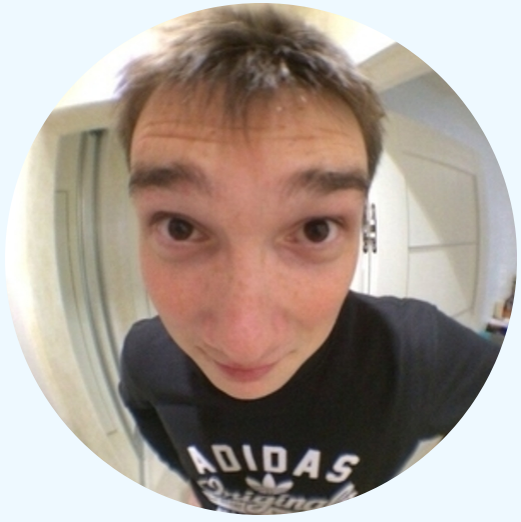
нетология

КОМПОЗИЦИЯ КОМПОНЕНТОВ

Александр Сивцов



АЛЕКСАНДР СИВЦОВ

Frontend-разработчик в Aliexpress Russia

Александр Сивцов

ПЛАН ЗАНЯТИЯ

1. Введение в композицию компонентов
2. Подходы к композиции компонентов
3. Передача компонентов в атрибуты
4. Вложенные компоненты - Children Components
5. Композиция списков
6. Функция как дочерний компонент
7. Spread оператор - оператор разворота

ВСПЛЫВАЮЩИЙ БЛОК

Допустим у нас на проекте мы хотим показывать горячие предложения всплывающим блоком:

```
function TodayOfferModal() {
  return (
  <div className="modal">
    <div className="modal-body">
      <h2>Только сегодня и только сейчас!</h2>
      <a href="/today/">Узнать подробнее</a>
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

Компонент может выглядеть примерно так.

ЧТО ЕСЛИ ТАКИХ ПРЕДЛОЖЕНИЙ БУДЕТ МНОГО?

Чувствуете проблему, которая будет нарастать как снежный ком с каждой новой акцией?

```
function DiscountOfferModal() {
  return (
  <div className="modal">
    <div className="modal-body">
      <h2>Самые большие скидки!</h2>
      <a href="/discount/">Узнать подробнее</a>
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

ЗАДЕЙСТВУЕМ СИЛУ `props`!

Вынесем заголовок и ссылку в атрибуты нашего компонента:

```
function OfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">
      <h2>{props.title}</h2>
      <a href="{props.link}">Узнать подробнее</a>
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

ПРОБЛЕМА РЕШЕНА!

Было:

```
<TodayOfferModal />
<DiscountOfferModal />
```

Стало:

```
<OfferModal
  title="Только сегодня и только сейчас!"
  link="/today/" />
<OfferModal
  title="Самые большие скидки!"
  link="/discount/" />
```

НО РАЗВИТИЕ ПРОЕКТА НЕ ОСТАНОВИТЬ

Со временем появится требование создать всплывающий блок с таким кодом:

```
<div className="modal">
  <div className="modal-body">
    <form action="/abonement/">
      <h2>Выбери свой абонемент!</h2>
      <select name="type">
        <option value="3">На 3 месяца</option>
        <option value="6">На 6 месяцев</option>
      </select>
      <button type="submit">Выбрать</button>
    </form>
  </div>
  <button className="modal-close">закрыть</button>
</div>
```

КТО ВИНОВАТ И ЧТО ДЕЛАТЬ?

Содержимое нашего всплывающего блока перестало подчиняться изначальному правилу «заголовок + ссылка» и стало свободным: просто какая-то разметка.

В каких направлениях мы можем двигаться:

- Создать отдельный компонент именно под эту акцию.
- Передавать в атрибуты компонент, вместо заголовка и ссылки.
- Передавать вообще HTML разметку как строку.

ОТДЕЛЬНЫЙ КОМПОНЕНТ

Выходит что мы лишь немного притормозили рост компонентов:

```
function AbonOfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">
      <form action="/abonement/">
        {/* … */}
      </form>
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

КОМПОНЕНТ В АТРИБУТЫ

На первый взгляд всё вроде элегантно:

```
function OfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">{props.body}</div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

А вот на второй — отвратительно. Как вам JSX внутри JSX:

```
<OfferModal body={<LinkOfferModalBody />} />
<OfferModal body={<AbonOfferModalBody />} />
```

ПЕРЕДАЧА HTML СТРОКОЙ

Вставка HTML-кода строкой возможна, но посмотрите как это выглядит:

```
<div dangerouslySetInnerHTML={props.body} />
```

`dangerouslySetInnerHTML` (*dangerously* — опасно) — разработчики React встроили напоминание о том, что нужно стараться этого избегать в саму библиотеку.

Поэтому этот вариант даже не будем рассматривать.

ВЛОЖЕННЫЕ ЭЛЕМЕНТЫ

ВЛОЖИМ ТЕЛО В КОМПОНЕНТ

До этого мы использовали наши компоненты как одиночный самодостаточный элемент. Но ведь в HTML у элементов есть дочерние элементы. А что же JSX? Вложим тело в компонент:

```
<OfferModal>
  <h2>{props.title}</h2>
  <a href="{props.link}">Узнать подробнее</a>
</OfferModal>
```

```
<OfferModal>
  <form action="/abonement/">
    {/* … */}
  </form>
</OfferModal>
```

ПОХОЖЕ ЧТО ЭТО ВОЗМОЖНО

Ошибок такой код с нашим компонентом не вызывает:

```
function OfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">{props.body}</div>
    <button className="modal-close">закрыть</button>
  </div>);
}
```

Но и чуда не случается. Получаем такой HTML-код:

```
<div class="modal">
  <div class="modal-body"></div>
  <button class="modal-close">закрыть</button>
</div>
```

ПОЛУЧАЕМ ВЛОЖЕННЫЕ ЭЛЕМЕНТЫ

Так как у нас функциональный компонент, и до этого мы имели дело только с аргументом `props`, начнем поиски с него:

```
function OfferModal(props) {
  console.log(props);
  return (
  <div className="modal">
    <div className="modal-body">{props.body}</div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

СРАЗУ В ЯБЛОЧКО!

Видим в консоле что в `props` есть свойство `children` и оно *массив*:

```
Object
└ children: Array(2)
```

В массиве два элемента, каждый из которых объект:

```
Array (2)
├ 0 { type: "h2", …}
└ 1 { type: "a", …}
```

Это и есть наши вложенные элементы. Как их вывести?

ВЫВОДИМ ВЛОЖЕННЫЕ ЭЛЕМЕНТЫ

Мы уже не раз выводили в JSX массив элементов. Помните занятия по спискам?

Мы можем просто вставить массив в нужное место кода нашего компонента:

```
function OfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">
      {props.children}
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

СВОБОДА ВЫБОРА

Стоит отметить, что мы вообще не обязаны куда-то выводить `props.children`. Мы так пробовали, и это не привело к ошибкам. Мы решаем — может ли компонент иметь вложенные элементы, и где они будут выводиться.

Но мы так же можем решить вывести только первый:

```
function OfferModal(props) {
  return (
  <div className="modal">
    <div className="modal-body">
      {props.children[0]}
    </div>
    <button className="modal-close">закрыть</button>
  </div>
  );
}
```

ИТОГИ ПО ВЛОЖЕННЫМ ЭЛЕМЕНТАМ

Возможность вкладывать элементы в компонент открывает перед нами новые возможности по созданию более гибких и универсальных компонентов. Подведем итоги:

- Созданный нами компонент можно использовать как одиночный `<OfferModal />`, так и двойной `<OfferModal>...</OfferModal>`.
- Если в наш компонент вложить элементы, то они передаются в компонент через аргумент функции `props` в его свойстве `children`.
- Свойство `children` — *массив*, каждый элемент которого является React-элементом.
- Мы сами определяем где и как выводятся вложенные элементы из `children`. Например, мы можем их вообще не выводить.
- В компонентах на основе классов вложенные элементы доступны через `this.props.children`. И работают так же.

КОМПОЗИЦИЯ КОМПОНЕНТОВ

КОМПОЗИЦИЯ КОМПОНЕНТОВ

Композиция компонентов — это процесс комбинирования двух или более компонентов с целью создания новой более сложного компонента.

Композиция React компонентов схожа с композицией функций, так как содержит те же принципы.

СООБЩЕНИЕ

Допустим у нас есть компонент для отображения всплывающих сообщений:

```
function Message(props) {
  return (
  <div className={`message message-${props.type}`}>
    {props.text}
  </div>
  );
}
```

Как поступить, если нам довольно часто нужно выводить сообщения об ошибке:

```
<Message type="error" text="Пароли не совпадают" />
```

БОЛЕЕ СПЕЦИАЛИЗИРОВАННЫЙ КОМПОНЕНТ

Всегда можно сделать так:

```
function ErrorMessage(props) {
  return (
  <div className="message message-error">
    {props.text}
  </div>
  );
}
```

Но так мы нарушаем принцип DRY. А так нет:

```
function ErrorMessage(props) {
  return <Message type="error" text={props.text} />;
}
```

ЭТО ИССКУСТВО

Создание компонентов React — это исскуство. И сама библиотека очень в этом помогает.

Представим что по какой-то причине наш компонент `Message` принимает множество аргументов:

```
<Message
  type="error"
  icon="/path/to/error.png"
  position="topright"
  timeout="60"
  priority="100"
  text="Пароли не совпадают" />
```

Как нам быть с `ErrorMessage`, если мы хотим в нём зафиксировать только один-два параметра?

ПЛОХОЕ РЕШЕНИЕ

Мы можем просто «пробросить» все известные нам атрибуты в `Message`:

```
function ErrorMessage(props) {
  return (
  <Message
    type="error" icon="/path/to/error.png"
    position={props.position} timeout={props.timeout}
    priority={props.priority} text={props.text} />
  );
}
```

Решение принесет первые проблемы сразу как только в компоненте `Message` появятся новые атрибуты, и кто-то попробует их использовать в `ErrorMessage`.

ОПЕРТОР РАЗВОРОТА В JSX

Для начала давайте вспомним про оператор разворота (*spread*) `...`.

Его можно использовать в JSX для передачи объекта целиком в качестве атрибутов. Допустим у нас есть объект:

```
const message = {
  type: 'info',
  text: 'Информационное сообщение'
};
```

С помощью оператора разворота мы можем записать вот так:

```
<Message {...message} />
```

Все свойства объекта `message` передадутся в `props` нашего компонента или в свойства элемента.

ПОРЯДОК ИМЕЕТ ЗНАЧЕНИЕ

Если атрибут с именем, которое есть в объекте идет перед оператором разворота, его значение будет переписано. Тут получим сообщение типа `info`:

```
<Message type="error" {...message} />
```

Если атрибут идет после оператора разворота, будет использовано его значение. Тут получим сообщение типа `error`:

```
<Message {...message} type="error" />
```

ПРИМЕНИМ ОПЕРАТОР РАЗВОРАТА

Пробросим все атрибуты и переопределим те что нам нужны:

```
function ErrorMessage(props) {
  return (
  <Message
    {...props}
    type="error"
    icon="/path/to/error.png" />
  );
}
```

ЧИСТОТА JSX

Рекомендуется использовать `Object.assign` для создания свойст для проброски в компонент, вместо переопределения их в JSX:

```
function ErrorMessage(props) {
  const base = {
    type: 'error',
    icon: '/path/to/error.png'
  };
  const newProps = Object.assign({}, props, base);
  // либо const newProps = {...props, ...base};
  return <Message  {...newProps} />;
}
```

Если вам вдруг пришла в голову идея переопределить свойства в `props`, напомню что в React это делать нельзя.

КОМПОНЕНТ БЕЗ КОМПОЗИЦИИ

Нам нужно создать компонент профиля пользователя. Как бы мы реализовали это без композиции компонентов?

Примерно так:

```
const Avatar = props => (
<div>
  <img src={`https://photo.yoursite.com/${props.username}`} />
  <a href={`https://yoursite.com/${props.username}`}>
    {props.username}
  </a>
</div>
)
```

Все вроде хорошо и работает. В чем тогда смысл перетруждать себя?

На лекции с HTTP мы рассмотрим, как для этой цели можно использовать environment.

КОМПОНЕНТЫ

Напоминаем, что компоненты (функциональные) - это просто функции и не важно, как вы их записываете в исходном файле:

```
const Avatar = props => (...);
// или
function Avatar(props) { return (...); }
```

ПРОФИЛЬ ПОЛЬЗОВАТЕЛЯ

Допустим, вы хотите создать профиль пользователя, тогда компонент будет выглядить так:

```
const Profile = props => (
<div>
  <img src={`https://photo.yoursite.com/${props.username}`} />
  <a href={`https://yoursite.com/${props.username}`}>{props.username}</a>
  <div>{props.age}</div>
  <div>{props.email}</div>
</div>
)

// в App.js
return (
  <Profile username='coollogin' age={24} email='coollogin@yoursite.com' /
)
```

Вопрос: какие недостатки у данного подхода?

ГЛАВНЫЙ НЕДОСТАТОК ПОДХОДА

Если мы захотим изменитьссылку на сайт, то нам придется пройтись по всем частям программы где используется изображение профиля и ссылка на него. В нашем случае таких частей две `Avatar` и `Profile`.

А что если их было бы намного больше?

А если бы тесты для них были бы разработаны и зависили от ссылок в явном виде?

ВВЕДЁМ ДОПОЛНИТЕЛЬНЫЕ КОМПОНЕНТЫ

```
const ProfilePic = props =>
  <img src={`https://photo.yoursite.com/${props.username}`} />
const ProfileLink = props =>
  <a href={`https://yoursite.com/${props.username}`}>{props.username}</a>

const Avatar = props => (
<div>
  <ProfilePic username={props.username} />
  <ProfileLink username={props.username} />
</div>
)

const Profile = props => (
<div>
  <ProfilePic username={props.username} />
  <ProfileLink username={props.username} />
  <div>{props.age}</div>
  <div>{props.email}</div>
</div>
)
```

РЕЗУЛЬТАТ ИЗМЕНЕНИЙ

Наш компонент `Avatar` состоит из компонентов `ProfilePic` и `ProfileLink`, как и компонент `Profile`.

Данный подход позволил нам существенно улучшить код и избежать в будущем проблем с изменением компонентов.

ПОДХОДЫ К КОМПОЗИЦИИ КОМПОНЕНТОВ

ВЫДЕЛЕНИЕ КОМПОНЕНТОВ

Рассмотрим вторую задачу: у нас есть боковая панель сайта (`sidebar`) на которой у нас расположены виджеты.

Наша задача выделить в данном коде компоненты написать их реализацию.

```
<aside id="sidebar">
  <div class="widget">
    <div>Поиск</div>
    <form method="GET" action="/search/" class="search">
      <input type="text" name="query" placeholder="Поиск..." value="">
    </form>
  </div>
  <div class="widget">
    <div>Теги</div>
    <div class="tag-block">
      <a href="/articles/tag/js/" class="tag-link clear">JavaScript</a>
      <a href="/articles/tag/nodejs/" class="tag-link clear">Node.JS</a>
    </div>
  </div>
  ... продолжение на следующем слайде ...
</aside>
```

```
<aside id="sidebar">
  ... начало на предыдущем слайде ...
  <div class="widget">
    <div>Социальные сети</div>
    <div class="social-block">
      <ul>
      <li><a href="https://vk.com/username" class="clear social-link">VK</a></li>
      <li><a href="mailto:username@gmail.com" class="clear social-link">Email</a></li>
      <li><a href="https://github.com/username" class="clear social-link">Github</a></li>
      </ul>
    </div>
  </div>
</aside>
```

ПОДХОДЫ ДЛЯ ВЫДЕЛЕНИЯ КОМПОНЕНТОВ

Существуют три подхода для выделения компонентов:

- От большего к меньшему;
- От меньшего к большему;
- Комбинирование.

ОТ БОЛЬШЕГО К МЕНЬШЕМУ

Используется когда вам надо выделить компоненты в достаточно объемном коде, где наблюдается существенная трудность для выделения всего и сразу.

ОТ МЕНЬШЕГО К БОЛЬШЕМУ

Используется когда область вашей задачи небольшая и вы вполне можите выделить маленькие части вашего интерфеса.

КОМБИНИРОВАНИЕ

Используется также когда надо выделить компоненты в достаточно большом коде, и представляет собой чередование подходов.

ПЕРЕДАЧА КОМПОНЕНТОВ В АТРИБУТЫ

СОЗДАДИМ КОМПОНЕНТ ДЛЯ КОНТЕНТА

В нашем случае мы будем использовать подход "от большего к меньшему".

Первым делом создадим компонент в котором будут содержаться контент для боковой панели:

```
const Sidebar = props =>
  <aside id={props.id}>{props.content}</aside>

// в родительском компоненте
const sidebarContent = (
  // ... внутри всё, что было в aside
  // ... все три виджета
)

return (
  <Sidebar id='sidebar' content={sidebarContent}/>
)
```

DOM В КАЧЕСТВЕ АТРИБУТА

В данном случае мы обернули нашу верстку в компонент `Sidebar` в котором определили два атрибута: `id` и `content`.

`content` содержит в себе DOM элементы являющиеся потомками для нашей боковой панели. Для этого мы отпределил переменную `sidebarContent` в которую поместили DOM.

Таким образом мы передали наш DOM в качестве атрибута. Но данный подход является неинтуитивным и мало чем схож с обычной верстской HTML, хотя изначально JSX таким себя позиционирует.

ВЛОЖЕННЫЕ КОМПОНЕНТЫ - CHILDREN COMPONENTS

ИСПОЛЬЗУЕМ `props.children`

Использовать атрибут `props.children` и наш компонент станет больше похожим на классическую HTML разметку.

```
1  const Sidebar = props =>
2    <aside id={props.id}>{props.children}</aside>
3
4  // в родительском компоненте
5  return (
6    <Sidebar id='sidebar'>
7      // ...
8    </Sidebar>
9  )
```

НЕЯВНОЕ ПОПАДАНИЕ В АТРИБУТ

Дочерные компоненты, которые располагаются между открывающим и закрывающим тегом нашего компонента *неявно* попадают в атрибут `props.children`, который может быть в последующим вызван.

А МОЖНО БЕЗ `props.children`?

А что будет если не указать в нашем компоненте `props.children`, а внутри тега передать ему дочерние компоненты?

```
const Sidebar = props => <aside id={props.id}></aside>

//...
```

БЕЗ `props.children` НЕ РАБОТАЕТ

Если мы не указываем `props.children`, то ничего не выведится на экран.

А ЕСЛИ НИЧЕГО НЕ ПЕРЕДАВАТЬ?

Что если указать в компоненте `props.children`, но дочерние компоненты ему не передать?

```
1  const Sidebar = props => {
2    console.log(props.children);
3
4    return <aside id={props.id}>{props.children}</aside>
5  }
6
7  // в родительском компоненте
8  return (<Sidebar id='sidebar'></Sidebar>)
```

ТАК ТОЖЕ НЕ СРАБОТАЕТ

Также ничего не выведится, так как в консоли мы получим значение `props.children` `undefined`, что попросту не отображается React в DOM элемент (вспомним также: `null`, `false`, `true`, пустые значения: `[]`, `{}` и т.д).

А ЕСЛИ ЯВНО УКАЗАТЬ АТРИБУТ `children`

```
1  const Sidebar = props =>
2    <aside id={props.id}>{props.children}</aside>
3
4  // В родительском компоненте
5  return (
6  <Sidebar id='sidebar'
7    children={<b>Я боковая панель!</b>}>
8    <b>А я дочерний компонент!</b>
9  </Sidebar>
10 )
```

Что выведется на экране в таком случае?

`children` ПЕРЕЗАПИШЕТСЯ

В итоге на экране мы получим следующее:

```
А я дочерний компонент!
```

Соотвественно сделаем вывод, что наш значение нашего атрибута `children` перезапишется дочерними компонентами между тегами родительского компонента.

ВЫВОДЫ

- JSX позволяет интуитивно задавать дочерние элементы.
- Если у нас не используется `props.children`, то мы можем задать наш компонент с помощью одиночного тега.
- Если у нас был задан явно атрибут `children`, то он перезапишется тем, что между тегами родительского компонента, а если между этими тегами ничего не заданно, то будет взято значение из нашего атрибута.

ПРИНЦИП ЕДИННОЙ ОТВЕТСВЕННОСТИ

Одним из ключевых принципов композиции является то, что мы должны выделить пересечение частей в нашем коде, чтобы в итоге выделить это значение в новый компонент. Данный подход широко используется в программировании и носит название *Принципа единной ответсвенности*.

ВЫДЕЛИМ КОМПОНЕНТ `Widget`

Для нашего примера можно выделить компонент `Widget` так как он часто повторяется, паттерн его поведения очевиден и прост.

```
const Widget = props => (
  <div className="widget">
    <div>{props.title}</div>
    {props.children}
  </div>
)
```

АТРИБУТЫ КОМПОНЕНТА `Widget`

Данный компонент принимает в качества атрибутов только название виджета `title`, а также у него определен неявный атрибут `props.children` для контента.

ОБНОВИМ КОМПОНЕНТ `Sidebar`

```
// в родительском компоненте
return (
<Sidebar id='sidebar'>
  <Widget title='Поиск'>
    // ...
  </ Widget>
  <Widget title='Теги'>
    // ...
  </ Widget>
  <Widget title='Социальные сети'>
    // ...
  </ Widget>
</Sidebar>
)
```

КОМПОЗИЦИЯ СПИСКОВ

КОМПОНЕНТЫ СПИСКА

В практике создания интерфейсов часто используются списки и нам бы хотелось создать компонент для работы с ними. Определим его компоненты:

```
const List = props =>
  <ul className={props.className} style={props.style}>
  {props.children}
  </ul>;
const ListItem = props =>
  <li className={props.className} style={props.style}>
  {props.children}
  </li>;
// в родительском компоненте
return (
<List className='list'>
  <ListItem>Написать отчет</ListItem>
  <ListItem>Купить молоко</ListItem>
  <ListItem>Забрать машину из ремонта</ListItem>
</List>
);
```

ВСЁ ХОРОШО...ВРОДЕ БЫ

Все хорошо, список отображается, но у нас тут также присутсвуют недостатки.

Какие недостатки вы тут видите?

НЕ ОПРЕДЕЛЕНА СВЯЗЬ С ДОЧЕРНИМИ КОМПОНЕНТАМИ

Самый очевидный недостаток состоит в том, что не определена связь между родительским компонентом и его детьми.

Что если родителю нужно передать некоторую информацию детям?

Для решения этой задачи мы можем использовать функцию как дочерний компонент.

ФУНКЦИЯ КАК ДОЧЕРНИЙ КОМПОНЕНТ

ФУНКЦИЯ В КАЧЕСТВЕ ДОЧЕРНЕГО ЭЛЕМЕНТА

Мы можем передавать в качестве дочернего элемента функцию, которую в последствии можем вызывать в нашем родительском компоненте, причем передавая ей нужные данные.

УСЛОВИЯ РАБОТЫ СПИСКА

Предположим, что список представляет собой массив и берется из элемента `List`. Причем в списке должно выводиться количество элементов в нем.

Как бы вы это сделали?

РЕАЛИЗАЦИЯ `List`

```
const List = props =>
  <ul className={props.className}>{props.children}</ul>;

const ListItem = props =>
  <li className={props.className}>{props.children}</li>;

// в родительском компоненте
const todos = ['Написать отчет', 'Купить молоко', 'Забрать машину из ремонта'];

const listItems = todos.map((item) => <ListItem>{item}</ListItem>);

return (
<List className='list'>
  {listItems}
  <li><b>Всего:</b> {listItems.length}</li>
</List>
);
```

НЕ СОБЛЮДЁН ПРИНЦИП ЕДИННОЙ ОТВЕТСТВЕННОСТИ

Данный вариант верный, но проблема заключается в том, что мы вынесли логику подсчета за пределы нашего компонента, а это означает, что если мы захотим создать еще раз такого рода список, то нам придется заново выполнить все пункты.

ПЕРЕНОСИМ ЛОГИКУ В `List`

```
const List = props => (
<ul className={props.className}>
  {props.children(props.items)}
  <li><b>Всего:</b> {props.items.length}</li>
</ul>
);
const ListItem = props =>
  <li className={props.className}>{props.children}</li>;

// в родительском компоненте
const todos = ['Написать отчет', 'Купить молоко', 'Забрать машину из ремонта'];

return (
<List className='list' items={todos}>
  {items => items.map((item, index) => <ListItem key={index}>{item}</ListItem>)}
</List>
);
```

ПОЧЕМУ ЭТО РАБОТАЕТ?

Все дело в том, что в атрибут `props.children` заносится функция, а не DOM элементы. В свою очередь в данной функции мы можем делать практически все, что пожелаем с дочерними компонентами, причем данные мы также можем брать от родительского компонента.

КОГДА ЭТО УДОБНО?

Это удобно когда вам нужно передать какие-то параметры от родительского компонента дочернему. К примеру если у вас для родителя определенно какое-то внутреннее состояние, которое может меняться, то вы можите об этом "информировать" дочерние элементы.

ВОЗМОЖНОСТЬ СКРЫТИЯ КОНТЕНТА

Создадим файл `section.js` в директории `components`:

```
function Section(props) {
  const [hidden, setHidden] = useState(false);

  toggleSection = () => { setHidden(prev => !prev); }

  return (<section>
    <h1>{this.props.title}</h1>
    <div className='content'>
      {this.props.children(this.state.hidden)}
    </div>
    <button onClick={this.toggleSection}>
      Переключить
    </button>
  </section>);
}
```

ЗАДАДИМ ФУНКЦИЮ КАК ДОЧЕРНИЙ ЭЛЕМЕНТ

В файле `index.js` импортируем данный компонент и зададим ему функцию, как дочерний элемент:

```
// в родительском компоненте
return (
  <Section title='Истории из детства'>
  {
    (isHidden) => isHidden ? null : <p>Большой блок текста</p>
  }
  </Section>
);
```

ЛОГИКА КОМПОНЕНТА `Section`

Таким образом, информация о состоянии родителя может передаваться его дочерним элементам и если мы нажмем на кнопку "Переключатель", то и будет прятаться или показываться блок текста в зависимости от состояния нашего компонента.

SPREAD ОПЕРАТОР - ОПЕРАТОР РАЗВОРОТА

ОПЕРАТОР РАЗВОРОТА

Оператор разворота был стандартизирован в *ES6*. Он позволяет "разворачивать" свойства объекта, или элементы массива.

КОМПОНЕНТ С МНОЖЕСТВОМ АТРИБУТОВ

У нас есть компонент, который выводит полную информацию о пользователе:

```
const User = props => (
<ul>
    <li>{props.name}</li>
    <li>{props.age}</li>
    <li>{props.school}</li>
    <li>{props.university}</li>
    <li>{props.work}</li>
    <li>{props.hobby}</li>
</ul>
);
```

Как можно задать атрибуты для данного компонента?

СТАНДАРТНЫЙ СПОСОБ ЗАДАТЬ АТРИБУТЫ

```
// в родительском компоненте
return (
<User
  name='Петр Петрович'
  age={28}
  school='СПБ №33'
  university='СПБГУ'
  work='Главный инжинер'
  hobby='рыбалка, музыка, игры' />
);
```

ГРОМОЗДКО И СЛОЖНОЧИТАЕМО

Получается громоздко и сложночитаемо, так как атрибутов много и они не помещаются в заветные 80 символом.

Возникает вопрос как решить данную проблему?

ИСПОЛЬЗУЕМ ОПЕРАТОР РАЗВОРОТА

```
// в родительском компоненте
const props = {
  name: 'Петр Петрович',
  age: 28,
  school: 'СПБ №33',
  university: 'СПБГУ',
  work: 'Главный инжинер',
  hobby: 'рыбалка, музыка, игры'
}

return (
  <User {...props}/>
);
```

ЧТО ПОД КАПОТОМ?

Мы задали объект `props` с определенными свойствами, а после с использованием оператора разворота, передали свойства находящиеся в нашем объекте в наш компонент, где ключ - название атрибута, а значение по ключу, значение атрибута.

`{...props}` СОЗДАЕТ НОВЫЙ ОБЪЕКТ

Важно понимать, что `{...props}` создает новый объект в который копируются все свойства `props`, и таким образом мы избавляемся от побочных эфектов "*принипа иммутабельности*" (о нем пойдет речь позже).

Таким образом компонент `User` обладает всеми нужными ему атрибутами, и читаймость кода существенно улучшилась.

КАК ПЕРЕПИСАТЬ АТРИБУТ?

Возможно, что во время проектирования интерфейсов вам придется переписать одно из свойств там, где вы использовали *spread operator*. Для данной цели возможно просто переопределить атрибут после задания `{...props}`, то есть:

```
ReactDOM.render(
  <User {...props} age={30} />,
  document.getElementById('root')
);
```

КАК ЕЩЕ МОЖНО ЭТО ИСПОЛЬЗОВАТЬ?

Перепишим наш компонент таким образом, чтоб мы получали атрибуты `name` и `age` не в составе объекта `props`:

```
const User = ({school, name, ...props}) => (
  <ul>
  <li>{name}</li>
  <li>{props.age}</li>
  <li>{school}</li>
  <li>{props.university}</li>
  <li>{props.work}</li>
  <li>{props.hobby}</li>
  </ul>
);
```

"ВЫТАСКИВАЕМ" ЗНАЧЕНИЯ СВОЙСТВ

В данном случае из объекта `props` мы "вытаскиваем" значения свойств из переданного объекта, а оставшиеся свойства помещаются в объект `props` при помощи оператора расширения `...props` и соответсвенно в коде из можно использовать без указания `props.name` и `props.shool`.

А ЕСЛИ УКАЗАТЬ ИХ ЧЕРЕЗ `props` ?

А что если указать через `props` "вынимаемые" свойства?

```
const User = ({school, name, ...props}) => (
  <ul>
  <li>{props.name}</li>
  <li>{props.age}</li>
  <li>{props.school}</li>
  <li>{props.university}</li>
  <li>{props.work}</li>
  <li>{props.hobby}</li>
  </ul>
);
```

СВОЙСТВА ПРИМУТ ЗНАЧЕНИЕ `undefined`

В данном случае `props.name` и `props.school` примут значение `undefined`.

ПОМЕНЯЕМ ПАРАМЕТРЫ МЕСТАМИ

Чтобы была возможность использовать свойства одновременно, принимаемые параметры должны быть обозначенны так:

```
const User = ({...props, school, name}) => (
  <ul>
  <li>{props.name}</li>
  <li>{props.age}</li>
  <li>{props.school}</li>
  <li>{props.university}</li>
  <li>{props.work}</li>
  <li>{props.hobby}</li>
  </ul>
);
```

ЧТО ИЗМЕНИЛОСЬ?

Во втором случает в объект `props` будут скопированы , а не извлечены все свойства передаваемого объекта (в нашем случае объекта с атрибутами), после чего в `school` и `name` извлекутся одноименные свойства.

нетология

Спасибо за внимание!

Время задавать вопросы ☒

АЛЕКСАНДР СИВЦОВ

Александр Сивцов