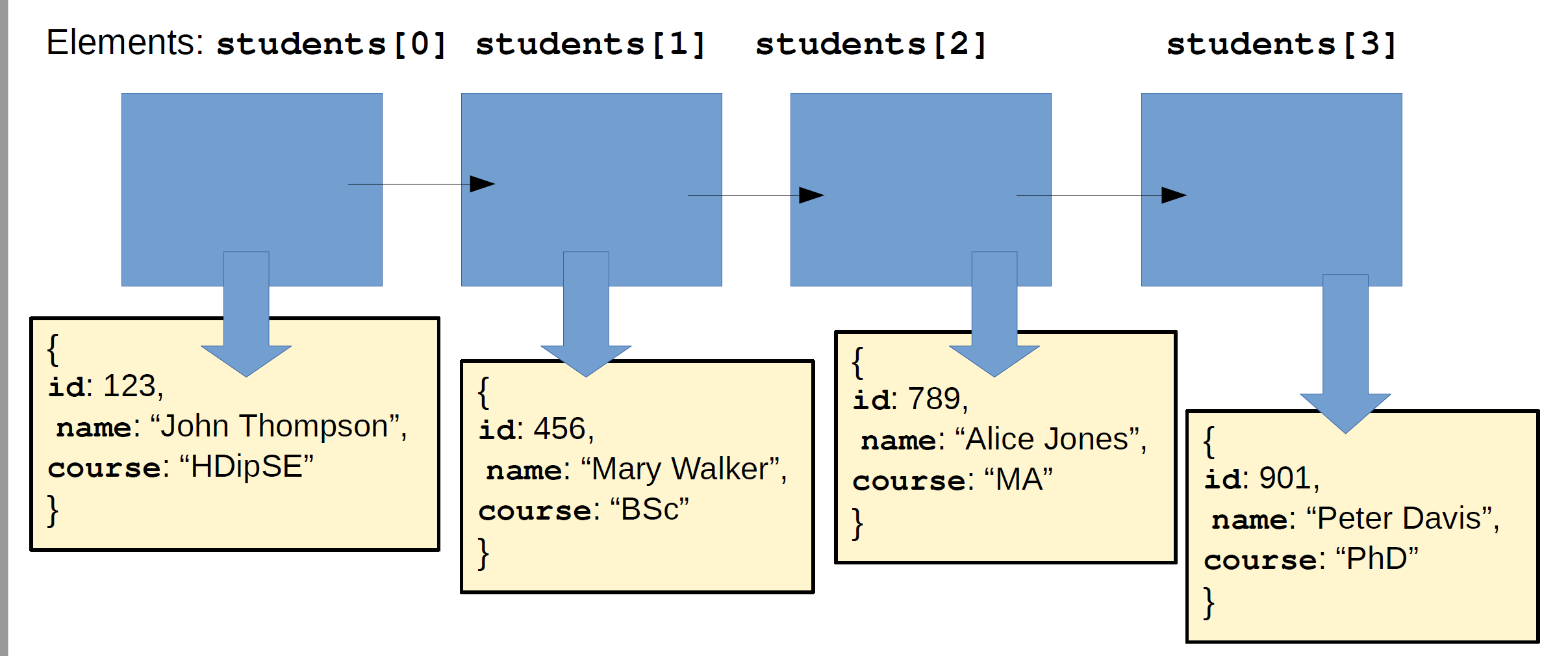
CS385 Learning Note

# Week 1

Function App() 是一个 component，它有返回语句，返回React需要渲染的JSX（HTML 和 JavaScript）

JSON文件是一个里面承载对象的数组。Property:Value

Each **Object** is an **Array** element.



# Week 2

<sup> 标签可定义上标文本 例如 ㎡

React框架

1. src 文件夹：App所有的源代码、数据、媒体文件储存的地方，可以在src中创建子文件夹。
2. Dependencies（依赖）：储存App需要用到的libraries和外来源代码
3. App.js 是默认的文件/组件名，export default app 默认导出。

function App() 是一个功能组件，没有参数输入，返回一个JSX(JavaScript 和 HTML的混合文件，这个会被渲染到HTML输出上)

export default用来从.js文件中导出一个类、函数或者原始数据类型。注意名字必须一样。

## **1. 默认导出（Default Export）**

// MyComponent.js

const MyComponent = () => {

// ...

};

export default MyComponent;

* 使用export default关键字
* 一个模块只能有一个默认导出
* 不需要使用大括号（{}）来包裹，在导入时可以使用任意名称来引用

## 2. 命名导出（Named Export）

// ComponentA.js

export const ComponentA = () => {

// ...

};

// ComponentB.js

export function ComponentB() {

// ...

}

//或者

//export {ComponentA,ComponentB};

* 使用export关键字，可以在函数关键字前，也可以在模块末尾
* 一个模块可以有多个命名导出
* 命名导出需要使用大括号（{}）来包裹，在导入时必须使用导出时指定的名称来引用导出的内容。

总结来说就是默认导出因为已经规定了default且一个模块只能导出一个，所以你在导入时指定模块路径后就算改名，react也知道你找的是哪个。

而命名导出可能有多个，且定义时没有定义默认值，所以你必须指定名字，react才能找到你想要的组件。

**导出的组件名、导入的组件名，和文件名必须一样。**

## 3 Map函数

Map ((c, index) -> (….))

C 用来对数组的每一个元素进行操作。不管数组元素多少，都可以从头到尾进行操作。

Index 用来从0到n，给数组的每一个元素进行编号，这是唯一标识符。这样在以后浏览器控制渲染的元素组件时，可以跟踪组件，避免重复。

使用map时，对象名称必须一样，包括大小写。

## 4 Filter函数

Array.filter(function(a))…

只有当function(a) 输出的结果为true时，array中的元素才会被筛选。

如果function返回false，那么这个array元素就省去。

可以把他用于下拉菜单，来筛选想要的元素。

## 5 Include函数

a.include(b) 。如果a 包含了b中的元素（连续），那么返回true。

toLowerCase() toUpperCase() 大小写转换

记得防止因为空输入，导致数据全部泄露。

If (length <=0 ) return false;

# Week 3

## 1 父子组件 Parent-Child Communications

Props. 用于父子组件传递信息

使用Props在父子组件传递，名字必须一样，包括大小写。

**父组件**

<ChildA xFromParent={parentX} />

**子组件**

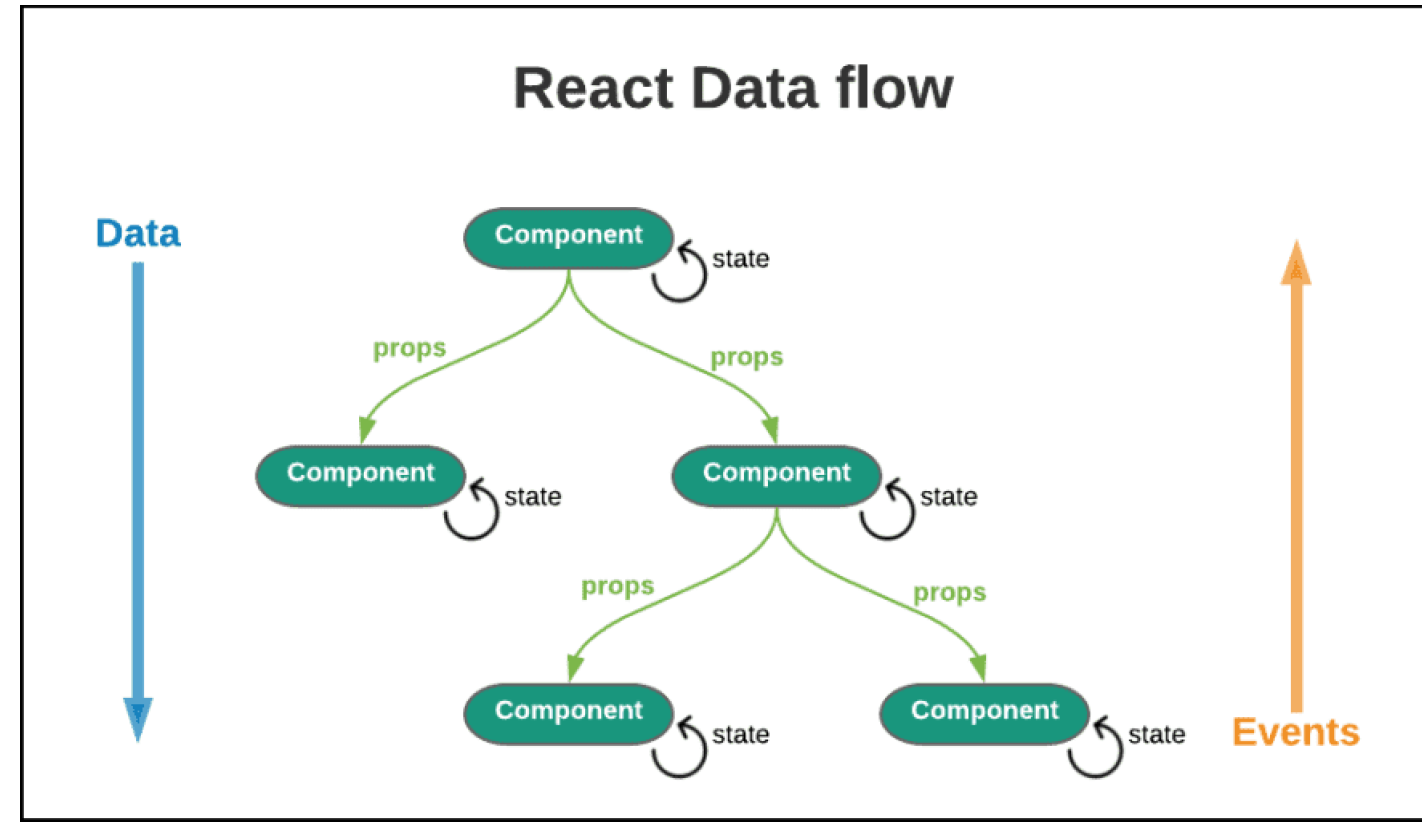
{props.xFromParent.map((i, index) => (

        <p key={index}>

          Name: {i.name}, Course: {i.course}

        </p>

      ))}



**父子组件通信的作用：**

1. 把代码分成小的单元，每个单元代表不同的用户界面元素。
2. 组件之间传递变量和处理方法的方法。

**Components 组件**

组件可以把代码分成小的包和单元

组件可以代表不同的用户界面

组件之间可以通过props传递State和handler functions

功能组件可以像App() 一样， 返回JSX( JavaScript+ HTML)

## 2 State和useState

Static 数据（静态）:代码中的数据不会变化和更新

State： React中Component的数据值

useState Hook。React唯一的 用来安全维持、更改组件里State值的方法

useState设定了state的初始值，以及改变state的方法。

const [ state, setState] = useState（0）

## 3 State VS Props

State 用来储存每个Component的变量值

Props 用于Component之间传递state的值

## 4 Reduce函数

同样用来处理JavaScript对象的数组。

Reduce函数把数组变成一个数值。它不会改变原数组的值。

从数组最左开始，每两个元素进行处理，并放入accumulator。

然后accumulator作为新的元素，再和下一个元素进行处理，直到最后。

basket.reduce(getBasketTotal, 0.0)

*getBasketTotal 是accumulator的处理方法，是对数组basket的回调函数*

*0.0是accumulator的初始值。*

function getBasketTotal(acc, obj) {

  return acc + obj.price;

}

Reduce函数可以和Filter函数一起使用。先用Filter筛选处需要处理的元素，然后再用reduce对这些选出来的元素进行accumulator操作。【考试必考】

# Week 4

e.target.value / event.target.value

**e:接收事件对象  
target:获取触发事件对象的目标  
e.target.value表示该DOM元素，然后在获取其相应的属性值**

1、定义：target属性返回的是触发该事件的元素。

2、用法：event.target

3、作用：就是获取当前会触发该事件的元素对象，如练习中的

$("input").select(function(e){

        alert(e.target.value)

    })

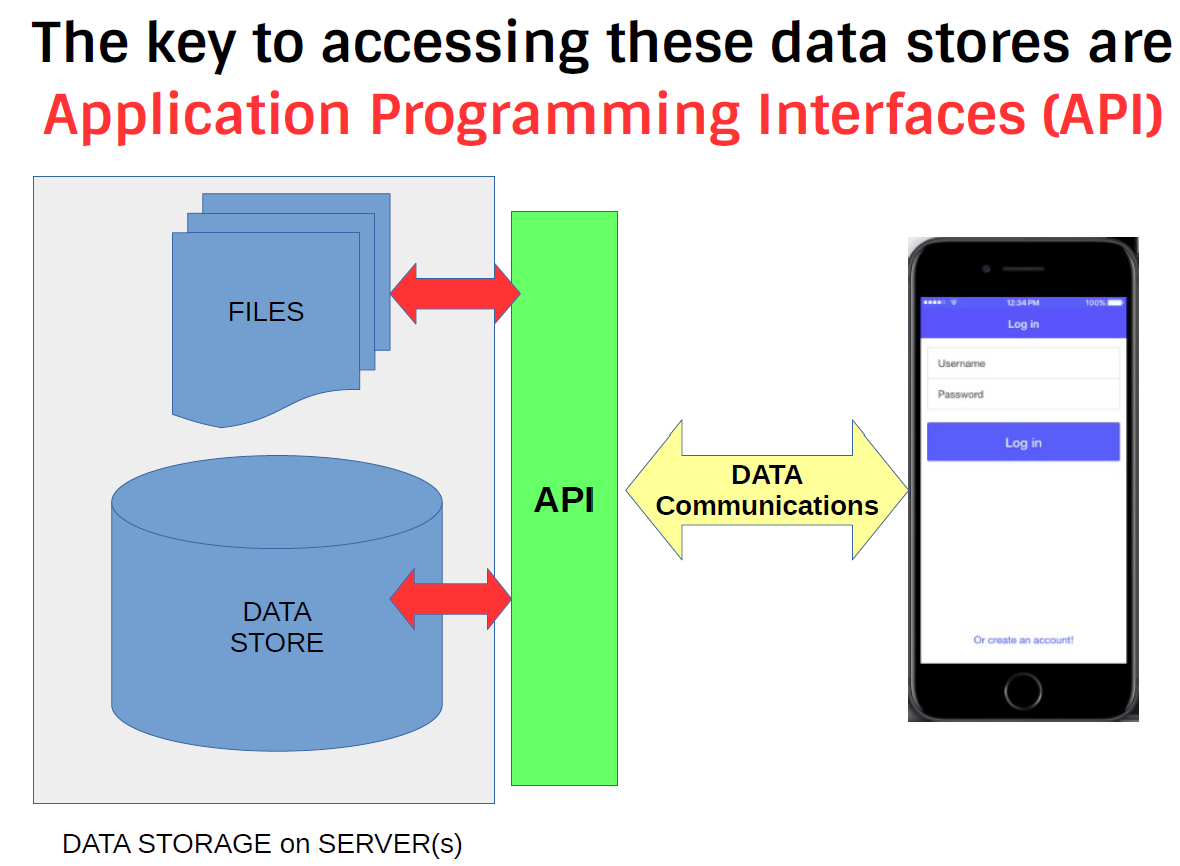
当前触发select事件的元素是input，故这里的e.target指的是input。而这里的e指代的是event对象，代表事件的状态。

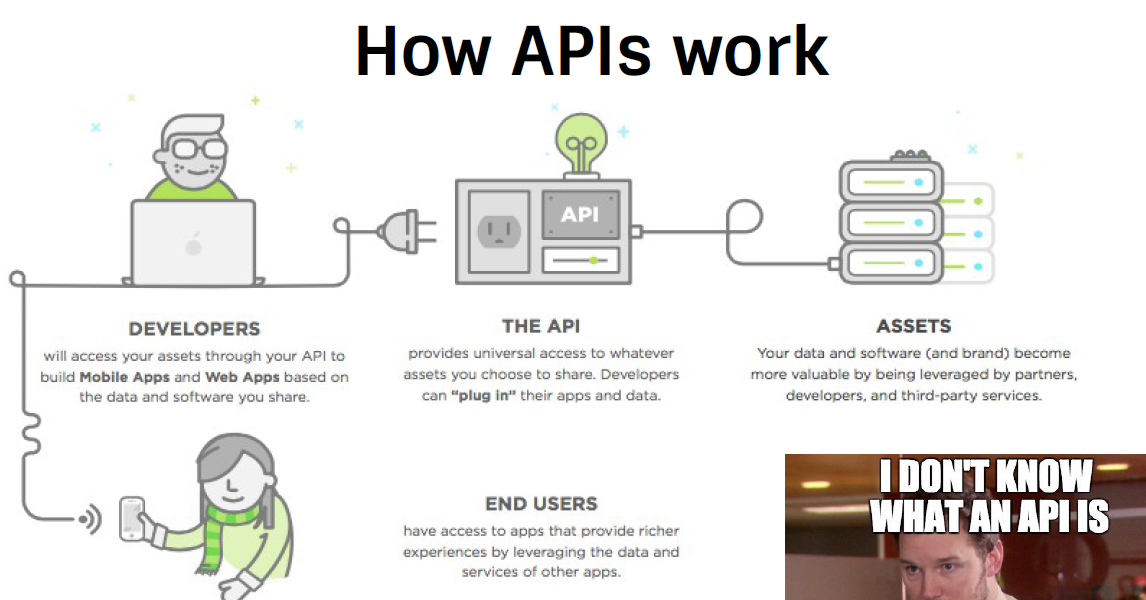
# Week 5

## **API**

**APIs (Application Programming Interfaces) 应用程序接口**

**API（应用程序接口）**是一种软件中介，它允许两个不相关的应用程序相互通信。它就像一座桥梁，从一个程序接收请求或消息，然后将其传递给另一个程序，翻译消息并根据 API 的程序设计执行协议。 中介 intermediary

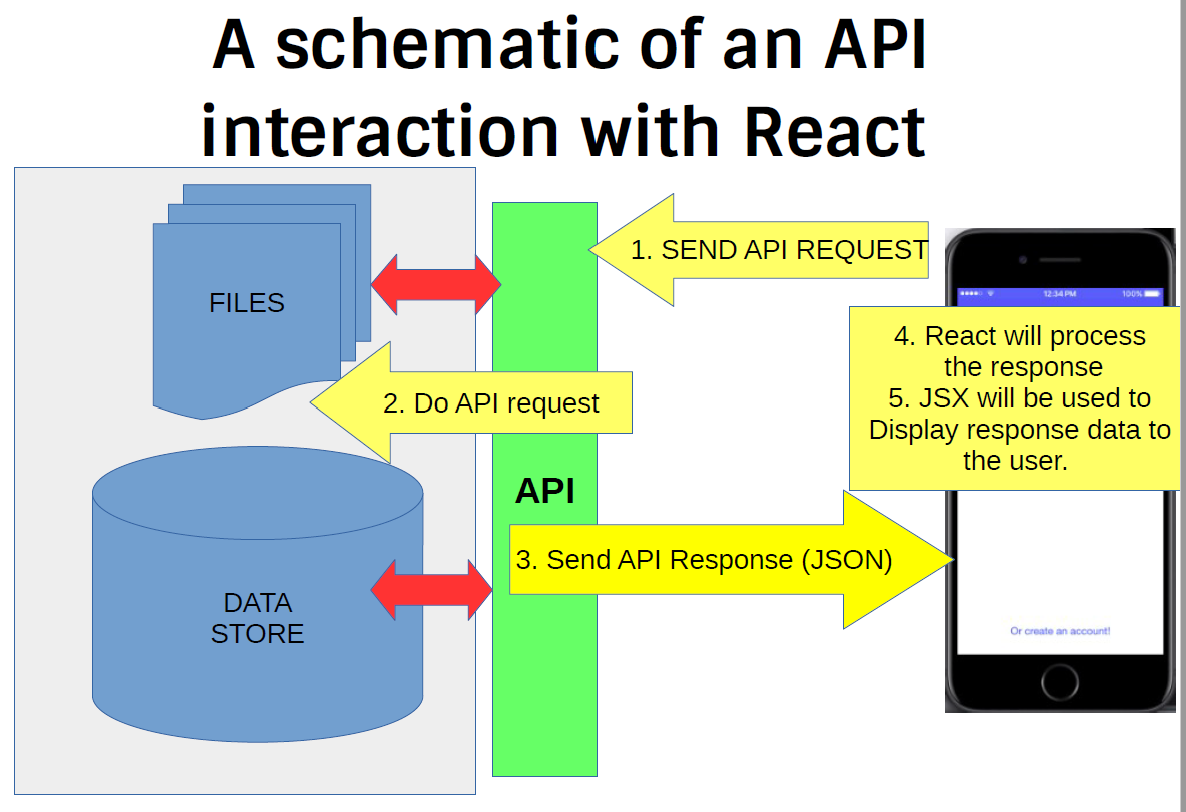
****

****

JSON(JavaScript Object Notation) JavaScript 对象表示法, 是用来机器之间互相交流的数据传输格式，如果需要人能读懂，需要进行格式转换。<https://jsonlint.com/>

Access Token 访问令牌。系统使用访问令牌来辨识拥有进程的用户，以及线程试图执行系统任务时是否具有所需的特权

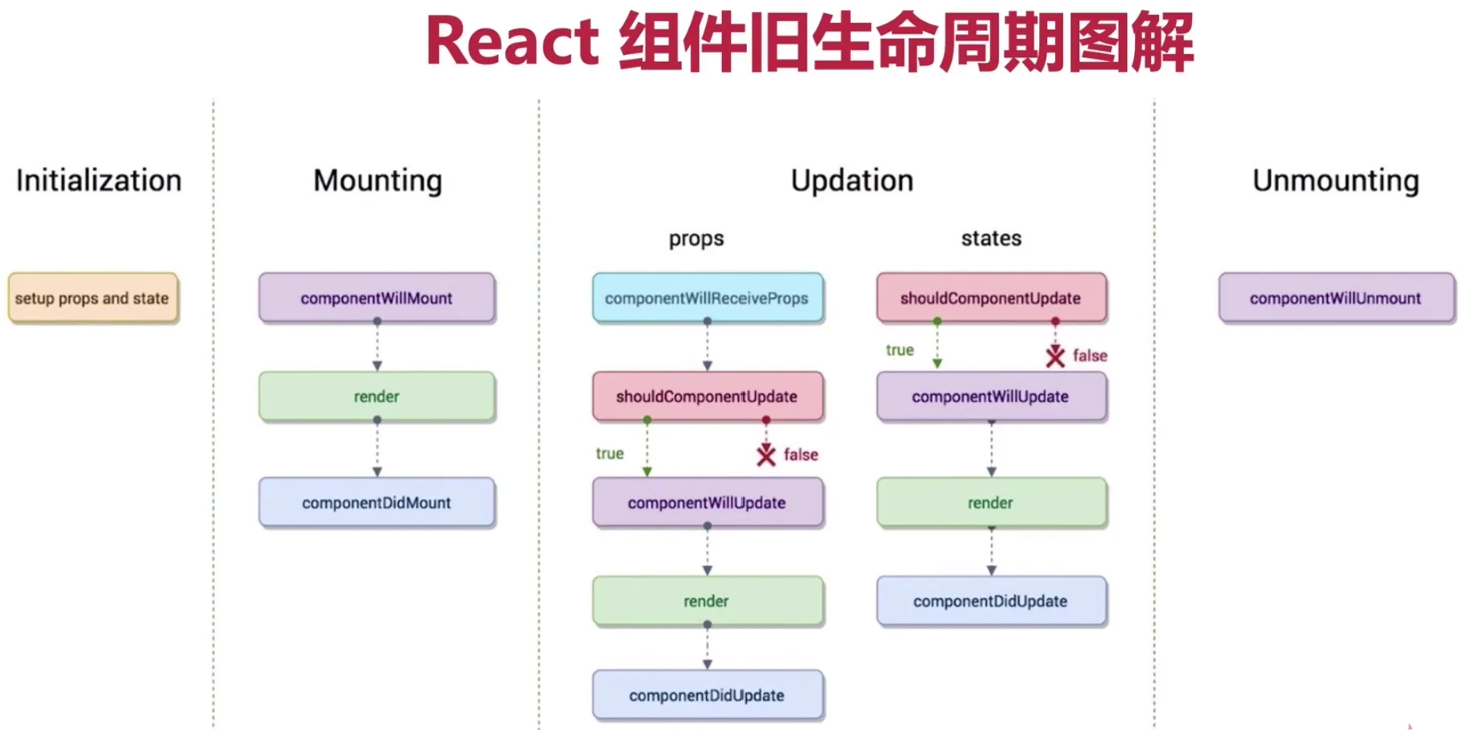
API Endpoint （API端点）-- 每个端点都是API 可以访问执行其功能所需的资源的位置。 API 使用“请求”和“响应”工作。 当API 从Web 应用程序或Web 服务器请求信息时，它会收到响应。 **API 发送请求的位置和资源所在的位置**称为端点。API端点通常是个网址。



**React处理API的示意图**

1. Client发送API请求
2. Server处理API请求
3. Server根据API请求，以JSON的格式发送API回复
4. Client接收到API回复后，通过React进行处理
5. 根据处理的结果，通过React渲染到输出界面上。

## **React 生命周期**



### 1. 组件初始化(Initialization)阶段

首先是Initialization,初始化state和props的数据，也就是类的构造方法 constructor()。

### 2. 组件的挂载(Mounting)阶段

componentWillMount

render

componentDidMount

组件挂载后（插入 DOM 树中）立即调用

### 3. 组件的更新(Updation)阶段

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render(这里只是重新调用)

componentDidUpdate

### 4. 组件的卸载(Unmounting)阶段

componentWillUnmount

useEffect( ( ) => {

componentDidMount();

componentDidUpdate();

return(( ) => {

componentWillUnmount();

})

},[ ])

## **UseEffect**

<https://www.ruanyifeng.com/blog/2020/09/react-hooks-useeffect-tutorial.html>

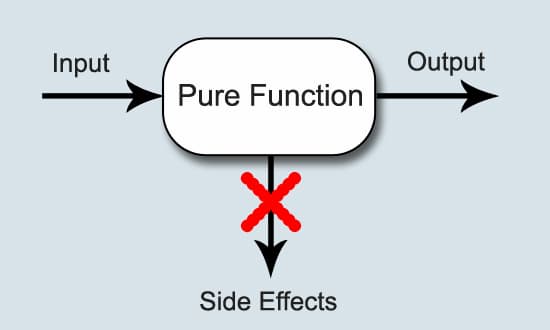
UseEffect是一个 React Hook，可以让你**同步**一个 带有外部系统的组件。

Effect让你在渲染后运行一些代码，以便你可以将您的组件与某些系统同步 在 React 之外。

UseEffect是属于函数组件。函数一般来说，只应该做一件事，就是**返回一个值**。React 的函数组件只应该做一件事情：**根据输入的参数，返回组件的 HTML 代码**，而没有其他的功能。这种只进行单纯的数据计算（换算）的函数，在函数式编程里面称为 **"纯函数"（pure function）。**

### 副效应

函数式编程将那些**跟数据计算无关的操作**，都称为 **"副效应" （side effect**） 。如果函数内部直接包含产生副效应的操作，就不再是纯函数了，我们称之为不纯的函数。



一句话，**钩子（hook**）就是 React 函数组件的副效应解决方案，用来为函数组件引入副效应。 函数组件的主体只应该用来返回组件的 HTML 代码，所有的其他操作（副效应）都必须通过钩子引入。

由于副效应非常多，所以钩子有许多种。React 为许多常见的操作（副效应），都提供了专用的钩子。

useState()：保存状态

useContext()：保存上下文

useRef()：保存引用

useEffect(function(), [dependency])是通用的**副效应钩子。**

**useEffect()的作用就是指定一个副效应函数，组件每渲染一次，该函数就自动执行一次。组件首次在网页 DOM 加载后，副效应函数也会执行。**

有时候，我们不希望useEffect()每次渲染都执行，这时可以使用它的第二个参数，使用一个数组指定副效应函数的**依赖项**，**只有依赖项发生变化，才会重新渲染。**

function Welcome(props) {

useEffect(() => {

document.title = `Hello, ${props.name}`;

}, **[props.name]);**

return <h1>Hello, {props.name}</h1>;

}

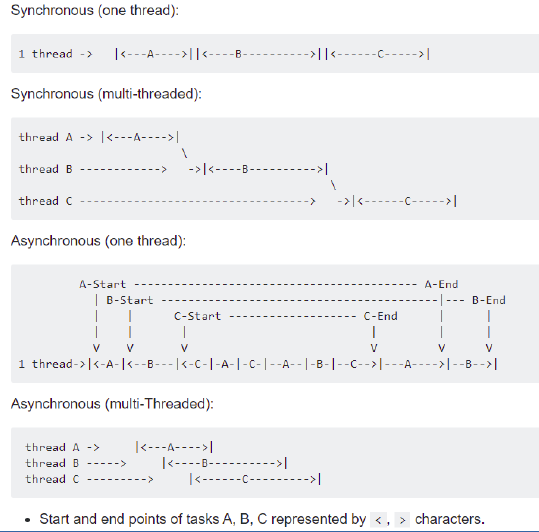
如果第二个参数是一个空数组，就表明副效应参数没有任何依赖项。因此，**副效应函数这时只会在组件加载进入 DOM 后执行一次，后面组件重新渲染，就不会再次执行。**

UseEffect的常见用途有下面几种。

1. 获取数据（data fetching）
2. 事件监听或订阅（setting up a subscription）
3. 改变 DOM（changing the DOM）
4. 输出日志（logging）

使用useEffect()时，有一点需要注意。如果有多个副效应，应该调用多个useEffect()，而不应该合并写在一起。

## **Async**



**async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成。**

await 只能出现在 async 函数中

async 函数返回的是一个 Promise 对象

await 等待的是一个表达式，这个表达式的计算结果是 Promise 对象。

## Promise

Promise 对象代表了未来将要发生的事件，用来传递**异步操作**的消息。

对象的状态不受外界影响。Promise 对象代表一个异步操作，有三种状态：

pending: 初始状态，不是成功或失败状态。

fulfilled: 意味着操作成功完成。

rejected: 意味着操作失败。

只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是 Promise 这个名字的由来，它的英语意思就是「承诺」，表示其他手段无法改变。

Promise 对象的状态改变，只有两种可能：从 Pending 变为 Resolved 和从 Pending 变为 Rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果。就算改变已经发生了，你再对 Promise 对象添加回调函数，也会立即得到这个结果。

var promise = new Promise(function(resolve, reject) {

// 异步处理

// 处理结束后、调用resolve 或 reject

});

Promise 构造函数包含一个参数和一个带有 resolve（解析）和 reject（拒绝）两个参数的回调。在回调中执行一些操作（例如异步），如果一切都正常，则调用 resolve，否则调用 reject。

# Week 6

&nbsp;

不换行空格，“Non-Breaking Space” 的缩写。

NBSP是常用的编程和网页创建的行不打破空间上的程序或文档的空间。

不换行空格是空格字符，用途是禁止自动换行。HTML页面显示时会自动合并多个连续的空白字符，但该字符是禁止合并的，因此该字符也称作“硬空格”

展开操作符(spread operator)

**slice()** 方法返回一个新的数组对象，这一对象是一个由 start 和 end 决定的原数组的[浅拷贝](https://developer.mozilla.org/zh-CN/docs/Glossary/Shallow_copy)（包括 start，不包括 end），其中 start 和 end 代表了数组元素的索引。原始数组不会被改变。

# Week 9

## **JSON.stringify()**

JSON.stringify() 把JSON对象转化为字符串

JSON对象是一个序列化的数组，用方括号[]把每个对象括起来，所有的对象从0到length-1分布，其他的元素无视。

## **useEffect(function(), [dependency])**

useEffect(function(), [dependency])

dependency是useEffect的依赖，里面储存的就是组件的State

1. 如果dependency里储存的是组件的State，那么每次State的值变化时，UseEffect就会运行依次。
2. 如果dependency里储存的是空数组[]，那么UseEffect只会在第一次加载的时候，运行一次。
3. 如果没有dependency，那么UseEffect() 在每次组件渲染的时候，都会运行。

UseEffect() 的Dependency在处理API URL时非常有用。

## **Spread Operator**

return { ...item, value: item.value + newValue };

这里处理对象item，保持item其他属性不变，修改value属性的值！

## **如何总结商品重复的次数**

1. 如果只有一个对象，创建qty属性，让qty=1。
2. 如果至少有2个对象以上了，寻找这个对象的位置，让qty+1

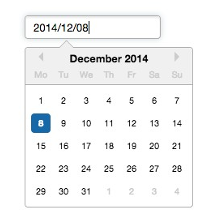
## **JavaScript.prototype.findIndex(fuc)**

如果fun函数返回true，那么函数返回第一个满足条件的元素下标值

如果fun函数全部都返回false，那么函数返回-1.

## **React-datepicker**

一个输入框，输入日期。



import DatePicker from “react-datepicker”;

安装依赖：

**npm install react-datepicker –save**

## **Firebase**

Firebase 是一种非关系型数据库 （NoSQL）。我们可以直接插入JSON到这种非关系型数据库。

文本

描述已自动生成

Firebase有real-time数据库和cloud-based数据库，他们能实时同步更新Component和Firebase之间的数据。**实时数据库**对于Firebase很重要

Oauth ---- Open Authentication 开放式授权

# Week 10

    <Router>

      <div>

        <h1>Welcome to CS385 React Router!</h1>

        <h2>Variable has the value {variable}</h2>

        <button onClick={changeVar}>Increment Variable</button>

        <p>Route Layer 1 - Home/A/B/C</p>

        <ul>

          <li>

            <Link to="/">Home Page</Link>

          </li>

          <li>

            <Link to="/a">Page A</Link>

          </li>

          <li>

            <Link to="/b">Page B</Link>

          </li>

          <li>

            <Link to="/c">Page C</Link>

          </li>

        </ul>

        <Routes>

          <Route path="/" element={<h2>This is Home Page</h2>} />

          {/\* Catch all route that begin with '/a' \*/}

          <Route path="/a/\*" element={<PageA theVar={variable} />} />

          <Route path="/b" element={<PageB theVar={variable} />} />

          <Route path="/c" element={<PageC theVar={variable} />} />

        </Routes>

      </div>

    </Router>

## react-router-dom

`react-router-dom` 是一个用于在 React 应用程序中实现路由（页面导航和管理）的库。它提供了一组组件和钩子（hooks），能够在单页应用程序中无需重新加载页面就可以导航。这些组件和钩子帮助你定义应用程序的导航结构、捕获 URL 变化，并根据这些变化呈现相应的界面。

下面是几个主要组件的作用：

## BrowserRouter

`BrowserRouter` 是一个路由器，它保持 UI 和 URL 的同步。它创建一个路由环境，允许在BrowerRouter中任何位置使用路由功能。通常，它被放置在应用程序的最顶层。

使用BrowserRouter as Router， 那么可以直接用<Router>标签代替所有<BrowserRouter>标签。

## Routes

- `Routes` 组件用于**定义应用中的路由规则**。它取代了早期版本的 `react-router-dom` 中的 `Switch` 组件，用于决定**Route中渲染的组件和URL路径的关系**。

所有的 **`Route`** 都是` **Routes`的子组件**。

## Route

- `Route` 组件是用来**定义 UI 响应路径的变化的**。

- 每个 **`Route` 对应一个路径**（通过 **`path` 属性**定义）以及当该**路径匹配时应该渲染的组件**（通过 **`element`** 属性定义）。

## Link

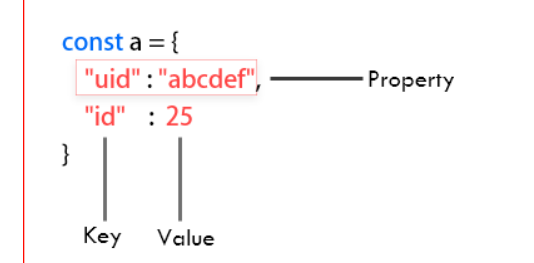
- `Link` 组件用于在应用中创建导航链接。

- 它类似于 HTML 中的 `<a>` 标签，但它使用 **`to` 属性**来**指定导航的目标地址**，并且它不会导致页面重新加载。

、

## Object.is()

比较两个对象里的**内存引用**和储存的值，是否一样。



## Object.getOwnPropertyNames()

处理对象，返回一个数组，这个数组里包含了对象中所有属性地名字。

## Shallow Comparison 浅比较

## Fuzzy matching 模糊匹配

## Object Deconstruction 对象解构

  const newArrayWithoutAge = originalArray.map(({ age, ...rest }) => {

    return rest;

  });

返回对象里除了age属性以外，其他属性的名字和它们的值。

# 拖动条 Scrollbar

import React, {useState} from 'react';

const Scrollbar = () => {

   const [size, setSize] = useState(250);

  const handleSizeChange = (value) => {

    setSize(value);

  };

    const handScroll = (e) => {

        const value = e.target.value;

        handleSizeChange (value);

    };

    return (

        <div className = "Scrollbar">

          <h4>{size}</h4>

          <input

                type="range"

                min = "50"

                max = "500"

                defaultValue = "400"

                onChange = {handScroll}

          />

        </div>

    )

};

export default Scrollbar;

# UseRef

**useRef主要的功能就是帮助我们获取到DOM元素或者组件实例，它还可以保存在组件生命周期内不会变化的值。**

<https://juejin.cn/post/6894963612687958024>