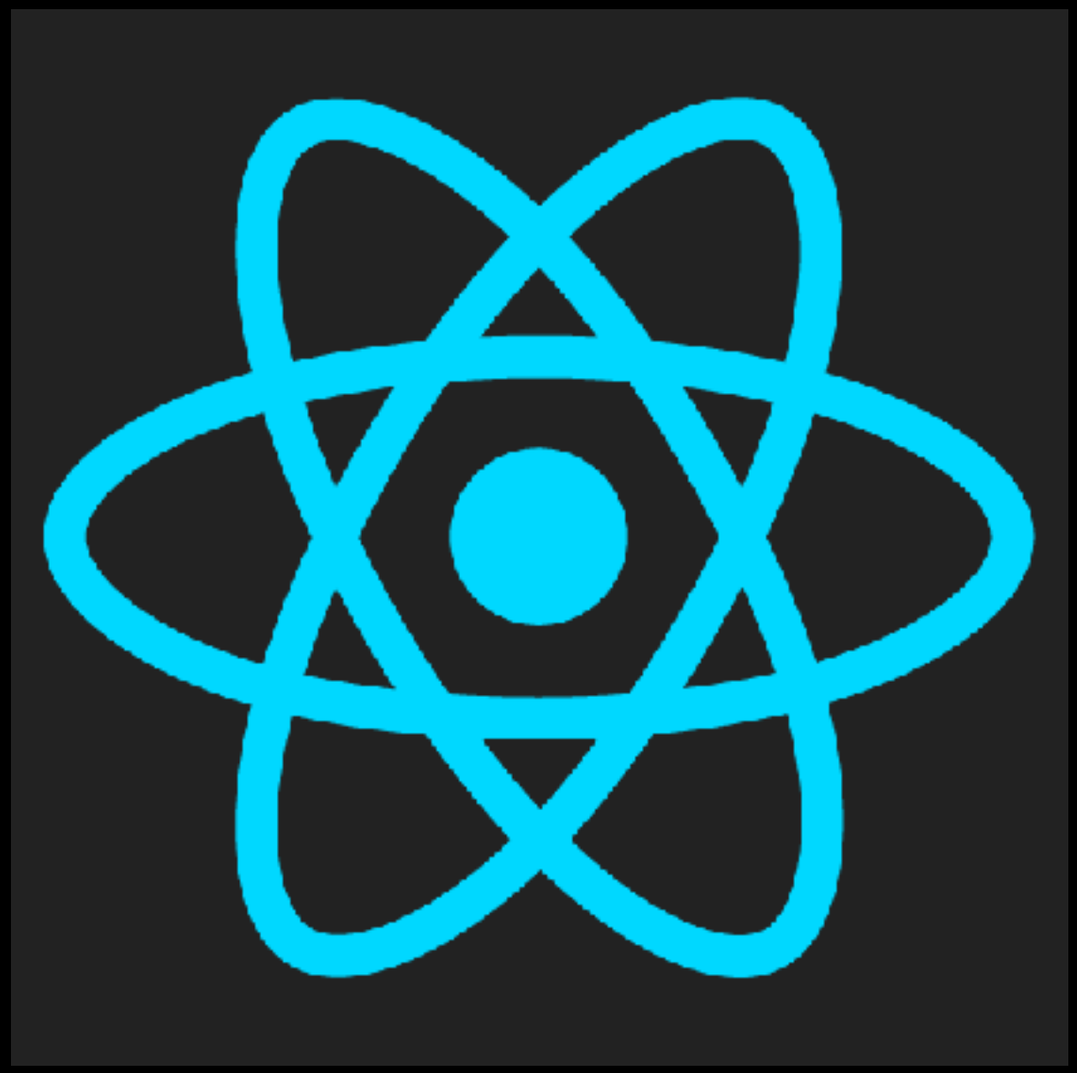
React开发入门

参考《React开发实战》笔记

React定义

- DOM抽象
- 响应式UI渲染引擎
- 创建可组合用户界面
- 使用JavaScript和XML技术



React优点

- 灵活的文档模型抽象表现。
- 响应式渲染，当数据发生变化的时候，在概念上React会重新渲染整个用户界面。
- 面向组件开发，使组件成为自包含的、并在相关视图逻辑中使用统一的标记，实现关注点的分离。
- React的革新在于融合HTML和JavaScript来描述组件，并通过使用周全的、隔离的、可复用的、可组合的组件来实现概念的分离。

DOM抽象

React组件中包含UI定义（内容标记）、交互（JavaScript）和样式（内联样式、CSS样式表两种）3部分。

- JSX

  JSX是React对JavaScript的语法扩展，用于在JavaScript代码中编写声明式XML风格语法。

- 事件

  合成事件系统，一致性为标准化事件（触摸和鼠标事件onClick，键盘事件onKeyPress，焦点和表单事件onSubmit，其他事件onScroll），并通过事件委托实现高性能。

深入了解JSX

React的JSX提供了一组类似于HTML的XML标签，转以后，XML会被转换为针对React库的函数调用。也存在针对性的React with SVG、React Canvas和React Native。

优势：

1. XML包含特性的元素树非常适合表示UI；
2. 能够更精确和更方便的呈现应用程序的结构；
3. 本质上是普通的JavaScript，不会改变JavaScript语义。

不同：

1. 足够像HTML，但不是HTML；
2. 标签特性采用驼峰大小写风格；
3. 所有元素必须闭合；
4. 特性名称基于DOM API而非HTML语言规范。

怪异：

1. 单一根节点，受限于JavaScript特性，一条返回语句只能返回单个值，对应一个节点；
2. 条件语句的使用，受限于转化后的JavaScript，内部可以直接使用三元等表达式，但if等结构性语句需要外置；
3. 不支持HTML注释（<!--comment-->），需要使用JavaScript注释；
4. 在多行元素之间不会渲染空格，需要显式写明（{" "}）；
5. 内置XSS攻击保护（可以使用dangerouslySetInnerHTML跳过）。

内联样式

可以将内联样式定义为一个JavaScript对象，样式名称使用驼峰大小写和DOM属性保持一致。使用内联样式存在以下优势：

- 不需要选择器来限定样式的范围；
- 避免特异性冲突；
- 源顺序无关。

虚拟DOM

React的关键设计决策是让API看起来像是要在每次更新时都渲染整个应用程序，但并没有在应用程序状态每次发生变化时都更新真实的DOM。而是在内部构建了一个对应应用程序状态的虚拟DOM，通过子级校正（reconciliation）来计算最小差异（Diff）进行增量更新。为了提高该算法的性能，做了如下假设：

1. 在比较DOM树中的节点时，如果节点类型不同，React会视为两个不同的子树，丢弃第一个，并插入第二个。

2. 自定定组件也会使用相同的逻辑。

3. 如果是同类型DOM元素，只修改特性和样式，而不是替换元素树。

4. 如果是同类型自定义组件，刚将新属性发送给当前挂载的组件，并在组件上触发一次新的rende方法，使用新结果重新进行初始化。

key属性

在大列表中，节点可被插入：删除、替换和移动，各种可行方案都有可能导致副作用，很难用一种算法来确定所有可能性中最好的方案。

为了解决这个问题，React引入了key特性唯一标识符，帮助React库进行快速查找匹配元素以避免性能瓶颈。

组件生命周期

- 加载阶段：constructor -> componentWillMount -> render -> componentDidMount
- 卸载阶段：componentWillUnmount
- props更改：componentWillReceiveProps -> shouldComponentUpdate -> componentWillUpdate -> render -> componentDidUpdate
- state更改：shouldComponentUpdate -> componentWillUpdate -> render -> componentDidUpdate

组件属性校验

组件的类属性propTypes显式的在组件中声明可以使用哪些属性、哪些属性是必需的、属性可以接受的数据类型等，React的propTypes对象包含一组校验器，也可以自定义propTypes校验器（函数签名是：function(props, propName, componentName)），以获得以下两个好处：

1. 直观的查看代码中的属性约束；

2. 当错误使用或者使用错误时及时获得警告。

疑问：这是合适的做法吗？还有更优雅的做法吗？

可以通过定义组件的类属性defaultProps定义属性的默认值。

组件组合策略

根据组件是否包含state，可以将组件分为有状态组件（包含内部的state）和无状态组件（又叫做单纯组件，没有内部的state，只显示通过props接收到的数据）两种，可以按照下面4步来确定组件是否应该包含状态：

1. 标识出基于state进行渲染的每一个组件；
2. 找到一个共用的父级Owner组件；
3. Owner组件或者更高层级的组件应该拥有state；
4. 如果无法找到一个合适的Owner组件，在共用的父级位置上创建一个Owner组件。

使用表单

React提供了受控组件和非受控组件两种形式来处理表单。

- 包含value值（或selected已选属性）的表单组件称为受控组件，其值由组件内部属性控制。

- 非受控组件不为表单提供相关属性值，渲染后的值对应用户的输入。非受控组件是一种反模式，适用于长表单等不需要即时监管的表单。需要用defaultValue属性设置表单的初始值，用onSubmit来处理表单值。

refs

在React的运作方式中，渲染组件时，与你打交道的总是虚拟DOM。例如，假设你更改了一个组件的state，或将新的props发送给子元素，它们会被响应式地渲染到虚拟DOM中，然后React会在子级校正结束后更新真实DOM。

在操作真实DOM时，请三思而后行，因为在几乎所有情形中，你都可以在React模型中清晰明确地组织你的代码。在那些有必要操作真实DOM的情况下，React提供了一个refs后门。

数据传递

- Props是React中的一种从父组件向子组件传输数据的机制。子组件里面不能被修改。
- 组件内的State由组件自己管理，要尽量保持精简，当State被修改时，组件会触发响应式渲染，组件自身及其子组件都会被重新渲染，保持State和UI同步。

不变性

在React中大多数时候性能取决于比较和检查一个对象是否被修改过，这种操作代价高昂，性能压力很大。简单的解决办法是：不修改对象，而是直接替换这个对象，简单的比较对象的引用以提高性能。这就是不变性的基本概念。

对象和数组都通过引用方式传递，并且数组的非侵入方式和Object.assign都不会做深度复制—性能消耗很大。

React的“immutability-helper（react-addons-update替代品）”不变性助手提供了$push，$unshift，$splice，$set，$unset，$merge，$apply等指令性来进行数据修改来生成数据的浅表副本。

复杂的程序中更推荐使用Immutable.js所提供的不可变数据结构。

使用不可变数据类型的另一个好处是：可以保存旧的state对象引用，发生问题时回滚到旧的state对象。

最佳实践

1. 让应用程序的大部分组件都是无状态组件。数据总是沿着组件的层级从上到下这一个方向单向传递，下层组件调用容器组件通过props传递过来的回调函数修改数据（或state）。

2. 创建专门的有状态组件（容器组件），来专门负责和API通信，并将数据和回调以props的方式传递给下层组件。用于分离和UI无关的业务逻辑和与其对应的子组件。容器是纯粹的业务组件，当state没有发生变化时，不会重新进行渲染。

3. 应该总是使用setState方法来更新组件UI的状态，不能直接修改this.state，应该认为this.state具有不变性。

钻取应用程序的问题

- 单向数据流是React的核心理念之一，即以props的形式从父组件流向子组件。当父组件需要子组件返回数据时，就可以像传递props一样传递一个回调函数。

- React应用程序常常会拥有许多层级，顶层组件扮演了容器的角色，而许多纯粹的组件则更像是界面树上的叶子节点。state位于层级的最高处，回调函数做为props向下传递，有时会在重复并且易于出错的任务中向下传递许多层级——钻取应用程序。

- 问题是：当应用程序规模逐渐变大时，如何将数据数据以及回调函数带入到嵌套组件中，并能通过回调函数来操作这些组件的数据?

将数据和回调函数分离到单独的文件（类）中。

Flux应用程序架构

Flux是Facebook创建的一份针对Web应用程序的架构指引。Flux的要点是允许应用程序中存在单向数据流。他有三个基本部分组成： Action，Store和Dispatcher。

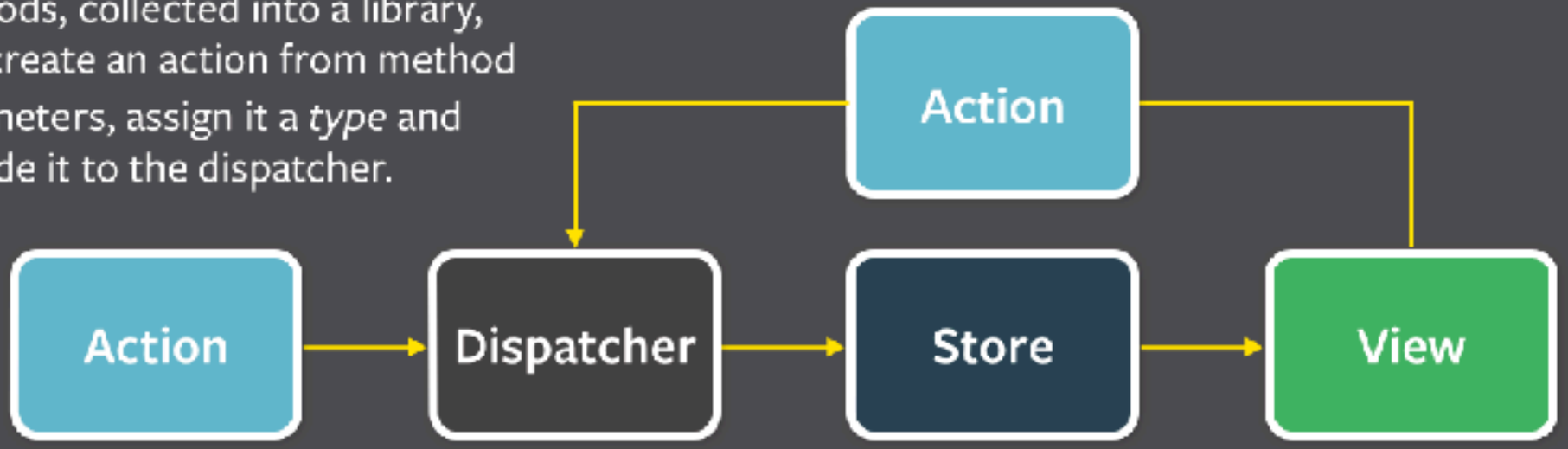
1. Store解决的问题是如何将数据带入到应用程序的每个组件。Store存放应用程序所有状态，包括数据和UI状态，在状态发生变化时分发事件。View订阅了所需的数据的Store，当数据发生变化时，他就会重新渲染自身。达到数据完全和组件分离，数据驱动View变化的理想状态。Store和Model相似，但Store只有getter，没有setter，只有store自身可以操作自己。

2. Action可以不严谨的定义为“应用程序中发生的事情”，每个Action都包含一个type和一个可选的payload。

3. Dispatcher负责协调Action传递到Store的过程，并确保以正确顺序执行Store的Action处理程序，在Flux工具包中使用waitFor协调Store的更新顺序。

在一个Flux应用程序中，Store拥有state，并在Dispatcher 注册了自己。每次分发Action时，所有Store都会被调用，并且可以决定是否关心该Action。如果有一个Store响应了Action并更改了它的内部state，发送一个事件通知View。

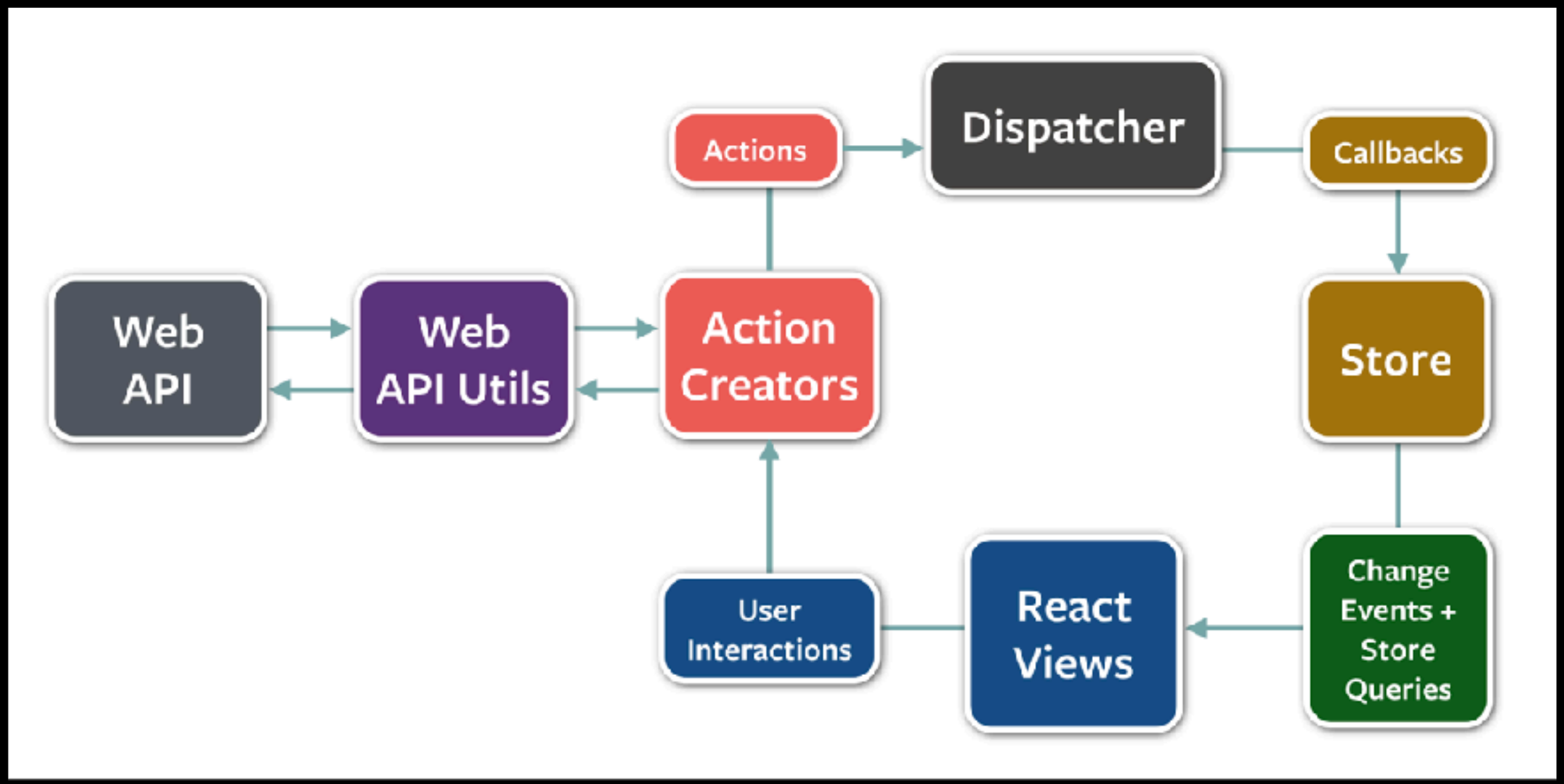
Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

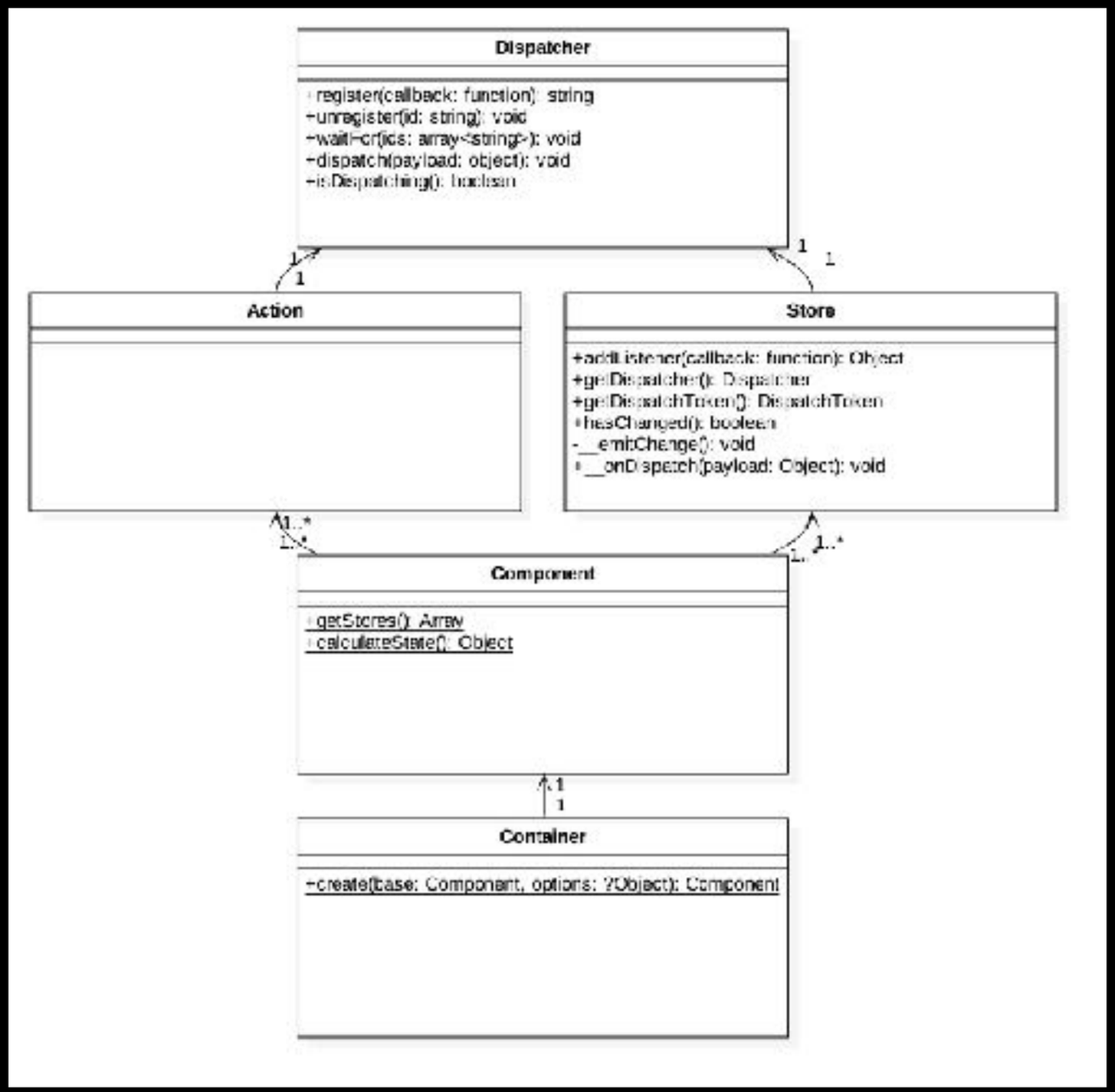


Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.


Actions
Dispatcher
Callbacks
Web API
Web API Utils
Action Creators
Store
User Interactions
React Views
Change Events + Store Queries


Dispatcher
+register(callback: function): string
+unregister(id: string): void
+waitFor(ids: array<string>): void
+dispatch(payload: object): void
+isDispatching(): boolean
1
1
1
1
Action
Store
+addListener(callback: function): Object
+getDispatcher(): Dispatcher
+getDispatchToken(): DispatchToken
#hasChanged(): boolean
-__emitChange(): void
#__onDispatch(payload: Object): void
1..*
1..*
1..*
1..*
Component
getStores(): Array
calculateState(): Object
1
1
Container
+create(base: Component, options: ?Object): Component

示例代码

示例托管在GitHub上，简单示例的react-todo。可参考标签和提交日志阅读。里面共存在三个分支，相关差别是：

1. master分支是使用React完成的第一个版本，视图、数据、行为混在一起；
2. refactor分支是在master分支基础上，将state拆分到store中，将行为拆分到actions中；
3. flux分支是在master分支基础上进行的按照flux形式的重构。

路由

Web应用程序中，看待URL的最佳方式是：应用程序当前状态的一种表示。状态变化后，即时反应到URL上；URL变化后，状态相应更新。这是一种双向映射。

React Router库，通过将组件与路由关联起来，使UI和URL保持同步。用户操作以<Link>更改URL；代码操作更改历史记录URL 促发路由跳转。

React Router中有两种传递props的方法：

1. 通过在路由配置对象上指定props
2. 使用“黑科技”在子组件上克隆注入props（组件某些场景下可能不会重新加载）。

提醒：在服务器上配置rewrite规则。

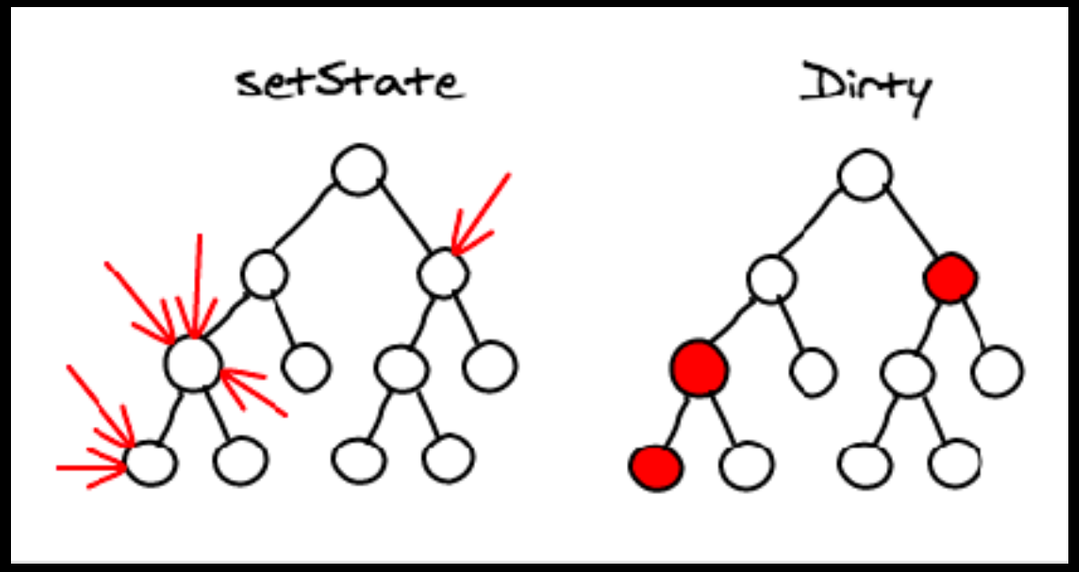
性能优化

- 避免过早优化，对应用程序进行分析检查他是否需要进行性能的调整，以及在哪里调整。
- React Perf是一个React插件，会对应用程序的整体性能进行概要性分析。
- 仅可在开发模式下使用，不应该在构建生产环境应用时包含这个插件。该插件虽然提供了一些很有价值的洞察分析，但他永远不可能检测到你的应用中所有可以优化的点。请配合使用浏览器的开发工具对应用进行检测和调试。

子级校正

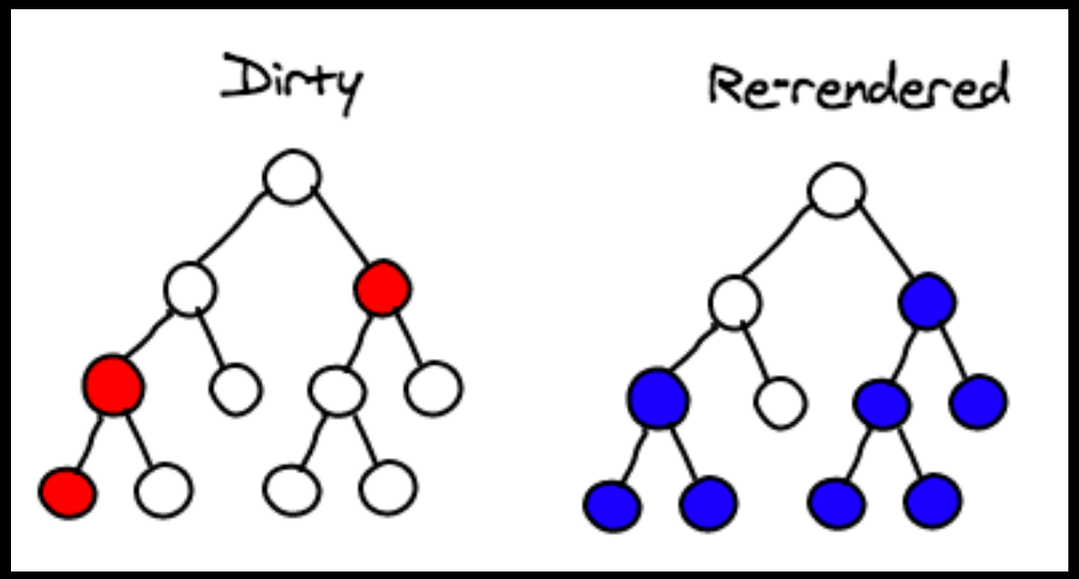
- React组件状态改变时，会触发组件的重绘过程。React首先构建一个新的虚拟DOM来呈现应用UI的状态，然后检测和当前的虚拟DOM之间的差异，从而计算出哪些DOM元素需要进行更新、添加或删除。这个过程被称为“子级校正（reconciliation）”。

- 在React中任何时候调用组件的setState方法，React都不会立即对其更新，而是将其标记为“脏”状态，这也意味着组件的状态变更不会立刻生效，React使用了时间轮询对变更内容进行批量绘制。通过对子级校正过程进行批处理，在每个事件轮训中，DOM节点只会进行一次更新，这正是实现一个高性能应用的关键所在。



子树渲染

- 当事件轮询结束后，React会将“脏”状态组件及其子组件进行重绘，调用所有后代节点的render方法，即使它们并没有发生变化。整个过程都发生在内存中的虚拟DOM上。



shouldComponentUpdate

- React提供了shouldComponentUpdate方法对子树渲染的过程进行优化，可以用来阻止子树的重绘。在子组件重绘之前，React都会调用它的shouldComponentUpdate方法。默认情况下始终返回true，可以自定义来返回false以跳过该组件及其子组件的重绘过程。

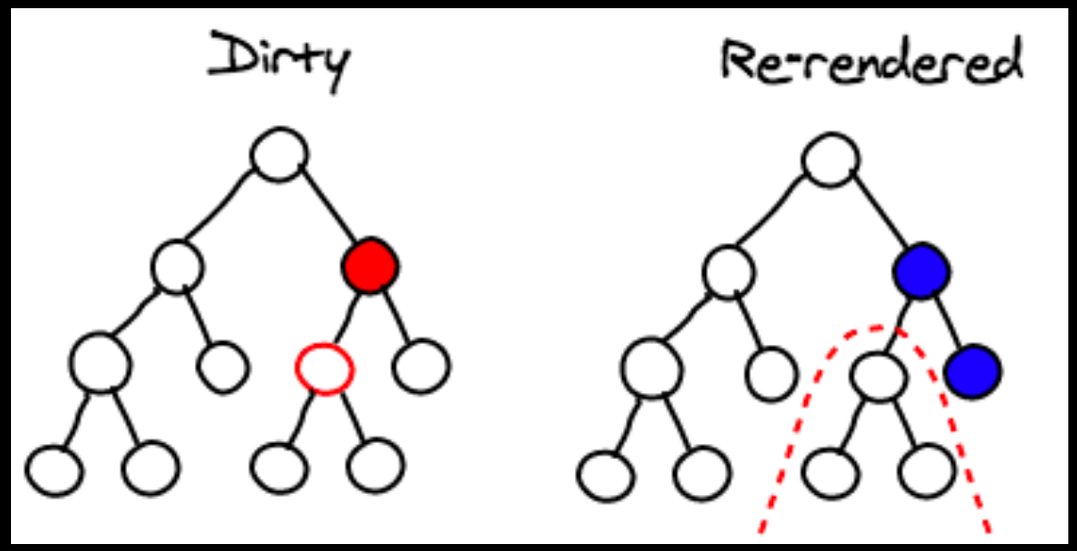
- shouldComponent方法接收nextProps和nextState作为参数，返要true或false来表示该组件是否需要进行重绘。

- 在实现该方法时所有的检测、必须非常快，否则通过这个方法来改善应用性能就没有什么意义了。比较两个简单的值会非常快。但如果比较两个对象中层级很深的属性开销就很大，这种方法就没有什么意义了。这就是使用不可变值的意义：他会让整个对象的跟踪和开销很小小、速度极快而且高度可靠。

- React提供了一个名为shallowCompare的插件来配合使用，可以浅度比较组件的props和state属性，使用前提是：

    1. 需要浅度比较的组件是纯粹的，相同的props和state总是返回相同的渲染；

    2. 使用了不可变值（Immutable.js）或不可变助手。



同构应用

传统的单页应用基本上就是一个空白的HTML，在JavaScript下载完成并运行之前，用户会看到一个白屏闪过，然后才是页面的内容。

同构应用也称为通用JavaScript的应用，指的是在客户端和服务器端之间完整（或部分）的共享代码的应用。通过在服务器端运行应用的JavaScript代码，页面可以在发送到浏览器之前预先填充内容，用户可以在浏览器的JavaScript运行之前就先看到内容。当本地的JavaScript运行时，它会接受后续的交互及导航操作，让用户在单页应用中得到流畅的交互体验。同构应用的好处：

1. 应用加载和渲染的速度更快，体验更好；
2. 获得渐进式增强能力、更好的可访问性；
3. 搜索引擎友好。

React和React-DOM包通过ReactDOMServer.renderToString方法内置支持在服务器端渲染组件的能力，该方法会对所需组件进行渲染，并生成带有注释的HTML然后发送到浏览器。在浏览器中，React会识别这些注释，并只进行事件处理程序的加载，从而使得应用在初次加载时获得极佳性能。

测试React组件

应用变得越来越复杂，功能持续不断增加，需要保证新功能在实现时没有为已有功能引入新Bug。自动化测试提供了一个活生生的文档来描述预期行为，能够让我们在开发过程中更有信心，在第一时间就能了解出现的问题。

Jest是React推荐的测试框架，它基于流行的Jasmine框架并加入了下列特性：

1. 在虚拟DOM上运行测试；
2. 内置支持JSX。

同时，React提供了react-addons-test-utils测试工具，该工具提供的renderIntoDocument方法会将组件渲染到一个脱离文档的独立DOM节点中。并提供了子节点遍历、事件模拟和浅渲染等特性。

复杂交互

- React中的动画可以使用react-transition-group，文档。
- React中的拖放可以使用react-dnd，官网。

乐观更新

乐观更新是指：在没有等到服务器返回响应是否真正操作成功之前，就已经更新了UI。

乐观更新对用户体验很重要：当用户和一个在线App进行交互时，任何等待都是漫长的，也不关心是否任务需要被保存到一个远程数据库中。所有操作都需要实时完成。但是如果服务器保存数据失败，则需要做一些新尝试保存、回滚UI上的更新、提示用户或执行其他操作。

谢谢