**什么是React？**

React 是一个用于构建用户界面的 JAVASCRIPT 库。

React主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）。

React 起源于 Facebook 的内部项目，用来架设 Instagram 的网站，并于 2013 年 5 月开源。

React 拥有较高的性能，代码逻辑非常简单，越来越多的人已开始关注和使用它。

既可以开发浏览器应用，又可以开发移动应用

**React 特点**

1.声明式设计 −React采用声明范式，可以轻松描述应用。

2.高效 −React通过对DOM的模拟，最大限度地减少与DOM的交互。

3.灵活 −React可以与已知的库或框架很好地配合。

4.JSX − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。

5.组件 − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。

6.单向响应的数据流 − React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。(数据的流向是单向的——从父节点传递到子节点，因为组件是简单而且易于把握的，他们只需从父节点获取props渲染即可，如果顶层组件的某个prop改变饿了，React会递归的向下便利整棵组件树，重新渲染所有使用这个属性的组件。)

**什么是JSX？**

React 使用 JSX（JavaScript XML） 来替代常规的 JavaScript。JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。我们不需要一定使用 JSX，但它有以下优点：

* JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。
* 它是类型安全的，在编译过程中就能发现错误。
* 使用 JSX 编写模板更加简单快速。

**State（状态)**

React 把组件看成是一个状态机（State Machines）。通过与用户的交互，实现不同状态，然后渲染 UI，让用户界面和数据保持一致。

React 里，只需更新组件的 state，然后根据新的 state 重新渲染用户界面（不要操作 DOM）。

不需要直接操作DOM，而是通过修改state，自动更新界面

**Props（属性）**

state 和 props 主要的区别在于 props 是不可变的，而 state 可以根据与用户交互来改变。这就是为什么有些容器组件需要定义 state 来更新和修改数据。 而子组件只能通过 props 来传递数据。

可以通过 getDefaultProps() 方法为 props 设置默认值

**React组件生命周期**

**实例化**

**首次实例化**

* getDefaultProps- 貌似没了
* getInitialState- 貌似没了
* componentWillMount
* render
* componentDidMount

**实例化完成后的更新**

* getInitialState
* componentWillMount
* render
* componentDidMount

**存在期**

组件已存在时的状态改变

* componentWillReceiveProps
* shouldComponentUpdate
* componentWillUpdate
* render
* componentDidUpdate

**销毁&清理期**

* componentWillUnmount

生命周期共提供了10个不同的API。

**1.getDefaultProps**

作用于组件类，只调用一次，返回对象用于设置默认的props，对于引用值，会在实例中共享。

**2.getInitialState**

作用于组件的实例，在实例创建时调用一次，用于初始化每个实例的state，此时可以访问this.props。

**3.componentWillMount**

在完成首次渲染之前调用，此时仍可以修改组件的state。

**4.render**

必选的方法，创建虚拟DOM，该方法具有特殊的规则：

* + 只能通过this.props和this.state访问数据
  + 可以返回null、false或任何React组件
  + 只能出现一个顶级组件（不能返回数组）
  + 不能改变组件的状态
  + 不能修改DOM的输出

**5.componentDidMount**

真实的DOM被渲染出来后调用，在该方法中可通过this.getDOMNode()访问到真实的DOM元素。此时已可以使用其他类库来操作这个DOM。

在服务端中，该方法不会被调用。

**6.componentWillReceiveProps**

组件接收到新的props时调用，并将其作为参数nextProps使用，此时可以更改组件props及state。

componentWillReceiveProps: function(nextProps) {

if (nextProps.bool) {

this.setState({

bool: true

});

}

}

**7.shouldComponentUpdate**

组件是否应当渲染新的props或state，返回false表示跳过后续的生命周期方法，通常不需要使用以避免出现bug。在出现应用的瓶颈时，可通过该方法进行适当的优化。

在首次渲染期间或者调用了forceUpdate方法后，该方法不会被调用

**8.componentWillUpdate**

接收到新的props或者state后，进行渲染之前调用，此时不允许更新props或state。

**9.componentDidUpdate**

完成渲染新的props或者state后调用，此时可以访问到新的DOM元素。

**10.componentWillUnmount**

组件被移除之前被调用，可以用于做一些清理工作，在componentDidMount方法中添加的所有任务都需要在该方法中撤销，比如创建的定时器或添加的事件监听器。



