ReactJS基础概览

1, 介绍react，明白react是什么，要解决什么问题

2，介绍一些react的基本概念，建立起react的整体印象

react是什么？

React是Facebook开发的一款JS库

那么Facebook为什么要建造React呢，主要为了解决什么问题，通过这个又是如何解决的？

Facebook认为市面上流行的MVC框架无法满足他们的扩展需求，由于他们非常巨大的代码库和庞大的组织，使得代码很快变得非常复复杂，每当需要添加一项新的功能或特性时，系统的复杂度就成级数增长，致使代码变得脆弱和不可预测，结果导致他们的架构正在土崩瓦解。认为MVC不适合大规模应用，当系统中有很多的模型和相应的视图时，其复杂度就会迅速扩大，非常难以理解和调试，特别是模型和视图间可能存在的双向数据流动。

解决这个问题需要“以某种方式组织代码，使其更加可预测”，于是他们创造了react。

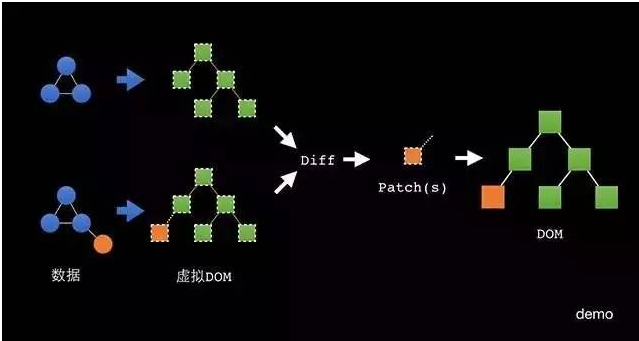
使用这个库可以很方便的开发交互式的、具有表达力的和可重用的UI组件。它本身并不是一个框架，它是视图层，并且是一个以组件为基础的高效视图。对于React应用而言，你需要分割你的页面，使其成为一个个的组件。也就是说你的应用是由一个个组件构成的。这种分割、复用组件的方式开发页面，我们称之为****组件驱动开发****。

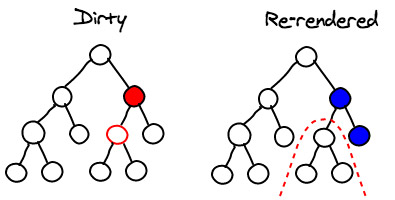
这个库还使用了一种叫做****虚拟DOM****（Virtual DOM）的概念，这些DOM可以根据状态有选择的渲染。这样，页面就会尽量的减少DOM操作而达到保持页面状态的效果。

虚拟DOM

在Web开发中，我们总需要将变化的数据实时反应到UI上，这时就需要对DOM进行操作。而****复杂或频繁的DOM操作通常是性能瓶颈产生的原因****（如何进行高性能的复杂DOM操作通常是衡量一个前端开发人员技能的重要指标）。React为此引入了****虚拟DOM（Virtual DOM）****的机制：在浏览器端用Javascript实现了一套DOM API。基于React进行开发时所有的DOM构造都是通过虚拟DOM进行，每当数据变化时，React都会重新构建整个DOM树，然后React将当前整个DOM树和上一次的DOM树进行对比，得到DOM结构的区别，然后仅仅将需要变化的部分进行实际的浏览器DOM更新。而且React能够批处理虚拟DOM的刷新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如你连续的先将节点内容从A变成B，然后又从B变成A，React会认为UI不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟DOM树，但是因为虚拟DOM是内存数据，性能是极高的，而对实际DOM进行操作的仅仅是Diff部分，因而能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的DOM元素，而只需要关心在任意一个数据状态下，整个界面是如何Render的。

如果你像在很久以前那样写过服务器端Render的纯Web页面那么应该知道，服务器端所要做的就是根据数据Render出HTML送到浏览器端。如果这时因为用户的一个点击需要改变某个状态文字，那么也是通过刷新整个页面来完成的。服务器端并不需要知道是哪一小段HTML发生了变化，而只需要根据数据刷新整个页面。换句话说，任何UI的变化都是通过整体刷新来完成的。而React将这种开发模式以高性能的方式带到了前端，每做一点界面的更新，你都可以认为刷新了整个页面。至于如何进行局部更新以保证性能，则是React框架要完成的事情。





组件化

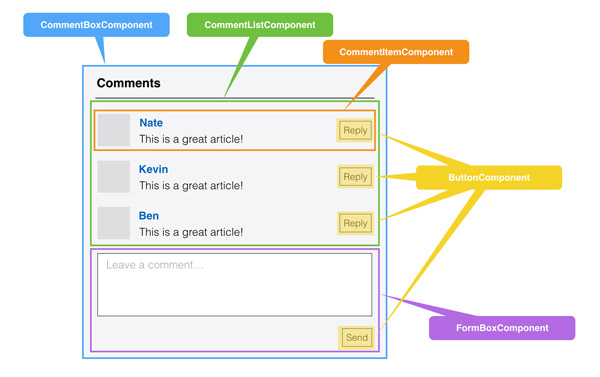
虚拟DOM(virtual-dom)不仅带来了简单的UI开发逻辑，同时也带来了组件化开发的思想，所谓组件，即封装起来的具有独立功能的UI部件。React推荐以组件的方式去重新思考UI构成，将UI上每一个功能相对独立的模块定义成组件，然后将小的组件通过组合或者嵌套的方式构成大的组件，最终完成整体UI的构建。例如，Facebook的instagram.com整站都采用了React来开发，整个页面就是一个大的组件，其中包含了嵌套的大量其它组件，大家有兴趣可以看下它背后的代码。

如果说MVC的思想让你做到视图-数据-控制器的分离，那么组件化的思考方式则是带来了UI功能模块之间的分离。我们通过一个典型的Blog评论界面来看MVC和组件化开发思路的区别。

对于MVC开发模式来说，开发者将三者定义成不同的类，实现了表现，数据，控制的分离。开发者更多的是从技术的角度来对UI进行拆分，实现松耦合。

对于React而言，则完全是一个新的思路，开发者从功能的角度出发，将UI分成不同的组件，每个组件都独立封装。

在React中，你按照界面模块自然划分的方式来组织和编写你的代码，对于评论界面而言，整个UI是一个通过小组件构成的大组件，每个组件只关心自己部分的逻辑，彼此独立。



React认为一个组件应该具有如下特征：

****（1）可组合（Composeable）****：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件；

****（2）可重用（Reusable）****：每个组件都是具有独立功能的，它可以被使用在多个UI场景；

****（3）可维护（Maintainable）****：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；

单向数据流

在jquery时代，我们都是基于事件驱动，对于简单的交互需求而言，这确实足够了，而且开发起来非常迅速。但业务一旦复杂，这种基于事件驱动的东西就会变得很乱，页面需要更新的DOM很多，就容易出错。

单向数据流的概念就出现了。更新 DOM 的数据总是从顶层流下来，用户事件不直接操作 DOM，而是操作顶层数据。这些数据从顶层流下来同时更新了DOM。你的代码就很少会直接处理DOM，而是只处理数据的变更。这样会很大程度上简化代码和逻辑。

举个例子：我点击一个button，然后页面上一个span里数字+1，原有的思考逻辑是“点击发生，然后数据变化，然后UI跟着变化+1”。而现在的思考逻辑是我的数据变化了，那么我的UI会自动更新，那么我只用考虑“点击发生，数据变化”。即可

在单个组件内，ui的变化通过state来控制

在多个组件嵌套中，父组件需要负责管理状态，并把数据通过props向下发放。

组件的状态通过setState方法更新。数据通过设置子组件的属性来传递给子组件，子组件通过this.props来获取这些数据

这就是单向数据流

****Jsx****

HTML 语言直接写在 JavaScript 语言之中，不加任何引号，这就是 JSX 的语法，它允许 HTML 与 JavaScript 的混写

****我们将用它来编写组件的ui部分****