1-Introduction

VScode扩展应用simple react snippets

设置emmet

Emmet: Include Languages

在默认不受支持的语言中启用 Emmet 缩写。添加 javascriptreact

2-Creating a React Application

```
npx create-react-app zz-blog
cd zz-blog
code .
```

Opening VSCode from the Terminal in macOS

- https://knasmueller.net/opening-vscode-from-the-terminal-in-macos
- 在终端执行：

```
ln -s "/Applications/Visual Studio Code.app/Contents/Resources/app/bin/code" /usr/local/bin/
```

- **Public是向浏览器公开的文件**

里面有一个index.html文件，所有的react代码注入这个div中

```
<body>
    <div id="root"></div>
</body>
```

- **index.js文件**

which is taking this app components and its rendering it give to the dom inside root elements.

- **npm指令**

```
npm  install
npm  run  start
```

3-Components & Templates

Our job: creat the component which render to the dom and show in the webpage by React.

```
function  App(){
    return(
        //jsx
        //babel把这些jsx最后转换为html呈现给dom
    )
}
```

jsx和html的区别：

jsx在js文件里写，所以不能用class，而要用className 但在浏览器里被渲染为class

react16以前的版本

需要在App.js里引入

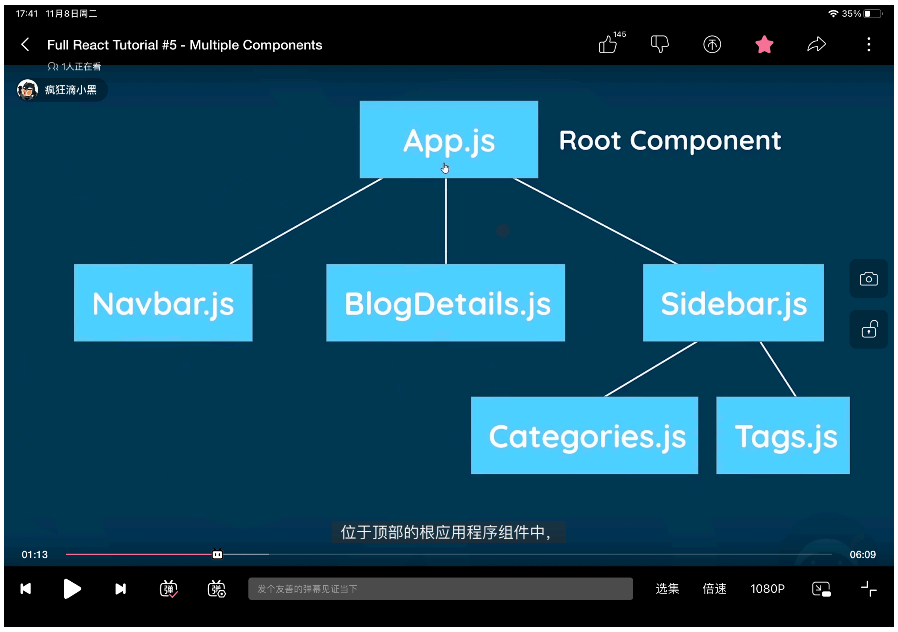
```
import  React  from  'react'
```

4-Dynamic Values in Templates

在函数内，返回jsx前，可以写任意的 js 代码

jsx的{ }里不能直接输出 对象、布尔值，其它都可以，并输出为字符串显示在浏览器中。

5-Multiple Components



- **A component is just a function which returns a jsx template, and that function is exported at the bottom of the file.**
- **sfc+tab 安装插件simple react snippets的作用，创建一个组件函数**

6-Adding styles

- 在jsx里 不用“-”号，用驼峰命名法
- 当我们需要动态变化的值时，放在{ }里

7-Click Events

- 不想一打开就立即调用 一个传参函数，想在点击按钮时再调用，那么就把这个传参函数包装在一个匿名函数（箭头函数）里
- 事件对象events object/events parameter
- 当我们调用函数时，它自动获得第一个参数：事件对象e

8-Using State(useState hook)钩子一

```
const Home = () => {
  // 解构出两个值
  const [name, setName] = useState("mario");

  const handleClick = () => {
    setName("luigi");
  };

  return (
    <div className="home">
      <h2>Homepage</h2>
      <p>{name}</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};
```

- The state of components: we just mean the data being used in that component and that point in time now. That's gonna could be values, booleans, strings, object or any other data that our component uses.
- React doesn't watch it for changes or react to this changes, so when the value change, it doesn't trigger out the rerender the templete with that new value insides, nothing happens.
- Hook, that hook is called useState. So hooking react is a special type of function, that does certain job
    - You can tell hook by its name, because it start with word USE.
    - The useState hook give us a way to make our reactive value, and also provide us with the way to change that value whenever we want.
- 钩子来监管值的变化，变让更新后的值渲染到页面
- When we use this function to change the value, that triggers react to rerender the component. When it rerender, we have that new value, because it's being updated.
- 用钩子创建那些将来某一时间点会变化的数据，因为发生变化时我们需要做出反应来更新dom。

9-Intro to React Dev Tools

谷歌扩展程序 React develop tools

10-Outputting Lists

用到钩子和map函数循环list：

```
{blogs.map((blog) => (
        <div className="blog-preview" key={blog.id}>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
        </div>
      ))}
```

11-Props

- 一个组件不能访问到另一个组件的数据，可以用props传数据
- props: to pass data from a parent component into a child component
    - parent component: Home 通过props传数据

```
return (
    <div className="home">
    //blogs={blogs} title="All Blogs!"——这里就是props
      <BlogList blogs={blogs} title="All Blogs!" />
    </div>
  );
```

    - child component: BlogList 通过props接收数据

```
const BlogList = (props) => {
  const blogs = props.blogs;
  const title = props.title;
  }
  //   更简单的写法，直接解构
  // const BlogList = ({ blogs, title })
```

12-Reusing Components

```
<BlogList blogs={blogs} title="All Blogs!" />
      <BlogList
        blogs={blogs.filter((blog) => blog.author === "mario")}
        title="Mario's Blogs"
      />
```

13-Function as Props

- 删除blog：用钩子里的方法来改变数据，在parent组件里写删除blog的函数，再通过props传到child组件里

```
  //通过对比点击拿到的id和blog本身的id进行删除
  const handleDelete = (id) => {
    const newBlogs = blogs.filter((blog) => blog.id !== id);
    setBlogs(newBlogs);
  };
```

14-useEffect Hook(the basics)钩子二

- Another Hook
- This hook run the function every render of the component
- The component renders initial when it first loads and it renders all of this to the dom, but it also happens when the state changes, it renders the dom. So we can update that states in the browser
- **useEffect Hook** is a way to run code on every render. **useEffect** 这个函数在每次渲染时都会触发.

```
useEffect(() => {
    console.log("use effect ran");
  });
```

15-useEffect Dependencies

- You don't always want to run the function after every single renders, may be some certain renders, to do that we can use somethings call Dependencies Array. 作为useEffect的第二个参数传入：

```
  useEffect(() => {
    console.log("use effect ran");
  }, []);
```

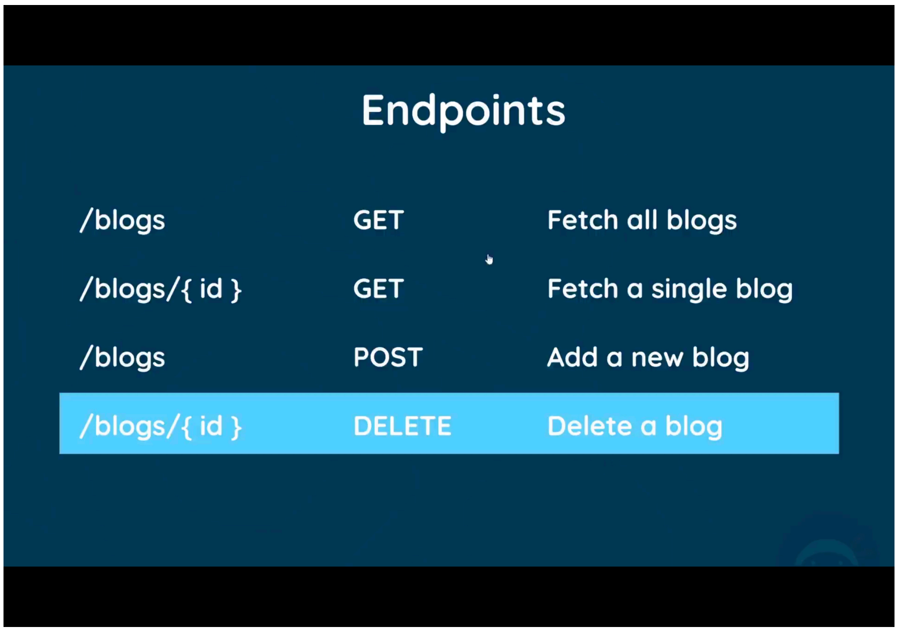
- 依赖数组为[]空时，只在初始化时渲染一次
- How we use dependencies: these dependencies array right here as second argument to useEffect to control when this function runs

```
const [name, setName] = useState("mario");
useEffect(() => {
    console.log("use effect ran");
    console.log(name);
}, [name]);

return (
    <div className="home">
      <button  onClick={  () => setName("luigi")  }>  change name  </button>
       <p>{name}</p>
     </div>
   );
```

16-Using JSON Server

- We gonna use useEffect to fetch some data, use a package called json server: which is gonna allow us to build a fake rest API, just as a json file, then we can use to test
  - creat a json file as our database

```
    npx json-server --watch data/db.json --port 8000
```



Endpoints

/blogs	GET	Fetch all blogs
/blogs/{ id }	GET	Fetch a single blog
/blogs	POST	Add a new blog
/blogs/{ id }	DELETE	Delete a blog

17-Fetching Data with useEffect

```
    /**
```

```
     * 第一步：返回一个响应对象, use json method on that
     *    fetch返回一个promise, res是一个response object, 并不是data, 为了获取数据, 用.then方法
     *    res.json() pass the json into a javaScript object, return 回来的也是一个promise, 因为
     * 第二步：再用一个.then方法获取数据
     */
   useEffect(() => {
   fetch("http://localhost:8000/blogs")
     .then((res) => {
       return res.json();
      })
      .then((data) => {
       console.log(data);
      });
  }, []);
```

18-Conditional Loading Message

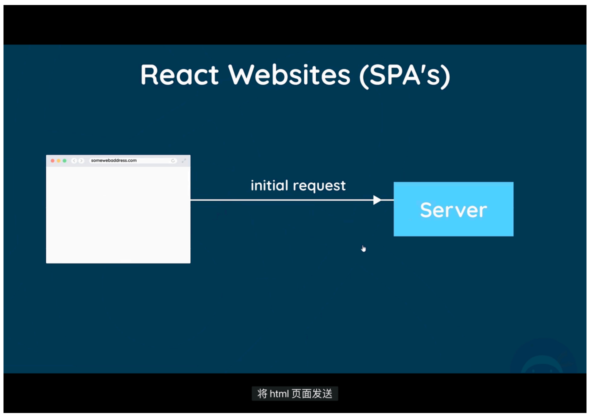
19-Handing Fetch Errors

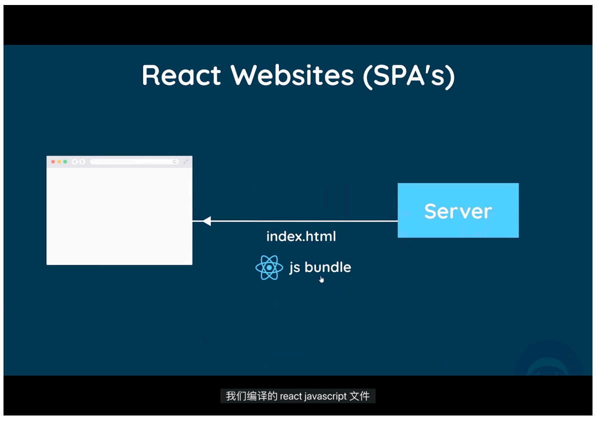
```
const Home = () => {
  const [blogs, setBlogs] = useState(null);
  const [isPending, setIsPending] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    setTimeout(() => {
      fetch("http://localhost:8000/blogs")
        .then((res) => {
          if (!res.ok) {
            throw Error("could not fetch the data for that resource");
          }
          return res.json();
        })
        .then((data) => {
          setBlogs(data);
          setIsPending(false);
          setError(null);
        })
        .catch((err) => {
          setIsPending(false);
          setError(err.message);
        });
    }, 1000);
  }, []);

   return (
     <div className="home">
       {error && <div>{error}</div>}
       {isPending && <div>Loading...</div>}
      {blogs && <BlogList blogs={blogs} title="All Blogs!" />}
    </div>
  );
};
```

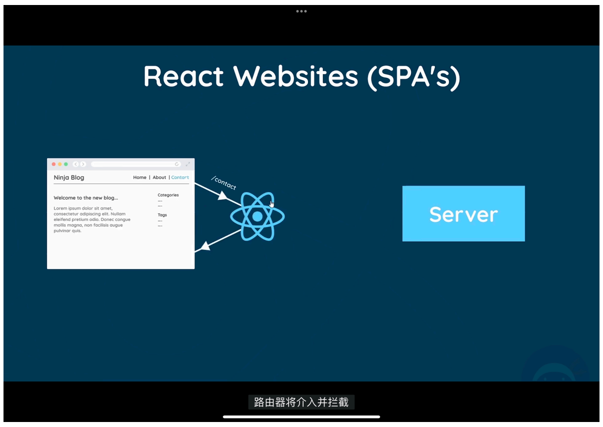
20-Making a Custom Hook

- 自定义钩子，把刚才的三个钩子放在一个自定义的钩子useFetch.js里，其它组件调用即可

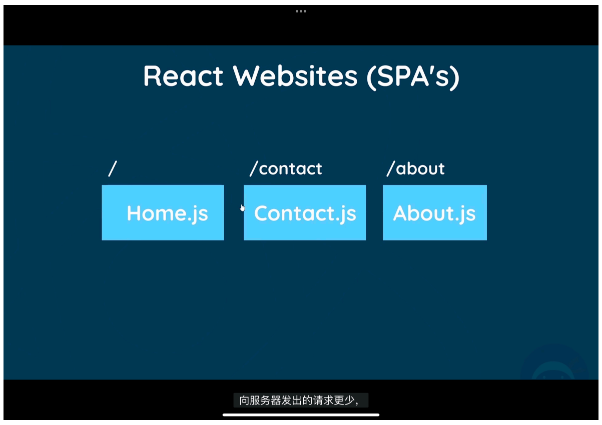
21-The React Router







- React and React router can take full control of the application
- 服务器返回一次请求后，就交给浏览器处理
- Initially, the HTML page that we get back is eventually empty, and then react injects the content dynamically using the component we creates.



- We assignment a top level component for each router or page, and that component is dynamically injected into the browser when we visit that route.

```
//App.js

import { BrowserRouter as Router, Route, Switch } from "react-router-dom";

function App() {
  return (
    <Router>
      <div className="App">
        <Navbar />
         <div className="content">
           <Switch>
             <Route path="/">
               <Home />
             </Route>
```

```
          </Switch>
        </div>
      </div>
    </Router>
  );
}
```

22-Exact Match Routes

```
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";

    <Switch>
    <Route exact path="/">
      <Home />
    </Route>
    <Route path="/create">
      <Create />
    </Route>
  </Switch>
```

- Switch组件让浏览器内只显示一个页面，从上到下找到第一个匹配的路由渲染显示

23-Router Links

```
<Link to="/">Home</Link>
<Link to="/create">New Blog</Link>
```

24-useEffect Cleanup

```
  useEffect(() => {
    const abortCont = new AbortController();

    setTimeout(() => {
      fetch(url, { signal: abortCont.signal })
        .then((res) => {
          if (!res.ok) {
            throw Error("could not fetch the data for that resource");
          }
          return res.json();
        })
        .then((data) => {
          setData(data);
          setIsPending(false);
          setError(null);
        })
        .catch((err) => {
          if (err.name === "AbortError") {
            console.log("fetch aborted");
          } else {
            setIsPending(false);
```

```
              setError(err.message);
            }
          });
      }, 1000);

      return () => abortCont.abort();
    }, [url]);
```

- AbortController, associated with a specific fectch request, we can use that AbortController to stop the fetch

25-Router Parameters

```
//BlogDetails.js
import { useParams } from "react-router-dom";

const BlogDeatails = () => {
  // 从 url 中获取参数
  const { id } = useParams();

  return (
    <div className="blog-details">
       <h2>Blog details - {id}</h2>
     </div>
   );
 };

 //BlogList.js
 <Link to={`/blogs/${blog.id}`}>
     <h2>{blog.title}</h2>
     <p>Written by {blog.author}</p>
 </Link>
```

26-Reusing Custom Hooks

- 复用了useFetch钩子

```
const BlogDeatails = () => {
  // 从 url 中获取参数
  const { id } = useParams();
  const {
    data: blog,
    error,
    isPending,
  } = useFetch("http://localhost:8000/blogs/" + id);

   return (
     <div className="blog-details">
       {isPending && <div>Loading...</div>}
       {error && <div>{error}</div>}
```

```
      {blog && (
        <article>
          <h2>{blog.title}</h2>
          <p>Written by {blog.author}</p>
          <div>{blog.body}</div>
        </article>
      )}
    </div>
  );
};
```

27-Controlled Inputs (forms)

- 一方面，存储用户输入值 + 存储输入时的状态
- 另一方面，状态发生变化时 + 更新值

```
const [title, setTitle] = useState("");

return(
    <form>
        <label>Blog title:</label>
        <input
          type="text"
          required
          value={title}
           onChange={(e) => setTitle(e.target.value)}
         />
    </form>
)
```

28-Submit Events

29-Making a POST Request

- 给json服务器提交一个post请求，给 data file 中添加 data

```
  const handleSubmit = (e) => {
    // 防止提交后页面自动刷新
    e.preventDefault();
    const blog = { title, body, author };

    setIsPending(true);

    fetch("http://localhost:8000/blogs", {
      method: "POST",
       headers: { "Content-Type": "application/json" },
       body: JSON.stringify(blog),
     }).then(() => {
       setIsPending(false);
     });
  };
```

30-Programmatic Redirects

- **useHistory 钩子三**

```
    // history.go(-1);
    // 提交表单后重定向到主页
    history.push("/");
```

31-Deleting Blogs

```
const handleClick = () => {
  fetch("http://localhost:8000/blogs/" + blog.id, {
    method: "DELETE",
  }).then(() => {
    history.push("/");
  });
```

32-404 Pages & Next Steps