# React核心概念


**React核心概念**

**课堂目标**

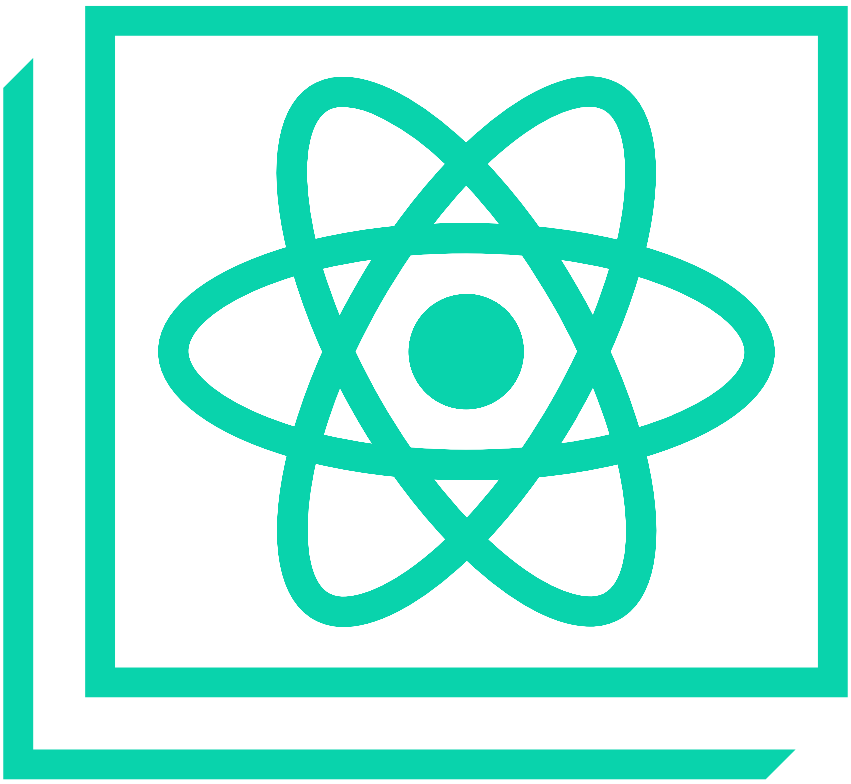
- React
- 资源
- 环境配置与快速上手
  - CDN链接
    - 开发环境，不适合生产环境
    - 生产环境
  - create-react-app
- 起步
  - hello world
  - 初识JSX
    - JSX的使用
    - Jsx中通过{}使用表达式
    - 动态渲染UI
    - 条件渲染
      - if语句
      - 逻辑与&&
      - 三元表达式 ||三目运算符
      - 循环列表 && key
      - React Dom元素属性的使用
      - 样式解决方案
  - 组件
    - 函数组件
    - 组件Props
    - 组件使用
    - 组件注意事项
    - class组件
    - 组件状态管理
      - 类组件中的状态管理
  - 事件处理
  - 组件通信

Props属性传递 遵守单项数据流
context
redux
生命周期
`Vue` 与 `React` 两个框架的粗略区别对比
相似之处
不同点
**React16 新特性**
1 引言
2 概述
React DevTools
下节课
回顾

# 课堂目标

1. create-react-app使用
2. 掌握组件使用
3. 掌握JSX语法
4. 掌握setState
5. 理解组件生命周期
6. 掌握组件通信各种方式

# React



**A JavaScript library for building user interfaces**

# 资源

1. react
2. create-react-app
3. 16.9新特性

# 环境配置与快速上手

## CDN链接

可以通过 CDN 获得 React 和 ReactDOM 的 UMD 版本。

### 开发环境，不适合生产环境

```
<script crossorigin
src="https://unpkg.com/react@16/umd/react.development.js">
</script>
<script crossorigin src="https://unpkg.com/react-
dom@16/umd/react-dom.development.js"></script>
```

### 生产环境

```
<script crossorigin
src="https://unpkg.com/react@16/umd/react.production.min.js
"></script>
<script crossorigin src="https://unpkg.com/react-
dom@16/umd/react-dom.production.min.js"></script>
```

如果需要加载指定版本的 react 和 react-dom，可以把 16 替换成所需加载的版本号。

如果你通过 CDN 的方式引入 React，我们建议你设置 `crossorigin` 属性：

## create-react-app

Create React App 是一个官方支持的创建 React 单页应用程序的方法。它提供了一个零配置的现代构建设置。

```
1. 安装官方脚手架：npm install -g create-react-app
2. 创建项目：create-react-app my-app
3. 启动项目：npm start 或 yarn start
4. 构建生产：npm run build 或 yarn build
5. webpack扩展：npm run eject //这是单向操作。一旦你 eject ，就不能恢复!
```

输出项目结构

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public //公共文件，里边有公用模板和图标等一些东西
|   ├── favicon.ico
|   ├── index.html
|   └── manifest.json
└── src //工作区域
    ├── App.css
    ├── App.js  //
    ├── App.test.js
    ├── index.css
    ├── index.js //入口文件
    ├── logo.svg
    └── serviceWorker.js //pwa 配合线上发布，实现离线使用的目的
```

# 起步

## hello world

删除src下面所有代码，新建index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

// 这里怎么没有出现React字眼?
// JSX => React.createElement(...)
ReactDOM.render(<h1>hello world</h1>,
document.querySelector('#root'));
```

> React负责逻辑控制，数据 -> VDOM
>
> ReactDom渲染实际DOM，VDOM -> DOM，如果换到移动端，就用别的库来渲染
>
> React使用JSX来描述UI

## 初识JSX

JSX是一种JavaScript的语法扩展，其格式比较像模版语言，但事实上完全是在JavaScript内部实现的。JSX 仅仅只是 `React.createElement(component, props, ...children)` 函数的语法糖。

JSX可以很好地描述UI，能够有效提高开发效率，

React官网体验JSX

Babel在线编译器体验JSX

> JSX实质就是React.createElement的调用，最终的结果是React“元素”（JavaScript对象）,React"元素"类型可以是原生Dom，字母小写，首字母大写的视为自定义组件。

```
const jsx = <h2>hello React</h2>;
ReactDOM.render(jsx, document.getElementById('root'));
```

## JSX的使用

## Jsx中通过{}使用表达式

```
const title = "hello React";
const jsx = <h2>{title}</h2>;
```

函数也是合法表达式

```
const user = { firstName: "tom", lastName: "jerry" };
function formatName(user) {
  return user.firstName + " " + user.lastName;
}
const jsx = <h2>{formatName(user)}</h2>;
```

React"元素"也是合法表达式

```
const subTitle = <p>hello, React</p>
const jsx = <h2>{subTitle}</h2>;
```

## 动态渲染UI

体验使用React.render更新UI

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
}

setInterval(tick, 1000);
```

## 条件渲染

`if` 语句以及 `for` 循环不是 JavaScript 表达式，所以不能在 JSX 中直接使用。但是，你可以用在 JSX 以外的代码中。可以基于上面结论实现。

### if语句

```
let isShowTitle = true;
if (isShowTitle) {
  title = <h1>是否显示h1</h1>;
}
```

### 逻辑与&&

```
const jsx = (
  <div>
    {isShowTitle && <h1>title1</h1>}
  </div>
);
ReactDOM.render(jsx, document.querySelector("#root"));
```

### 三元表达式 ||三目运算符

```
const showTitle = true;
const title = name ? <h2>{name}</h2> : null;
const jsx = (
  <div>
    {/* 条件语句 */}
    {title}
  </div>
);
```

### 循环列表 && key

数组会被作为一组子元素对待，数组中存放一组jsx可用于显示列表数据

```
const arr = [1,2,3].map(num => <li key={num}>{num}</li>)
const jsx = (
  <div>
    {/* 数组 */}
    <ul>{arr}</ul>
  </div>
);
```

### React Dom元素属性的使用

```
import logo from "./logo.svg";
import "index.css";

const box = {
  color:"blue",
  border:"1px blue solid"
}
const jsx = (
  <div style={box}>
    {/* 属性：静态值用双引号，动态值用花括号；class、for等要特殊处理。 */}
    <img src={logo} style={{ width:100,height:100,border:"1px red solid" }} className="width height" />
  </div>
);
```

> css模块化，创建index.module.css，index.js

```
import style from "./index.module.css";
<img className={style.img} />
<img className={`${style.font14} ${style.red}`} />
```

更多css modules规则参考

React的样式解决方案一直是个让人诟病的地方

**样式解决方案**

- styled-components
- styled-jsx
- classnames

# 组件

组件允许你将 UI 拆分为独立可复用的代码片段，并对每个片段进行独立构思。

组件类型

- 函数组件
- class组件

## 函数组件

定义组件最简单的方式就是函数组件，本质上就是JavaScript函数

函数组件通常**无状态**，仅**关注内容展示**，返回渲染结果即可

(从React16.8开始引入了**hooks**，函数组件也能够拥有状态，后面组件状态管理部分讨论)

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

## 组件Props

当 React 元素为用户自定义组件时，它会将 JSX 所接收的属性（attributes）转换为单个对象传递给组件，这个对象被称之为 “props”。

## 组件使用

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" msg={'消息'} arr=
{[1,2,3,4]} obj={{id:0}}/>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

**渲染过程分析：**

1. 我们调用 `ReactDOM.render()` 函数，并传入 `<Welcome name="Sara" />` 作为参数。
2. React 调用 `Welcome` 组件，并将 `{name: 'Sara'}` 作为 props 传入。

3. `Welcome` 组件将 `<h1>Hello, Sara</h1>` 元素作为返回值。
4. React DOM 将 DOM 高效地更新为 `<h1>Hello, Sara</h1>`。

## 组件注意事项

**组件名称必须以大写字母开头。**

React 会将以小写字母开头的组件视为原生 DOM 标签。例如，`<div />` 代表 HTML 的 div 标签，而 `<Welcome />` 则代表一个组件

return的内容只能有一个根节点，需要一个包裹元素，比如使用

，如果想返回多个兄弟元素，不想额外的嵌套，可以使用数组的方式 或者**Fragments**。

**所有 React 组件都必须像纯函数一样保护它们的 props 不被更改。**

```
-----------数组方式--------------
return [<h1/>,<h2/>,<h3/>]

-----------Fragments------------
return (
  <React.Fragment>
  <h1 />
  <h2 />
  <h3 />
  </React.Fragment>
);

//短语法
return (
  <>
  <h1 />
  <h2 />
  <h3 />
```

```
  </>
);
```

注意：使用显式 `<React.Fragment>` 语法声明的片段可能具有 key。

```
 <dl>
      {props.items.map(item => (
        // 没有`key`，React 会发出一个关键警告
        <React.Fragment key={item.id}>
          <dt>{item.term}</dt>
          <dd>{item.description}</dd>
        </React.Fragment>
      ))}
 </dl>
```

## class组件

使用 ES6 的 class 来定义组件，称为class组件

class组件通常**拥有状态**和**生命周期**，**继承于Component**，**实现render方法**

```
import React from "react";

class Welcome extends React.Component {
  //通过构造函数接受props,可以省略
  render() {
    return <h1>Hello, {this.props.name}</h1>;
    //return null
  }
}
```

`render` 方法直接返回 `null`，表示不进行任何渲染。

## 组件状态管理

如果组件中数据会变化，并影响页面内容，则组件需要拥有状态（state）并维护状态。

`props`是在父组件中指定，而且一经指定，不再改变。 对于需要改变的数据，我们需要使用`state`。

state状态改变，组件会重新调用render方法，更新UI

### 类组件中的状态管理

class组件通过state和setState维护状态

创建一个Clock组件，改造之前的tick案例

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    // 使用state属性维护状态，在构造函数中初始化状态
    this.state = { date: new Date() };
  }
  componentDidMount() {
    // 组件挂载时启动定时器每秒更新状态
    this.timerID = setInterval(() => {
      // 使用setState方法更新状态
      this.setState({
        date: new Date()
      });
    }, 1000);
  }
```

```
  componentWillUnmount() {
    // 组件卸载时停止定时器
    clearInterval(this.timerID);
  }
  render() {
    return <div>{this.state.date.toLocaleTimeString()}
</div>;
  }
}
```

拓展：**setState特性讨论**

- 不要直接修改state，要用setState更新状态

```
this.state.counter += 1; //错误的
```

- state的更新会被合并：当你调用 `setState()` 的时候，React 会把你提供的对象合并到当前的 state，这里的合并是浅合并
- state的更新可能是异步的
  - 对于多个setState执行，会合并成一个调用，因此对同一个状态执行多次只起一次作用，多个状态更新合并在一个setState中进行：

```
componentDidMount() {
// 假如couter初始值为0，执行三次以后其结果是多少?
this.setState({counter: this.state.counter + 1});
this.setState({counter: this.state.counter + 1});
this.setState({counter: this.state.counter + 1});
}
```

获取到最新状态值有以下三种方式：

1. 传递函数给setState方法：可以让 `setState()` 接收一个函数而不是一个对象。这个函数用上一个 state 作为第一个参数，将此次更

新被应用时的 props 做为第二个参数

```
this.setState((state, props) => ({  counter:
state.counter + 1}));// 1
this.setState((state, props) => ({  counter:
state.counter + 1}));// 2
this.setState((state, props) => ({  counter:
state.counter + 1}));// 3



//es5
this.setState(function(state, props) {
  return {
    counter: state.counter + 1
  };
});
```

2. 使用定时器:

```
setTimeout(() => {
  console.log(this.state.counter);
}, 0);
```

3. 原生事件中修改状态

```
componentDidMount(){
    document.body.addEventListener('click',
this.changeValue, false)
}

changeValue = () => {
    this.setState({counter: this.state.counter+1})
    console.log(this.state.counter)
}
```

# 事件处理

React 事件的命名采用小驼峰式（camelCase），而不是纯小写

使用 JSX 语法时你需要传入一个函数作为事件处理函数，而不是一个字符串

例如：onClick={activateLasers}

范例：用户输入事件，创建EventHandle.js

```
import React, { Component } from "react";

export default class EventHandle extends Component {
  constructor(props) {
    super(props);

    this.state = {
      name: ""
    };
```

```
    this.handleChange = this.handleChange.bind(this);
  }
  handleChange(e) {
    this.setState({ name: e.target.value });
  }
  render() {
    return (
      <div>
        {/* 使用箭头函数，不需要指定回调函数this，且便于传递参数
*/}
        {/* <input
          type="text"
          value={this.state.name}
          onChange={e => this.handleChange(e)}
        /> */}
        {/* 直接指定回调函数，需要指定其this指向，或者将回调设置为
箭头函数属性 */}
        <input
          type="text"
          value={this.state.name}
          onChange={this.handleChange(e)}
        />
        <p>{this.state.name}</p>
      </div>
    );
  }
}
```

事件回调函数注意绑定this指向，常见三种方法：

1. 构造函数中绑定并覆盖：this.textChange = this.textChange.bind(this)
2. 方法定义为箭头函数：textChange = ()=>{}
3. 事件中定义为箭头函数：onChange={()=>this.textChange()}

> react里遵循单项数据流，没有双向绑定，输入框要设置value和onChange，称为受控组件

# 组件通信

## Props属性传递 遵守单项数据流

Props属性传递可用于父子组件相互通信

如果父组件传递的是函数，则可以把子组件信息传入父组件，这个通常称为状态提升，StateMgt.js

```
// StateMgt
<Clock change={this.onChange}/>

// Clock
this.timerID = setInterval(() => {
    this.setState({
        date: new Date()
    }, ()=>{
        // 每次状态更新就通知父组件
        this.props.change(this.state.date);
    });
}, 1000);
```

## context

跨层级组件之间通信

> 主要用于组件库开发中，后面组件化内容中详细介绍

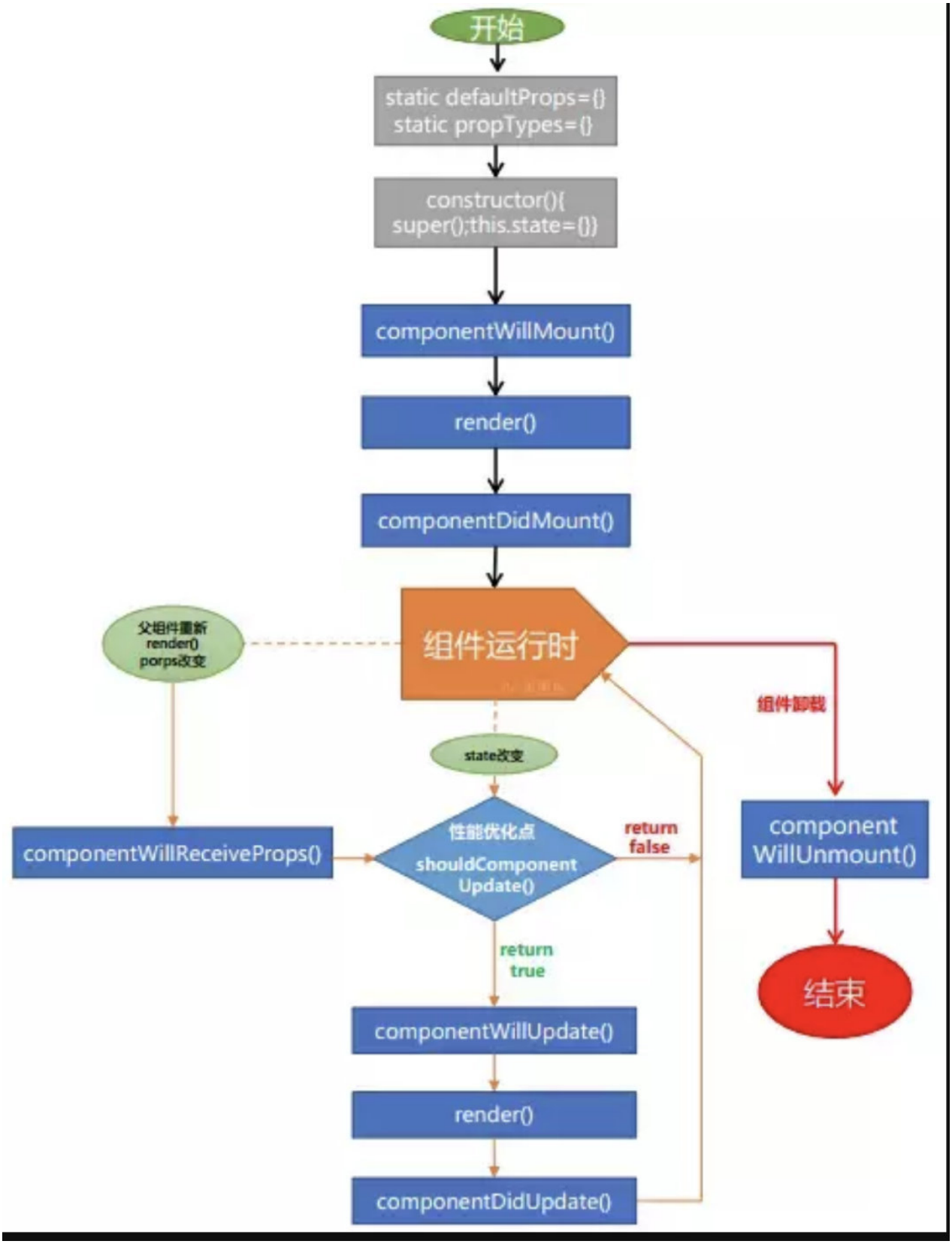
## redux

类似vuex，无明显关系的组件间通信

> 后面全家桶部分详细介绍

# 生命周期

React V16.3之前的生命周期




范例：验证生命周期，创建Lifecycle.js

```
import React, { Component } from "react";
export default class Lifecycle extends Component {
```

```
constructor(props) {
  super(props);
  // 常用于初始化状态
  console.log("1.组件构造函数执行");
}
componentWillMount() {
  // 此时可以访问状态和属性，可进行api调用等
  console.log("2.组件将要挂载");
}
componentDidMount() {
  // 组件已挂载，可进行状态更新操作
  console.log("3.组件已挂载");
}
componentWillReceiveProps(nextProps, nextState) {
  // 父组件传递的属性有变化，做相应响应
  console.log("4.将要接收属性传递");
}
shouldComponentUpdate(nextProps, nextState) {
  // 组件是否需要更新，需要返回布尔值结果，优化点
  console.log("5.组件是否需要更新？");
  return true;
}
componentWillUpdate() {
  // 组件将要更新，可做更新统计
  console.log("6.组件将要更新");
}
componentDidUpdate() {
  // 组件更新
  console.log("7.组件已更新");
}
componentWillUnmount() {
  // 组件将要卸载，可做清理工作
  console.log("8.组件将要卸载");
}
render() {
  console.log("组件渲染");
  return <div>生命周期探究</div>;
```

```
  }
}
```

> 组件生命周期在React v16.x之后的变化

## `Vue` 与 `React` 两个框架的粗略区别对比

Vue 的优势包括：

1. 模板和渲染函数的弹性选择
2. 简单的语法及项目创建
3. 更快的渲染速度和更小的体积

React 的优势包括：

1. 更适用于大型应用和更好的可测试性
2. 同时适用于 Web 端和原生 App
3. 更大的生态圈带来的更多支持和工具

### 相似之处

React 与 Vue 有很多相似之处，React 和 Vue 都是非常优秀的框架，它们之间的相似之处多过不同之处，并且它们大部分最棒的功能是相通的：如他们都是 JavaScript 的 UI 框架，专注于创造前端的富应用。不同于早期的 JavaScript 框架“功能齐全”，Reat 与 Vue 只有框架的骨架，其他的功能如路由、状态管理等是框架分离的组件。

- 两者都是用于创建 UI 的 JavaScript 库；
- 两者都快速轻便；
- 都有基于组件的架构；

- 都是用虚拟 DOM；
- 都可放入单个 HTML 文件中，或者成为更复杂 webpack 设置中的模块；
- 都有独立但常用的路由器和状态管理库；
- 它们之间的最大区别是 Vue 通常使用 HTML 模板文件，而 React 则完全是 JavaScript。Vue 有双向绑定语法糖。

## 不同点

- Vue 组件分为全局注册和局部注册，在 react 中都是通过 import 相应组件，然后模版中引用；
- props 是可以动态变化的，子组件也实时更新，在 react 中官方建议 props 要像纯函数那样，输入输出一致对应，而且不太建议通过 props 来更改视图；
- 子组件一般要显示地调用 props 选项来声明它期待获得的数据。而在 react 中不必需，另两者都有 props 校验机制；

- 每个 Vue 实例都实现了事件接口，方便父子组件通信，小型项目中不需要引入状态管理机制，而 react 必需自己实现；
- 使用插槽分发内容，使得可以混合父组件的内容与子组件自己的模板；
- 多了指令系统，让模版可以实现更丰富的功能，而 React 只能使用 JSX 语法；
- Vue 增加的语法糖 computed 和 watch，而在 React 中需要自己写一套逻辑来实现；

- react 的思路是 all in js，通过 js 来生成 html，所以设计了 jsx，还有通过 js 来操作 css，社区的 styled-component、jss 等；而 vue 是把 html，css，js 组合到一起，用各自的处理方式，vue 有单文件组件，可以把 html、css、js 写到一个文件中，html 提供了模板引擎来处理。

- react 做的事情很少，很多都交给社区去做，vue 很多东西都是内置的，写起来确实方便一些， 比如 redux 的 combineReducer 就对应 vuex 的 modules， 比如 reselect 就对应 vuex 的 getter 和 vue 组件的 computed， vuex 的 mutation 是直接改变的原始数据，而 redux

的 reducer 是返回一个全新的 state，所以 redux 结合 immutable 来优化性能，vue 不需要。

- react 是整体的思路的就是函数式，所以推崇纯组件，数据不可变，单向数据流，当然需要双向的地方也可以做到，比如结合 redux-form，组件的横向拆分一般是通过高阶组件。而 vue 是数据可变的，双向绑定，声明式的写法，vue 组件的横向拆分很多情况下用 mixin。

# React16 新特性

## 1 引言

于 2017.09.26 Facebook 发布 React v16.0 版本，时至今日已更新到 React v16.6，且引入了大量的令人振奋的新特性，本文章将带领大家根据 React 更新的时间脉络了解 React16 的新特性。

## 2 概述

按照 React16 的更新时间，从 React v16.0 ~ React v16.6 进行概述。

**React v16.0**

- render 支持返回数组和字符串、Error Boundaries、createPortal、支持自定义 DOM 属性、减少文件体积、fiber；

**React v16.1**

- react-call-return；

**React v16.2**

- Fragment；

**React v16.3**

- createContext、createRef、forwardRef、生命周期函数的更新、Strict Mode；

**React v16.4**

- Pointer Events、update getDerivedStateFromProps；

**React v16.5**

- Profiler；

**React v16.6**

- memo、lazy、Suspense、static contextType、static getDerivedStateFromError()；

**React v16.7（~Q1 2019）**

- Hooks；

**React v16.8（~Q2 2019）**

- Concurrent Rendering；

**React v16.9（~mid 2019）**

- Suspense for Data Fetching；

下面将按照上述的 React16 更新路径对每个新特性进行详细或简短的解析。

## React DevTools

React开发者工具，目前可以在 Chrome，Firefox 以及 （Chromium） Edge 中使用！

chrome浏览器中安装，需要科学上网

```
//进入扩展程序，打开网上应用商店
chrome://extensions/

//搜索 React DevTools
//安装 点击添加至chrome 即可
//状态 有三种颜色，三种颜色代表三种状态
灰色： 这种就是不可以使用，说明页面不是React编写的。
黑色： 说明页面是用React编写的，并且处于生成环境当中。
红色： 说明页面是用React编写的，并且处于调试环境当中。
```

# 下节课

组件设计思想

# 回顾

React

Imr 导入react

imrc 导入react component

imrd 导入React.Dom

cc 创建组件

ccc 创建构造函数组件

sfc 创建函数组件