面试

今日课程内容

- 简历编写
- 面试前准备
- 手写简易 React

简历编写

以面试官的角度编写自己的简历

- 内容简单易懂、重点突出
- 关键字要密集

简历板块划分

个人信息

- 姓名
- 年龄
- 教育背景
- 联系方式
- 工作年限(选填)
- 形象照（选填）
- 输出内容（笔记、博客、github）

个人技术栈

- 忌造假。可夸张，但不可以浮夸
- 最好以数据为支撑

工作经历

- 公司介绍
- 职位、负责内容
- 优秀成绩(选填)
- 非专业工作经历可简写

项目经历

- 项目背景
- 负责模块
- 涉及技术栈

没有工作经验简历怎么写

- 表现自己的学习能力
- 在社群活跃
- 收到认可(点赞、观看、转载)
- 个人作品

手写 React

- React 虚拟 DOM 生成

```
  function createElement(type,props,...children){
      return {
          type,
          props,
          children
      }
  }
```

- 添加 class 组件

```
class Component {
      constructor(props){
          this.props = props;
          this.state = {}
      }
      setState(newState){
          this.state = {...this.state,...newState};
          this.updater();
      }
}
```

- DOM.render 将虚拟 DOM 挂载到真实 DOM

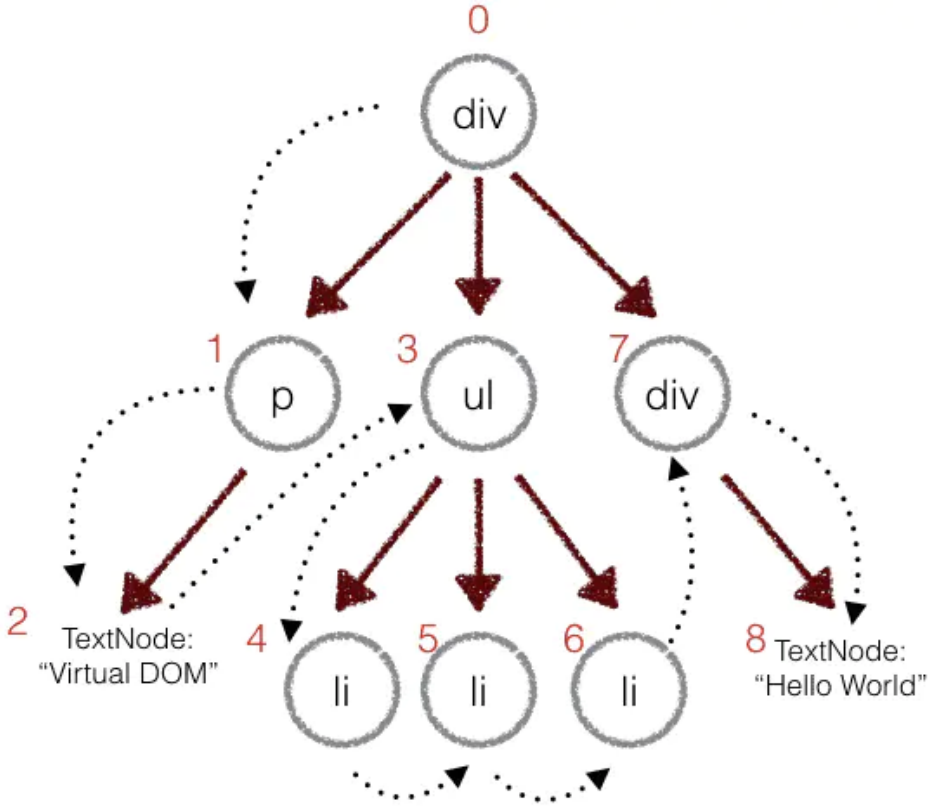
```
  // render 将虚拟 DOM 挂载到真实DOM上
  function isObj(data){
      return typeof data === "object";
  }
  function render(vnode,container){
      let dom = createNode(vnode);
      container.appendChild(dom);
  }
  // 根据虚拟 DOM 生成真实DOM
  function createNode(vnode){
      if(!isObj(vnode)){ // 文本节点
          return document.createTextNode(vnode);
      }
      if(typeof vnode.type === "function"){ // 组件
          return createClass(vnode);
      }
      let node = document.createElement(vnode.type);
      createProps(node,vnode.props);
      vnode.children.forEach(child => {
          render(child,node);
      });
      return node;
  }
  // 生成 props
  function createProps(node,props){
      let key = Object.keys(props);
      key.forEach(k=>{
          if(k === "style"){
```

```
                for(let attr in props[k]){
                    node.style[attr] = props[k][attr];
                }
            } else if(k.slice(0,2)==="on"){

node.addEventListener(k.slice(2).toLocaleLowerCase(),props[k]);
            } else {
                node[k] = props[k]
            }
        });
    }
    // 初始化组件
    function createClass(vnode){
        let cmp = new vnode.type(vnode.props);
        cmp.vnode = cmp.render(vnode.props);
        cmp.dom = createNode(cmp.vnode);
        // 监听组件更新
        cmp.updater = function(){
            let newNode = cmp.render();
            let newDom = createNode(newNode)
            cmp.dom.parentNode.replaceChild(newDom,cmp.dom);
            cmp.vnode = newNode;
            cmp.dom = newDom;
        };
        return cmp.dom;
    }
```

扩展内容 Diff

- react 中的 diff，`https://zhuanlan.zhihu.com/p/20346379`
- 在 React 会对比新旧虚拟DOM，生成一个补丁包，然后根据补丁包进行 DOM 更新。
- 手写简易diff



手写简易 diff

```
const TEXT = "TEXT";
const REPLACE = "REPLACE";
const ATTRS = "ATTRS";
let index = 0;
function notObj(data){
    return typeof data !== "object";
}
function diff(oldTree,newTree){
    let pacthes={};
    index = 0;
    walk(oldTree,newTree,pacthes,index);
    return pacthes;
}
function walk(oldNode,newNode,pacthes,index){
    let currentPacthes = [];
    // 判断是否是同一类型，不是直接替换
    if(notObj(oldNode)&&notObj(newNode)){
        if(oldNode!==newNode){
            currentPacthes.push({
                type:TEXT,
                newNode
            })
        }
    }else if(oldNode.type === newNode.type){
        // 对比属性是否一致
        let attrs = diffAttrs(oldNode.props,newNode.props);
        if(Object.keys(attrs).length>0){
            currentPacthes.push({
                type:ATTRS,
                attrs
            });
        }
        // 对比子节点
        diffChild(oldNode.children,newNode.children,pacthes);
    } else {
        currentPacthes.push({
            type:REPLACE,
            newNode
        })
    }
    if(currentPacthes.length>0){
        pacthes[index] = currentPacthes;
    }
}
//对比子节点不完整版
function diffChild(oldChildren,newChildren,pacthes){
    oldChildren.forEach((node,idx) => {
        ++index;
        walk(node,newChildren[idx],pacthes,index);
    });
}
function diffAttrs(oldAttrs,newAtts){
    let attrs = {};
    for(let key in oldAttrs){
        if(oldAttrs[key]!==newAtts[key]){
            attrs[key] = newAtts[key]
        }
```

```
    }
    for(let key in newAtts){
        if(!oldAttrs.hasOwnProperty(key)){
            attrs[key] = newAtts[key];
        }
    }
    return attrs;
}
export default diff;
```

手写简易 patches

```
let index = 0;
let allPatches = {};
function patches(dom,patches){
    allPatches = patches;
    index = 0;
    walk(dom,index);
}
function walk(dom,idx){
    let children = [...dom.childNodes];
    children.forEach(child=>{
        index++;
        walk(child,index);
    });
    if(allPatches[idx]){
        toPatches(dom,allPatches[index]);
    }
}
function toPatches(dom,patches){
    patches.forEach(p => {
        switch(p.type){
            case "TEXT":
                console.log(p.newNode);
                dom.textContent = p.newNode;
                break;
            case "REPLACE":
                dom.replaceChild(p.newNode,dom);
                break;
            case "ATTRS":
                patchAttr(dom,p.attrs);
                break;
        }
    });
}
function patchAttr(node,props){
    let key = Object.keys(props);
    key.forEach(k=>{
        if(k === "style"){
            for(let attr in props[k]){
                node.style[attr] = props[k][attr];
            }
        } else if(k.slice(0,2)==="on"){
            node.addEventListener(k.slice(2).toLocaleLowerCase(),props[k]);
        } else {
            node[k] = props[k]
        }
```

```
    });
}
export default patches;
```

修改组件更新代码

```
cmp.updater = function(){
    let newNode = cmp.render();
    let AllPacthes = diff(cmp.vnode,newNode);
    if(Object.keys(AllPacthes).length>0){
        for(let key in AllPacthes){
            AllPacthes[key].forEach(p=>{
                if(p.type == "REPLACE"){
                    p.newNode = createNode(p.newNode);
                }
            });
        }
        patches(cmp.dom,AllPacthes);
        cmp.vnode = newNode;
    }
};
```