



Передовые  
инженерные  
школы



МИНОБРНАУКИ  
РОССИИ



УНИВЕРСИТЕТ  
ИННОПОЛИС

## Занятие 25

useMemo | useReducer | useCallback |  
useContext

# План занятия



ПЕРЕДОВАЯ  
ИНЖЕНЕРНАЯ ШКОЛА  
УНИВЕРСИТЕТА ИННОПОЛИС

1. `useMemo`
2. `useReducer`
3. `useCallback`
4. `useContext`

Время от времени компонентам React приходится выполнять дорогостоящие вычисления. Например, при наличии большого списка сотрудников и поискового запроса компонент должен фильтровать имена сотрудников по запросу.

как и когда использовать `useMemo()`



Время от времени компонентам React приходится выполнять дорогостоящие вычисления. Например, при наличии большого списка сотрудников и поискового запроса компонент должен фильтровать имена сотрудников по запросу.

как и когда использовать `useMemo()`

`useMemo()` это встроенный `React`-хук, который принимает 2 аргумента – функцию, `compute` которая вычисляет результат, и `dependencies` массив:

```
const memoizedResult = useMemo(compute, dependencies)
```

# useMemo

```
const memoizedResult = useMemo(compute, dependencies)
```

Во время начального рендеринга `useMemo(compute, dependencies)` вызывает `compute`, запоминает результат вычисления и возвращает его компоненту.

Если зависимости не изменяются во время следующих визуализаций, то `useMemo()` не вызывается `compute`, но возвращается сохраненное значение.

Но если зависимости меняются во время повторного рендеринга, то `useMemo()` вызывает `compute`, запоминает новое значение и возвращает его.

В этом суть `useMemo()` hook.

# useMemo

```
import { useState } from 'react';

export function CalculateFactorial() {
  const [number, setNumber] = useState(1);
  const [inc, setInc] = useState(0);

  const factorial = factorialOf(number);

  const onChange = event => {
    setNumber(Number(event.target.value));
  };
  const onClick = () => setInc(i => i + 1);

  return (
    <div>
      Factorial of
      <input type="number" value={number} onChange={onChange}
    />
      is {factorial}
      <button onClick={onClick}>Re-render</button>
    </div>
  );
}

function factorialOf(n) {
  console.log('factorialOf(n) called!');
  return n <= 0 ? 1 : n * factorialOf(n - 1);
}
```

## Пример

Компонент `<CalculateFactorial />`  
вычисляет факториал числа,  
введенного в поле ввода.

# useMemo

Каждый раз, когда меняется входное значение, вычисляется факториал, `factorialOf(n)` который `'factorialOf(n) called!'` записывается в консоль.

Каждый раз, когда нажимается кнопка повторного отображения, `inc` значение состояния обновляется. Обновление `inc` значения состояния запускает `<CalculateFactorial />` и повторный рендеринг. Но, в качестве вторичного эффекта, во время повторного рендеринга факториал пересчитывается снова - `'factorialOf(n) called!'` записывается в консоль.

Используя `useMemo(() => factorialOf(number), [number])` вместо `simple factorialOf(number)`, React запоминает вычисление факториала.

# useMemo

```
import { useState, useMemo } from 'react';

export function CalculateFactorial() {
  const [number, setNumber] = useState(1);
  const [inc, setInc] = useState(0);

  const factorial = useMemo(() => factorialOf(number), [number]);

  const onChange = event => {
    setNumber(Number(event.target.value));
  };
  const onClick = () => setInc(i => i + 1);

  return (
    <div>
      Factorial of
      <input type="number" value={number} onChange={onChange} />
      is {factorial}
      <button onClick={onClick}>Re-render</button>
    </div>
  );
}

function factorialOf(n) {
  console.log('factorialOf(n) called!');
  return n <= 0 ? 1 : n * factorialOf(n - 1);
}
```



# useCallback

`useCallback()` по сравнению с `useMemo()` более специализированный hook, который запоминает обратные вызовы

```
import { useCallback } from 'react';

function MyComponent({ prop }) {
  const callback = () => {
    return 'Result';
  };
  const memoizedCallback = useCallback(callback, [prop]);

  return <ChildComponent callback={memoizedCallback} />;
}
```

`useCallback(() => {...}, [prop])` возвращает тот же экземпляр функции, пока `prop` зависимость остается неизменной

# useCallback

МОЖНО ИСПОЛЬЗОВАТЬ `useMemo()` для запоминания обратных вызовов

```
import { useMemo } from 'react';

function MyComponent({ prop }) {
  const callback = () => {
    return 'Result';
  };
  const memoizedCallback = useMemo(() => callback, [prop]);

  return <ChildComponent callback={memoizedCallback} />;
}
```

# useCallback

## Используйте useMemo с осторожностью

`useMemo()` может повысить производительность компонента, в начале выполнить профилирование компонента без и с `useMemo()`. Только после этого определяйте нужно использовать или нет.

Когда `useMemo()` используется ненадлежащим образом, это снизить производительность.



# useCallback

**Используйте useMemo и useCallback с осторожностью**

useMemo() может повысить производительность компонента, в начале выполнить профилирование компонента без и с useMemo(). Только после этого определяйте нужно использовать или нет.

Когда useMemo() используется ненадлежащим образом, это снизить производительность.



## useState vs useReducer: что это такое и когда их использовать?

Рассмотрим пример выборки данных:

Если у нас есть состояние, представляющее данные, которые извлекли из **API**, состояние будет либо одним из трех "состояний": **loading**, **data** или **error**

Когда извлекаем данные из **API**, состояние переходит из **loading** (ожидание получения данных) либо в **data**, либо получим **error**

сравним, как обрабатываем состояние с помощью хука **useState** и хука **useReducer**

# useReducer



## useState hook:

```
function Fetcher() {  
  const [loading, setLoading] = useState(true)  
  const [data, setData] = useState(null)  
  const [error, setError] = useState(false)  
  
  useEffect(() => {  
    fetch('https://jsonplaceholder.typicode.com/posts/1').then(res => {  
      setLoading(false)  
      setData(res.data)  
      setError(false)  
    }).catch((err) => {  
      setLoading(false)  
      setData(null)  
      setError(true)  
    })  
  }, [])  
  
  return (  
    {loading ? <p>Loading...</p>  
      : <div>  
        <h1>{data.title}</h1>  
        <p>{data.body}</p>  
      </div> }  
    {error && <p>"An error occurred"</p> }  
  )  
}
```

## useReducer hook:

```
const initialState = {
  loading: true,
  data: null,
  error: false
}

const reducer = (state, action) => {
  switch (action.type) {
    case "SUCCESS":
      return {
        loading: false,
        data: action.payload,
        error: false
      };

    case "ERROR":
      return {
        loading: false,
        data: null,
        error: true
      };

    default:
      return state;
  }
}
```

```
function Fetcher() {
  const [state, dispatch] = useReducer(reducer, initialState)

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts/1').then(res => {
      dispatch({type: "SUCCESS", payload: res.data})
    }).catch(err => {
      dispatch({type: "ERROR"})
    })
  }, [])

  return (
    {state.loading ? <p>Loading...</p>
    : <div>
      <h1>{state.data.title}</h1>
      <p>{state.data.body}</p>
    </div> }
    {state.error && <p>"An error occurred"</p> }
  )
}
```

С помощью useReducer, сгруппировали три состояния вместе и также обновили их вместе.

Хук useReducer **чрезвычайно полезен**, когда есть состояния, которые связаны друг с другом.

Попытка обработать их все с помощью хука useState может вызвать трудности в зависимости от сложности и бизнес-логики этого.



В React context больше похож на глобальную переменную, которая может использоваться во всех компонентах приложения.

Использовать context только тогда, когда нужно, чтобы некоторые данные были доступны многим компонентам.



# useContext

```
import { useState, createContext, useMemo } from 'react';    userDetails.jsx

const UserContext = createContext();

const UserProvider = (props) => {
  const [username, setUsername] = useState('');

  const value = useMemo(
    () => ({username, setUsername}), [username])

  return (
    <UserContext.Provider
      value={value}
    >
      {props.children}
    </UserContext.Provider>
  );
}
export { UserContext, UserProvider };
```

# useContext

```
import { useState, createContext, useMemo } from 'react';    userDetails.jsx

const UserContext = createContext();

const UserProvider = (props) => {
  const [username, setUsername] = useState('');

  const value = useMemo(
    () => ({username, setUsername}), [username])

  return (
    <UserContext.Provider
      value={value}
    >
      {props.children}
    </UserContext.Provider>
  );
}
export { UserContext, UserProvider };
```

использовали `useMemo` для сохранения значений в `Provider`, это делается для предотвращения повторного рендеринга приложения, когда значение не обновляется.

# useContext



```
import { BrowserRouter, Routes, Route } from "react-router-dom";  
import { UserProvider } from './userDetails';
```

```
const App = () => {  
  return (  
    <UserProvider>  
      <BrowserRouter>  
        <Routes>  
          <Route path="/" exact component={SetUserDetails} />  
          <Route  
            path="/user"  
            exact  
            component={FetchUserDetails} />  
        </Routes>  
      </BrowserRouter>  
    </UserProvider>  
  )  
}
```

```
export default App;
```

# useContext



## SetUserDetails.jsx

```
import React, { useState, useContext } from "react";
import { useHistory } from "react-router-dom";
import { UserContext } from "../userDetails";

const SetUserDetails = () => {
  const [name, setName] = useState("");
  const history = useHistory();

  const { setUsername } = useContext(UserContext);
  const handleSetName = () => {
    setUsername(name);
    history.push("/user");
  };
  return (
    <>
      <input
        value={name}
        onChange={(e) => setName(e.target.value)} />
      <button onClick={handleSetName}>Set Name </button>
    </>
  );
};

export default SetUserDetails;
```

# useContext



## FetchUserDetails.jsx

```
import React, { useContext } from "react";
import { UserContext } from "../userDetails";

const FetchUserDetails = () => {
  const { username } = useContext(UserContext);

  return <>{username ? `Hello ${username}` : `Hello User`}</>;
};

export default FetchUserDetails;
```



Передовые  
инженерные  
школы



МИНОБРНАУКИ  
РОССИИ



УНИВЕРСИТЕТ  
ИННОПОЛИС

# Спасибо за внимание

