3.设计思路与流程架构

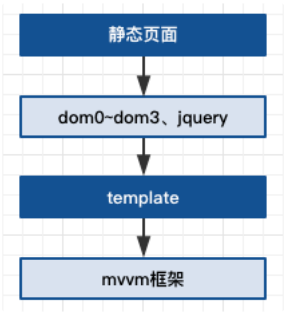
> 珠峰前端架构师技术分享课 <https://ke.qq.com/course/272058>

课前思考题

- react为什么诞生，为了解决什么问题？
- 除了你所熟知的dom-diff,virtual dom等概念之外，思考过react还有其他什么创新么？
- 猜想一下，react内部如何工作。

设计思路

在mvvm框架没有流行之前，前端经历过以下几个黑暗时期：



jquery和template大多数同学已经很熟悉，简单易学易用。

但是缺点也很明显，不仅操作dom非常繁琐，而且在大量修改dom节点性能低下。

后来产生了template，让各位前端coder眼前一亮，根据数据渲染页面，而不用js操作dom，在jq那个年代，这不就是我想要的么？

可是，每次数据变化，仍然需要操作template全量渲染（更新模板innerHTML），即使teaplate拆的很细，仍然保证不了有没有变化的dom渲染。

React设计灵感

react最初的设计灵感源于游戏，游戏渲染的机制：当数据变化时，界面仅仅更新变化的一部分形成新的一帧渲染。之前的jq和template显然无法做到这一点。

设计react的核心是认为UI只是把数据通过映射关系变换成另一种形式的数据，也就是展现方式。传统上，web框架使用模板或者html指令构造页面。

react处理构建用户界面通过将他们分解为虚拟组件，虚拟组件是react的核心，整个react框架的设计理念，都是围绕虚拟组件进行的。

1. 组件

在界面变更时，react不直接操作dom，而是通过组件对比(dom-diff)，找到界面更新前后组件的差异，而只更新有变化的dom节点。现有技术 <https://reactjs.org/docs/reconciliation.html> 的时间时间复杂度O(n^3)，n是页面上的元素，这意味着如果页面上有1000个节点，会对比10亿次。react重写了dom-diff算法，把时间复杂度降低到O(n)。

react把组件分成了三类，三类合称为：React Component，它另一个名字我们也许更加熟悉：**Virtual DOM**。

1.1 ReactTextComponent

文字组件:

```
<div>
  <span>hello</span>
  world
</div>
```

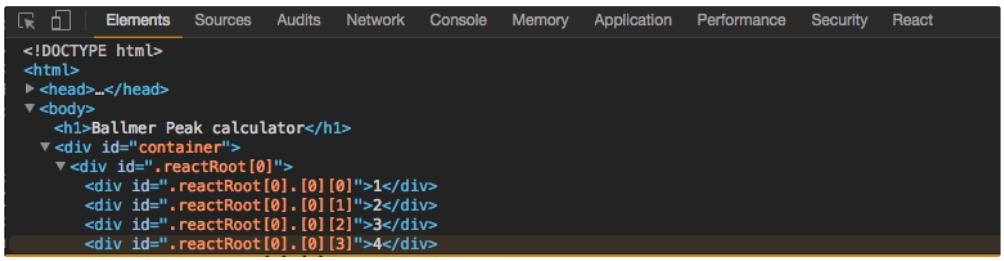
这里，world就是文字组件，为什么单独给world设置成文字组件？每个文字组件外面，都需要包装一层span用来设置id（或者key），用来diff。

1.2 ReactNativeComponent

html原生组件:

```
<div>
  <span>hello</span>
  world
</div>
```

还是刚才那个例子，div和span都是原生组件。react的主要算法（如mount、dom-diff）都是在这里实现。用来dom-diff的在渲染阶段会直接设置在原生组件标签上。就像下面这样：



1.3 ReactCompositeComponent

react复合组件：

```
<Table>
  <Table.Columns .../>
  ...
</Table>
```

这次我们来换个例子，复合组件就是我们常见的react组件，只不过在react内部叫做复合组件。而react组件是三种组件的合称。

复合组件有以下特点：复合组件可以聚合其他的复合组件与原生组件，但是最底层的复合组件一定只能聚合原生组件。复合组件通常以大写开头。

1.4 组件实现

每个组件都有自己的props和children，props就是组件的属性，比如style，id等。children是当前组件的子组件。比如：

```
<Form>
  <Form.Item>
    用户名:
    <input placeholder="请输入用户名" />
  </Form.Item>
</Form>
```

这段jsx会创建3个组件：Form,Form.Item,input，前两个是复合组件，最后一个是原生组件，连小学生都能看懂他们的关系：

- Form<ReactCompositeComponent>
  - children<Array>: [Form.Item<ReactCompositeComponent>]
- Form.Item<ReactCompositeComponent>
  - children: [span<ReactTextComponent> ，input<ReactNativeComponent>]
- span<ReactNativeComponent>
  - children: string<String>
- input<ReactNativeComponent>
  - props: { placeholder: string<String> }
  - children: []

在众多教材中，这组树形结构被描述成为一个json:

```
{
    "component": Form<ReactCompositeComponent>,
    "children": [{
        "component": Form.Item<ReactCompositeComponent>,
        "children": [{
            "component": span<ReactNativeComponent>,
            "children": string<String>
        }, {
            "component": input<ReactNativeComponent>,
            "props": { placeholder: string<String> },
            "children": []
        }]
    }]
}
```

每个虚拟组件都有自己的children，这样一来，从container元素开始的dom树就有一颗结构相同的虚拟组件树。

2. dom diff

在实际项目中，随着页面数据变化（用户交互）或者后端数据返回，更新大多数只有三种情况：

- dom的属性或者内容更新（update）。
- dom元素类型发生变化(insert)。
- dom元素的位置发生变化，或者新增(insert)，后者删除(remove)。

当然还有第四种：对于跨层级的移动更新少之又少，比如像这种

```
<div>
  <title>子标题</title>
  <input />
</div>
```

更新成

```
<title>标题</title>
<div>
  <input></span>
</div>
```

当然一些业务需要时（比如某同学转班或者能从左表格移动到右表格的穿梭框），我们可以认为它是dom删除与dom插入两个操作，而不是一次move（insertBefore），因为这种业务并不多见。

大多数情况下，我们只需要diff同一子节点下面的元素变化情况，比如：

```
<div>
  <h1>h1</h1>
  <h2>h2</h2>
  <h3>h3</h3>
</div>
```

更新成

```
<div>
  <h2>h2</h2>
  <p>p</p>
  <h3>new h3</h3>
  <h1>h1</h1>
</div>
```

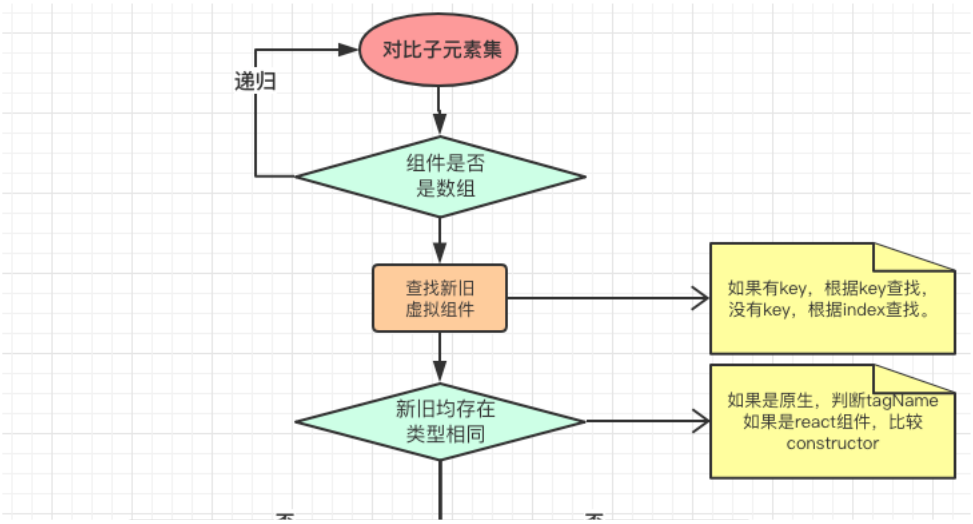
react是如何做到的呢？

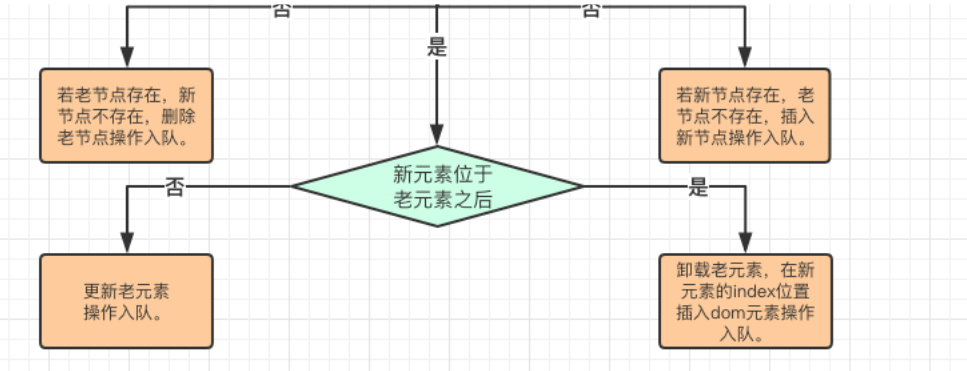
最外层的div标签一致，然后对比div的children。

- 循环第一个新节点h2和老节点h1，发现节点类型不同，删除掉h1，插入h2。
- 同理，p不等于h2，删除h2，插入p。
- 继续，h3等于h3，更新h3的内容new h3。
- 最后，发现之前没有h1，插入h1。

这就是一个最简单的dom-diff结果。真实的dom-diff比这个复杂一点点，原理大体相似，我们先做个试验 <https://github.com/antgod/my-react/blob/master/src/App.js> ，来验证下流程：

- 经过判断和操作两个步骤。





- 特别说明
  - 如果一个有key,一个没有key,仍然是新增节点。比如新节点没有key，老节点有key，仍然执行插入操作。
  - 如果能找到key，但是key的顺序不一致，则使用key去老的children按照顺序遍历，当老的children顺序遍历完，所有新节点全部重新插入。比如abcdef改成了bcfdea，则bcf在老节点都能找到，而f已经是老节点最后一个节点，所以之后的节点都是重新插入。
  - 以数组范围的作用域，计算结果

3. 层次

3.1 展示层

用户声明、创建虚拟组件以及渲染虚拟组件。

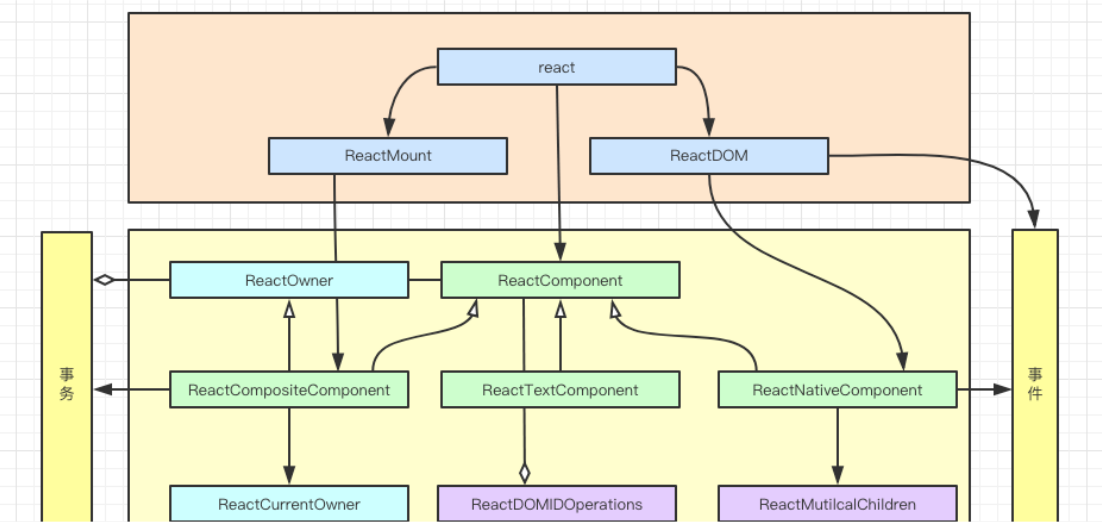
3.2 Virtual组件层

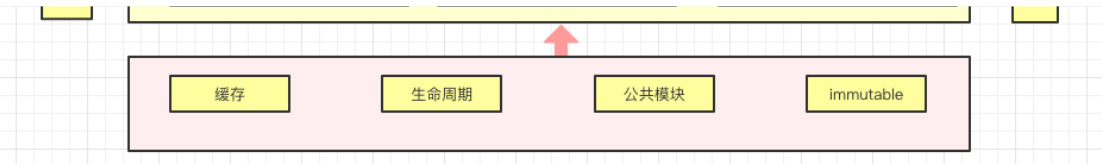
react的绝大多数代码，包括虚拟组件的创建、渲染和更新流程dom-diff。

3.3 基础功能

所有公共代码，提供底层功能。

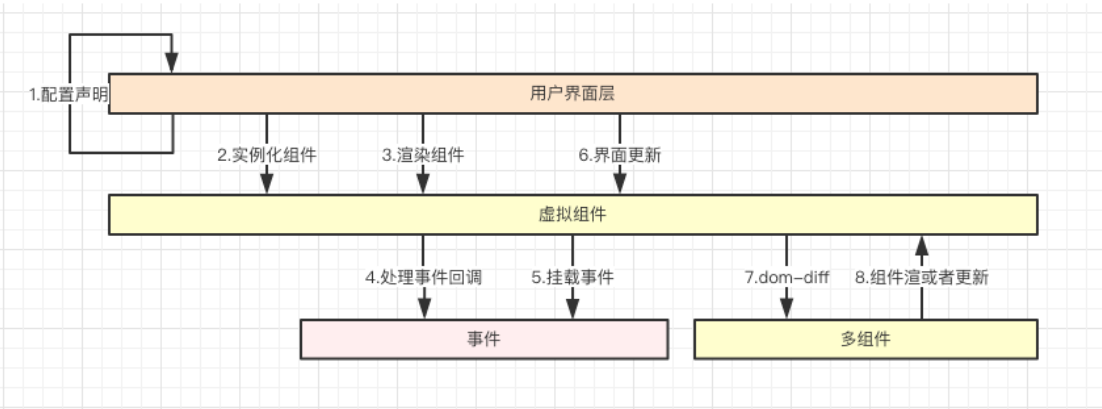
整体架构





- ReactMount: 顾名思义，负责react组件的渲染
- ReactDOM: 用来创建native组件
- ReactComponent: react虚拟组件
- ReactOwner\ReactCurrentOwner: 父子组件指针
- ReactDOMIDOperations: 真实的dom操作
- ReactMuticalChildren: dom-diff的实现
- 功能层（黄色部分）：为react业务提供基础功能，第六章会详细讲解。

模块调用关系以及流程图



- 1. 配置声明：用户代码
- 2. 组件实例化：ReactComponsiteComponent.createClass
- 3. 组件渲染：ReactMount.renderComponent
- 4. 处理事件回调：renderComponent->ReactMount.prepareTopLevelEvents
- 5. 挂载事件：ReactNativeComponent._updateDOMProperties->ReactEvent.putListener
- 6. 界面更新：ReactComposite.setState | ReactComponsite.receiveProps
- 7. dom-diff: receiveProps -> ReactMutiChild.updateMultiChild
- 8. 组件渲染或更新：ReactDOMIDOperations->DOMChildrenOperations