Debounce Input React TS

how to do debounce function for input with react ts

To implement a debounce function for an input field in a React TypeScript application, you can use the `useState`, `useEffect`, and `useRef` hooks, along with `setTimeout` to delay the function call until the user has stopped typing for a specified time.

Here’s an example of how you can implement this:

Step-by-Step Implementation:

1. **Install lodash (optional)**: If you want to use a pre-built debounce function, you can install `lodash`:

   ```bash
   npm install lodash
   ```

   Alternatively, you can implement your own debounce logic.

2. **Create a Custom Hook or Function for Debouncing**:

   Here's an example without using external libraries:

   ```tsx
   import React, { useState, useEffect, useRef } from 'react';

   const DebounceInput: React.FC = () => {
     const [inputValue, setInputValue] = useState<string>('');
     const [debouncedValue, setDebouncedValue] = useState<string>('');

     const debouncedFn = useRef<NodeJS.Timeout | null>(null);

     const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
       setInputValue(e.target.value);
     };

     useEffect(() => {
       // Clear the timeout if inputValue changes again before the delay
       if (debouncedFn.current) {
   ```

```
      clearTimeout(debouncedFn.current);
    }

    debouncedFn.current = setTimeout(() => {
      setDebouncedValue(inputValue);
    }, 500); // 500ms delay before updating debouncedValue

    // Cleanup function
    return () => {
      if (debouncedFn.current) {
        clearTimeout(debouncedFn.current);
      }
    };
  }, [inputValue]);

  return (
    <div>
      <input
        type="text"
        value={inputValue}
        onChange={handleChange}
        placeholder="Type something..."
      />
      <p>Debounced Value: {debouncedValue}</p>
    </div>
  );
};

export default DebounceInput;
```

Key Points:

1. `inputValue`: The current value from the input field.
2. `debouncedValue`: The value that updates after the debounce delay (e.g., 500ms).
3. `useEffect`: The hook listens for changes in `inputValue` and only updates `debouncedValue` after the user has stopped typing for a specified time.
4. `setTimeout`: Delays the update to `debouncedValue` to ensure that rapid changes don’t trigger an update too often.

5. `useRef`: Keeps a reference to the timeout so that you can cancel it if the component unmounts or the input changes again before the timeout is reached.

Using lodash for Debouncing:

If you prefer to use `lodash`, it simplifies the debouncing:

```tsx
import React, { useState } from 'react';
import { debounce } from 'lodash';

const DebounceInput: React.FC = () => {
  const [inputValue, setInputValue] = useState<string>('');
  const [debouncedValue, setDebouncedValue] = useState<string>('');

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setInputValue(e.target.value);
  };

  const debouncedHandleChange = debounce((value: string) => {
    setDebouncedValue(value);
  }, 500);

  // Using useEffect to trigger the debounced function
  React.useEffect(() => {
    debouncedHandleChange(inputValue);
  }, [inputValue]);

  return (
    <div>
      <input
        type="text"
        value={inputValue}
        onChange={handleChange}
        placeholder="Type something..."
      />
      <p>Debounced Value: {debouncedValue}</p>
    </div>
  );
};
```

```
export default DebounceInput;
```

In this case, `lodash` provides a more concise way of handling debouncing. The `debounce` function automatically handles clearing the timeout when necessary.

and just with function without hook

If you want to implement a debounce function without using React hooks, you can create a standalone debounce function and use it directly in a React component. Here's how you can do that with just a function (no hooks):

Example: Debounce Function Without React Hooks

1. **Create a Debounce Function**: First, you need a debounce utility function that will return a debounced version of a given function. This function ensures that the action is only triggered after the user stops typing for a set delay.

```tsx
const debounce = (func: Function, delay: number) => {
  let timeoutId: NodeJS.Timeout;

  return (...args: any[]) => {
    if (timeoutId) {
      clearTimeout(timeoutId); // Clear the previous timeout
    }
    timeoutId = setTimeout(() => {
      func(...args); // Call the function after the delay
    }, delay);
  };
};
```

2. **Use the Debounced Function in a Component**: In the component, you'll call the debounced function when the input changes, but the actual update will only happen after the specified delay.

```tsx
```

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

const DebounceInput: React.FC = () => {
  const [inputValue, setInputValue] = useState<string>('');
  const [debouncedValue, setDebouncedValue] = useState<string>('');

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setInputValue(e.target.value);
  };

  const debouncedSetValue = debounce((value: string) => {
    setDebouncedValue(value);
  }, 500); // Debounce delay of 500ms

  const handleDebouncedChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const value = e.target.value;
    debouncedSetValue(value); // Call the debounced function
    setInputValue(value); // Set the immediate input value
  };

  return (
    <div>
      <input
        type="text"
        value={inputValue}
        onChange={handleDebouncedChange}
        placeholder="Type something..."
      />
      <p>Debounced Value: {debouncedValue}</p>
    </div>
  );
};

export default DebounceInput;
```

Key Points:

- `debounce` **function**: This function ensures that the actual update (`setDebouncedValue`) only happens after the specified delay (500ms).
- `handleChange`: This updates the immediate state (`inputValue`) to show what the user types.

- `debouncedSetValue` : This is the debounced version of the function that updates the state for the debounced value.

How it works:

- When the user types in the input field, `handleDebouncedChange` is triggered.
- The `debouncedSetValue` function is called, but the update to the `debouncedValue` state happens only after the user stops typing for 500ms (you can adjust the delay to your needs).
- The immediate value (`inputValue`) is updated instantly for responsiveness, while the `debouncedValue` is updated only after the debounce delay.

This approach avoids using React hooks directly for debouncing and can be used with class components as well as function components.

show me example react ts with search with debounce with loadsh

Here's an example of a React TypeScript component that implements a search input field with debouncing using `lodash` 's `debounce` function:

Steps:

1. Install `lodash` (if you haven't already):

   ```bash
   npm install lodash
   ```

2. **Debounced Search Component:**

```tsx
import React, { useState, useEffect } from 'react';
import { debounce } from 'lodash';

const SearchWithDebounce: React.FC = () => {
  const [searchQuery, setSearchQuery] = useState<string>('');
  const [debouncedSearchQuery, setDebouncedSearchQuery] = useState<string>('');

  // Function to handle input change
```

```
  const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setSearchQuery(event.target.value); // Immediately set the input value
  };

  // Debounced function that only triggers after the user stops typing for 500ms
  const debouncedSearch = debounce((query: string) => {
    setDebouncedSearchQuery(query); // Update the debounced search query
  }, 500); // 500ms delay

  // Effect to trigger the debounced search whenever the input value changes
  useEffect(() => {
    debouncedSearch(searchQuery); // Call the debounced function
    // Cleanup function to cancel the debounce if the component unmounts
    return () => {
      debouncedSearch.cancel();
    };
  }, [searchQuery]);

  return (
    <div>
      <input
        type="text"
        value={searchQuery}
        onChange={handleInputChange}
        placeholder="Search..."
      />
      <p>Search Query: {debouncedSearchQuery}</p>
    </div>
  );
};

export default SearchWithDebounce;
```

Explanation:

1. **State Setup:**
    - `searchQuery`: Stores the immediate value of the input field (what the user types).
    - `debouncedSearchQuery`: Stores the value of the input after the debounce delay.
2. **Debounced Search Function (`debouncedSearch`):**

  - We use `lodash`'s `debounce` function to create a debounced version of the function that updates the `debouncedSearchQuery` state. It waits 500ms after the user stops typing before executing the search logic.
3. `useEffect`:
   - Whenever `searchQuery` changes (as the user types), the `useEffect` hook will trigger and call the debounced search function (`debouncedSearch`).
   - The `return` function in `useEffect` ensures that any pending debounced calls are cancelled when the component unmounts or when `searchQuery` changes again.
4. **Input Handling:**
   - The `handleInputChange` function updates the `searchQuery` immediately with each keystroke, while the debounced value (`debouncedSearchQuery`) only updates after the delay.

Key Points:

- The `debouncedSearch` function is called after the user stops typing for 500ms.
- The `debouncedSearch.cancel()` in the cleanup function ensures that no outdated debounced calls are executed if the component unmounts or if a new search starts before the previous debounced call completes.
- The `searchQuery` is the real-time value, and `debouncedSearchQuery` represents the value that gets updated after the debounce delay.

Usage Example:

In your app, you would import and use the `SearchWithDebounce` component:

```tsx
import React from 'react';
import SearchWithDebounce from './SearchWithDebounce';

const App: React.FC = () => {
  return (
    <div>
      <h1>Search with Debounce Example</h1>
      <SearchWithDebounce />
    </div>
  );
};

export default App;
```

This will allow you to perform a search with debounced input, making it efficient by limiting the number of updates or API calls triggered by the user's typing.

Is this conversation helpful so far?