一定交互，所以一般不需要多大的脚本。过了几年，我们现在有了运行大量 JavaScript 脚本的复杂程序，还有一些被用在其他环境（例如 Node.js）。

      复杂的项目需要一种将 JavaScript 程序拆分为可按需导入的单独模块的机制。Node.js 已经提供这个能力很长时间了，还有很多的 JavaScript 库和框架已经开始了模块的使用（例如，CommonJS 和基于 AMD 的其他模块系统，如 RequireJS、webpack 和 Babel）。
      - 模块化的背景
  - 例子
    - index.html
      main.js
      modules/
          canvas.js
          square.js
    - 导出模块的功能
      - 为了使用,需要先导出来
        ```
        export const name = "square";

        export function draw(ctx, length, x, y, color) {
          ctx.fillStyle = color;
          ctx.fillRect(x, y, length, length);

          return { length, x, y, color };
        }
        ```
    - 你能够导出函数 var、let、const 和等会看到的类。export 要放在最外层；比如你不能够在函数内使用 export。

      更方便的导出模块的方法是，在模块文件末尾使用一个 export 语句，以花括号括起来并用逗号分隔的形式列出所有需导出的功能。比如：
      - export { name, draw, reportArea, reportPerimeter };
        - export
    - 导入功能到你的脚本
      - import { name, draw, reportArea, reportPerimeter } from "./modules/square.js";
        - import
      - 导入功能到你的脚本
      - 子主题 3
      - 模块标识符提供一个 JavaScript 环境可以解析为模块文件路径的字符串。在浏览器中，它可以是一个相对于站点根目录的路径，对于我们的 basic-modules 示例来说是 /js-examples/module-examples/basic-modules。但是，这里我们使用点（.）语法来表示"当前位置"，然后紧跟着我们想要找的文件的相对路径。这比每次都要写下整个绝对路径要好得多，因为相对路径更短，并且使 URL 可移植——如果你将其移动站点目录中的其他位置，该示例仍然有效。
        - 模块标识符 .表示当前目录
  - 使用导入映射导入模块
    - 
      使用导入映射导入模块
      上面我们看到浏览器如何使用模块标识符（可以为绝对 URL，或使用文档的基础 URL 解析的相对 URL）导入模块：
      ```
      import { name as squareName, draw } from "./shapes/square.js";
      import { name as circleName } from "https://example.com/shapes/circle.js";
      ```
    - 
      例如，下面导入映射中的 imports 键定义了一个"模块标识符映射"JSON 对象，其中属性名称可以用作模块标识符，当浏览器解析模块 URL 时，相应的值将被替换。这些值必须是绝对或相对 URL。使用文档包含导入映射的基础 URL 将相对 URL 解析为绝对 URL。
      ```
      <script type="importmap">
        {
          "imports": {
            "shapes": "./shapes/square.js",
            "shapes/square": "./modules/shapes/square.js",
            "https://example.com/shapes/square.js": "./shapes/square.js",
            "https://example.com/shapes/": "/shapes/square/",
            "../shapes/square": "./shapes/square.js"
          }
        }
      </script>
      ```
      使用导入映射导入模块
      - 导入映射是在一个 <script> 元素中定义的 JSON 对象，type 属性设置为 importmap。文档中只能有一个导入映射，因为它用于解析静态和动态导入的模块，所以必须在导入模块的任何 <script> 元素之前声明。请注意，导入映射仅适用于文档——规范不涵盖如何在 worker 或 worklet 上下文中应用导入映射
    - 使用导入映射导入模块
    - 使用导入映射导入模块
  - 导出的两种方式
    - 具名导出
      - 子主题 1
    - 默认导出
      - export default randomSquare;
      - 
        我们可以把 export default 放到函数前面，定义它为一个匿名函数，像这样：
        ```
        export default function (ctx) {
          // ...
        }
        ```
        在我们的 main.js 文件中，我们通过以下代码导入默认函数：
        ```
        import randomSquare from "./modules/square.js";
        ```
        同样，没有大括号，因为每个模块只允许有一个默认导出，我们知道 randomSquare 就是需要的那个。上面的那一行相当于下面的缩写：
        ```
        import { default as randomSquare } from "./modules/square.js";
        ```
        子主题 2
    - 重命名
      - 
        重命名导入与导出
        在 import 和 export 语句的大括号中，可以使用 as 关键字为功能指定新名称，从而更改在顶级模块中使用的标识名称。
        例如，以下方法虽然方式略有不同，但可以完成相同的工作：
        ```
        // module.js 中
        export { function1 as newFunctionName, function2 as anotherNewFunctionName };

        // main.js 中
        import { newFunctionName, anotherNewFunctionName } from "/modules/module.js";
        ```
        ```
        // module.js 中
        export { function1, function2 };

        // main.js 中
        import {
          function1 as newFunctionName,
          function2 as anotherNewFunctionName,
        } from "./modules/module.js";
        ```
        子主题 1
      - 
        创建模块对象
        上述方法虽然有效，但有些冗长和混乱。一个更好的解决方案是，将每一个模块功能导入到一个模块功能对象中。可以使用以下语法形式：
        ```
        import * as Module from "/modules/module.js";
        ```
        这将获取 module.js 中所有可用的导出，并使它们可以作为对象模块的成员使用，从而有效地为其提供自己的命名空间。例如：
        ```
        Module.function1();
        Module.function2();
        ```
        再次，让我们看一个真实的例子。如果你转到我们的 module-objects 目录，将再次看到相同的示例，但利用上述的新语法进行重写。在模块中，导出都是以下简单形式：
        ```
        export { name, draw, reportArea, reportPerimeter };
        ```
        创建模块对象
      - 
        另一方面，导入看起来像这样：
        ```
        import * as Canvas from "./modules/canvas.js";

        import * as Square from "./modules/square.js";
        import * as Circle from "./modules/circle.js";
        import * as Triangle from "./modules/triangle.js";
        ```
        在每种情况下，你现在可以访问指定对象名称下面的模块导入。
        ```
        const square1 = Square.draw(myCanvas.ctx, 50, 50, 100, "blue");
        Square.reportArea(square1.length, reportList);
        Square.reportPerimeter(square1.length, reportList);
        ```
        因此，你现在可以像以前一样编写代码（只要你在需要时包含对象名称），并且导入更加整洁。
        子主题 3
- **模版字面量**
  - 使用(`) 分割的字面量，用于多行字符串，表达式，字符串插值的一种带标签模版的特殊结构
    ```
    `string text`

    `string text line 1
     string text line 2`

    `string text ${expression} string text`

    tagFunction`string text ${expression} string text`
    ```
  - 模版字面量转义
    ```
    `\``
    `\${}` 转义后 插值表达式就不会生效
    ```
  - 这有助于构建结构，并便于您以后轻松查找笔记。
- **React三个概念**
  - components
    - 允许构建组合使用的的代码片段,类似于乐高
  - props
    - 父组件传递给子组件信息
      - 使用prop来显示信息
      - title=react,解析出来，recat.title,需要使用{}解析
        - 
          子主题 1
      - 或使用.模版输出法进行解析
      - array.map迭代数组
        - 
          子主题 1
  - state
    - 维持用户的状态
      - 事件
        - 在 React 中，事件名称是驼峰式的。onClick 事件是可用于响应用户交互的众多可能事件之一。例如，您可以对输入字段使用 onChange，或对表单使用 onSubmit。
      - hook
        - React 有一组称为 hooks 的函数。Hook 允许你向组件添加额外的逻辑，比如 state。您可以将状态视为 UI 中随时间变化的任何信息，通常由用户交互触发
        - 您可以使用 state 来存储和增加用户单击 "Like" 按钮的次数。事实上，用于管理 state 的 React 钩子是：useState（）
        - 将 useState（）添加到你的项目中。它返回一个数组，你可以使用数组解构在组件中访问和使用这些数组值：
        - props与state的区别
          - 与作为第一个函数参数传递给组件的 props 不同，状态是 initiate 并存储在组件中。你可以将 state 信息作为 props 传递给子组件，但更新 state 的逻辑应该保留在最初创建 state 的组件中。
    - 子主题 8
      - 你可以了解更多关于在 React 应用程序中管理 state 和数据流的信息
  - 识别核心思想对于高效捕捉精要至关重要。
- **从recat 构建**
  - next js 属于React的框架
    - 虽然 React 擅长构建 UI，但要将该 UI 独立构建成功能齐全的可扩展应用程序确实需要一些工作。还有一些较新的 React 功能，如 Server 和 Client Components，需要框架。好消息是 Next.js 可以处理大部分设置和配置，并具有额外的功能来帮助您构建 React 应用程序。
  - 构建Npm
    - npm install react@latest react-dom@latest next@latest
      - next
      - react
      - react-dom
  - nextjs 有一个编译器可以将 jsx 转换为浏览器理解的 javascript
- **nextjs**
  - 文件路由系统
    - Next.js 使用文件系统路由。这意味着，您可以使用文件夹和文件，而不是使用代码来定义应用程序的路由。
  - 服务器和客户端组件
    - 客户端
      - 客户端是指用户设备上向服务器发送应用程序代码请求的浏览器。然后，它将从服务器接收到的响应转换为用户可以与之交互的接口
    - 服务端
      - 服务器是指数据中心中的计算机，用于存储您的应用程序代码、接收来自客户端的请求、执行一些计算并发回适当的响应。
    - 子主题 3
      - 在幕后，组件被拆分为两个模块图。服务器模块图（或树）包含在服务器上渲染的所有服务器组件，客户端模块图（或树）包含所有客户端组件。
        - 子主题 1
          - 服务器组件渲染后，一种称为 React 服务器组件有效负载 （RSC） 的特殊数据格式被发送到客户端。RSC 有效负载包含：
  - 客户端组件
    - 此功能称为 Fast Refresh。它会为您提供有关您所做的任何编辑的即时反馈，并预先配置了 Next.js。
    - 1.新建组件
    - 2.编写组件
    - 3.使用组件及时反馈
  - 服务器组件
    - 服务器组件的N大好处
      - 数据获取：服务器组件允许您将数据获取移动到服务器，使其更靠近您的数据源。这可以通过减少获取渲染所需数据所需的时间以及客户端需要发出的请求数来提高性能。
      - 安全性：服务器组件允许您将敏感数据和逻辑（例如令牌和 API 密钥）保留在服务器上，而不会将它们暴露给客户端。
      - 缓存：通过在服务器上渲染，可以缓存结果并在后续请求和用户之间重复使用。这可以通过减少对每个请求完成的渲染和数据获取量来提高性能并降低成本。
      - 性能：服务器组件为您提供额外的工具，以优化基准的性能。例如，如果您从完全由客户端组件组成的应用程序开始，则将 UI 的非交互式部分移动到服务器组件可以减少所需的客户端 JavaScript 数量。这对于 Internet 速度较慢或设备功能较弱的用户非常有用，因为浏览器需要下载、解析和执行的客户端 JavaScript 较少。
      - 初始页面加载和首次内容绘制（FCP）：在服务器上，我们可以生成 HTML，允许用户立即查看页面，而无需等待客户端下载、解析和执行呈现页面所需的 JavaScript。
      - 搜索引擎优化和社交网络可共享性：搜索引擎机器人可以使用呈现的 HTML 为您的页面编制索引，社交网络机器人可以为您的页面生成社交卡片预览。
      - 流式处理：服务器组件允许您将渲染工作拆分为多个块，并在它们准备就绪时将其流式传输到客户端。这样，用户就可以更早地看到页面的某些部分，而不必等待整个页面在服务器上呈现。
    - 服务器组件是如何渲染的
      - 静态渲染
      - 动态渲染
      - 通过流式处理，您可以从服务器逐步呈现 UI。工作被拆分为块，并在客户端准备就绪时流式传输到客户端。这允许用户在整个内容完成渲染之前立即看到页面的某些部分。
  - 客户端组件
    - 子主题 1
      - 交互性：客户端组件可以使用状态、效果和事件侦听器，这意味着它们可以向用户提供即时反馈并更新 UI。
    - 子主题 2
      - 浏览器 API：客户端组件可以访问浏览器 API，如地理位置或 localStorage。
- **nextjs 组件之间共享数据**
  - 1.在服务器上获取数据时，可能需要在不同组件之间共享数据。例如，您可能有一个依赖于相同数据的布局和页面。
  - 2.你可以使用 fetch 或 React 的缓存函数在需要它的组件中获取相同的数据，而不必担心对相同的数据发出重复请求，而不是使用 React Context（在服务器上不可用）或将数据作为 props 传递。这是因为 React 扩展了 fetch 来自动记住数据请求，并且当 fetch 不可用时可以使用 cache 函数。
    - 
      npm install server-only