# 18 React Components for Web Developers 2020 - ColorlibОснови на

React обикновено се смята за изгледен слой в приложение. Точно както jQuery манипулира елементи на потребителския интерфейс, React променят какво потребителят вижда. В общия случай имаме някаква логика на приложението, която генерира данни. Искаме да предоставим тези данни на потребителския интерфейс, така че го предаваме на React Component, който се справя със задачата да въведе HTML страницата.

React е разделен на два основи:

* React Component API: Това са частите на страницата, които всъщност са изведени от React DOM.
* React DOM: Това е API, което се използва за реално изобразяване в страница.

В рамките на React компонента имаме следните области:

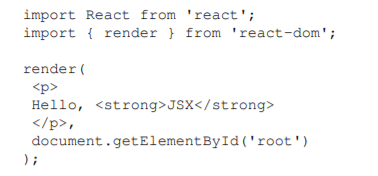
* Данни: Идват от някъде и се визуализират от компонента.
* Жизнен цикъл: Състои от методи или „куки“ (Hooks), които реагират на фазиъе за влизане и излизане на компонента от процеса на рендиране през времето. Например една фаза от жизнения цикъл е когато компонент е на път да бъде изобразен.
* Събития: Това е кодът, който пишем за отговор на потребителските взаимодействия.
* JSX: Това е синтаксисът на React компонентите, използвани за описване на UI структури.

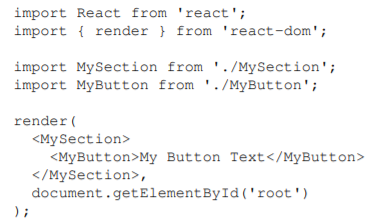
Функцията React.memo () е съвременният еквивалент на класа PureComponent. Запомнените компоненти избягват повторно изобразяване, ако данните за компонентите не са се променили. Това автоматично се справя с проверка дали данните на компонентите са се променили или не и дали компонентът трябва да се визуализира отново или не. Предизвикателството с този подход е, че сега е обичайно за големите приложения на React да имат много функционални компоненти. Преди React.memo () нямаше начин да се запомнят компоненти, за да се избегне повторно изобразяване. Сега може да предадете вашите функционални компоненти на React.memo() и те ще се държат като PureComponent.

React е сравнително малка библиотека, използван за изграждане на потребителски интерфейси. Има декларативен характер на компонентите чрез JSX. (код, който може да се изобразява отново и отново).

## Представяне(Rendering) чрез JSX

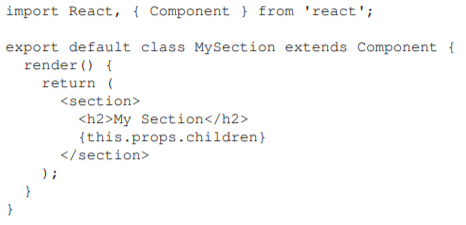
JSX е XML / HTML синтаксисът за маркиране, който е вграден във вашия JavaScript код и използван за деклариране на вашите React компоненти. В най-ниското ниво, ще използвате HTML маркиране, за да опишете частите от потребителския си интерфейс.

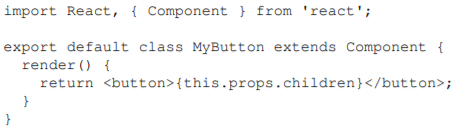
Функцията render () приема JSX като първи аргумент и го изобразява на подадения DOM като втори аргумент. В този пример показва абзац с някакъв удебелен текст вътре. Функцията render казва на React да вземе вашият JSX код и да го трансформира в JavaScript, който актуализира потребителския интерфейс по възможно най-ефективния начин. Ето как React позволява да декларирате структурата на вашия потребителски интерфейс, без да се налага да мислите за изпълнение на стъпките за актуализиране на елементи на екрана. React поддържа стандартните HTML тагове, които бихте намерили на всяка HTML страница. За разлика от статичния HTML, React има уникални конвенции, които трябва да се спазват при използване. Когато изобразявате HTML тагове в JSX, трябва да използвате малки букви за името на маркера. Имената на маркерите са чувствителни към малки и малки букви, а не-HTML елементите се пишат с главни букви.

Използването на JSX е полезно за описване на UI структури, които имат родител-дете взаимоотношения. Например, <li> таг е полезен само като дете на <ul> таг или <ol>.

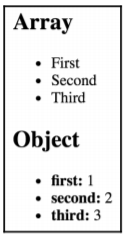
Примера включва 2 React компонента: MySection и MyButton. Сега, ако погледнете JSX маркировката, ще забележите, че <MyButton> е дете на <MySection>.

Също така ще забележите, че компонентът MyButton приема текст като свое дъщерно устройство, вместо други JSX елементи.

MySection компонентът показва стандартен <section> HTML елемент, заглавие и след това {this.props.children}. Това е последното парче, което позволява на компонентите да имат достъп до вложени елементи или текст и да ги изобразява.

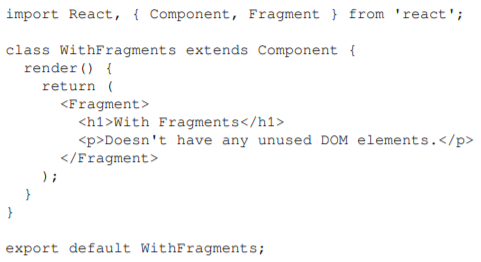
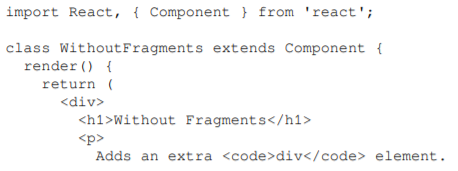
Нека разгледаме и компонента MyButton. Този компонент използва точно същия модел като MySection; вземете

  
{this.props.children} стойност и я обградете с маркиране. React се справя с детайлите.  
В този пример текстът на бутона е дъщерно устройство на MyButton, което от своя страна е дете на MySection.

Първата колекция е масив, наречен array, запълнен с низови стойности. Преминавайки надолу към маркировката JSX, можете да видите извикването на array.map(), което връща нов масив. Mapping функцията всъщност връща JSX елемент (<li>), което означава, че всеки елемент в масивът вече е представен в маркировката.

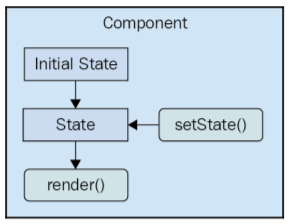
Ето така изглежда примерът.

React 16 представя концепцията за JSX фрагменти. Фрагментите са начин да се групират заедно парчета маркировка, без да се налага да добавяте ненужна структура към вашата страница. Например, често срещан подход е да се върне съдържание на React компонент, увито в <div> елемент. Този елемент няма реална цел и добавя безпорядък към DOM.

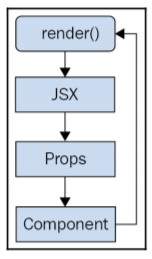
Вместо да увива съдържанието на компонента в <div>, се използва елементът <Fragment>. Това е специален тип елемент, който показва, че трябва да бъдат изобразени само неговите деца.

Стенографичен начин за изразяване на фрагменти в JSX: <>My Content</>

## Component Properties, State, and Context

Компонентите на React декларират структурата на UI елементите, използвайки JSX. Въпреки това, компонентите се нуждаят от данни, за да бъдат полезни. Например вашият компонент JSX може да декларира <ul> който съпоставя колекция . Откъде идва тази колекция?  
State е динамичната част на React компонент. Можете да декларирате първоначалното състояние на компонент, който се променя с течение на времето. Представете си, че изобразявате компонент, при който част от състоянието му се инициализира в празен масив. По-късно този масив се попълва с данни, използвайки setState (). Това се нарича промяна в състоянието и всеки път, когато кажете на React компонент да промени състоянието си, компонент автоматично ще се рендерира, извиквайки render(). Състоянието на компонент е нещо, което или самият компонент може да зададе, или други парчета код, извън компонента.

Свойствата (Properties) се използват за предаване на данни във вашите React компоненти. Вместо да извика метод с ново състояние като аргумент, свойствата се предават само когато компонентът е предоставен (rendered). Това означава, че предавате стойности на свойствата на JSX елементи. В контекста на JSX свойствата се наричат атрибути, вероятно защото така се наричат на XML език.

Свойствата (Properties) са различни от състоянието (state), защото не се променят след първоначалното изобразяване на съставна част. Ако стойността на свойството се е променила и искате да изобразите отново компонента, тогава трябва да изобразим отново JSX, който е бил използван за визуализирането му на първо място. Това изглежда много по-различно от компонент със състояние. Истинската разлика е, че със свойства, често родителският компонент, решава кога да изобрази JSX. Самият компонентът всъщност не знае как да се изобрази отново.