JavaScriptのdebounceの仕組みがよくわかっていないで理解したいという人向けにJavaScriptだけでなくReact, vue.jsではどのようにdebounceを利用することができるのか解説しています。lodashライブラリのdebounceを利用すれば簡単に設定できますが本書では自作のdebounceを作成することでdebounceの仕組みを理解してReact, Vue.jsでの設定方法を確認していきます。lodashライブラリを利用した場合の設定方法も合わせて説明を行なっています。

作成するdebounce関数はReact, vue.jsで共通なのでReact, Vue.jsで設定を行う前にPureなJavaScript(Vanilla JavaScript)でコードの説明と動作確認を行います。

目次

Debounceとは

例えばユーザがinput要素に入力を行いタイトルの更新を行いたいとします。input要素に入力した文字はネットワーク越しにあるサーバにデータを送信しなければなりません。文字を1文字1文字打つごとにデータを送信することができますがある程度入力した後に送信するこ

とができればネットワーク、サーバへの負荷が下がることは想像できるかと思います。すべて入力後に送信ボタンを押して送信という方法もありますが送信処理は自動で行いたい。そのような時に利用できるのがdebounceです。この例であればユーザが文字を入力してそのキーボードの打つ手が一定時間止まった時にサーバへの送信を行います。ユーザがキーボードを打つのをやめるまで処理を待たせることができます。タイトルが決まっており連続して文字を入力するのであれば1回のデータ送信で処理は完了します。このようにdebounceを利用することでシステムの処理の負荷を下げることができます。

特にReactやVue.jsなどでユーザとのインタラクションの多いアプリケーションを作成する際に負荷を減らしたいという時にdebounceを利用することができます。

debounceの他にthrottleというものもあります。一緒に説明されることから区別がつかなくなる場合もあります。それも当然でlodashのthrottleはdebounceのパラメータを変えただけのものだからかもしれません。ソースコードの確認できます。throttleはdebounceとは違い一定間隔ごとに処理を実行したい場合に利用することができます。画面でscrollを行う場合うdebounceではスクロール中に処理の発生を抑えてスクロールをやめた時に処理を実施するということができます。throttleの場合は1000ms後ごとに処理を実行するといったことが可能になるのでスクロールを実施すると1秒ごとに何か処理を行うといったことができます。

本文書ではinput要素への入力の例を通してdebounceの説明を行なっていきます。

Vanilla JavaScriptでdebounce

debounceを利用しない場合

index.htmlファイルを任意の場所に作成して下記のコードを記述します。

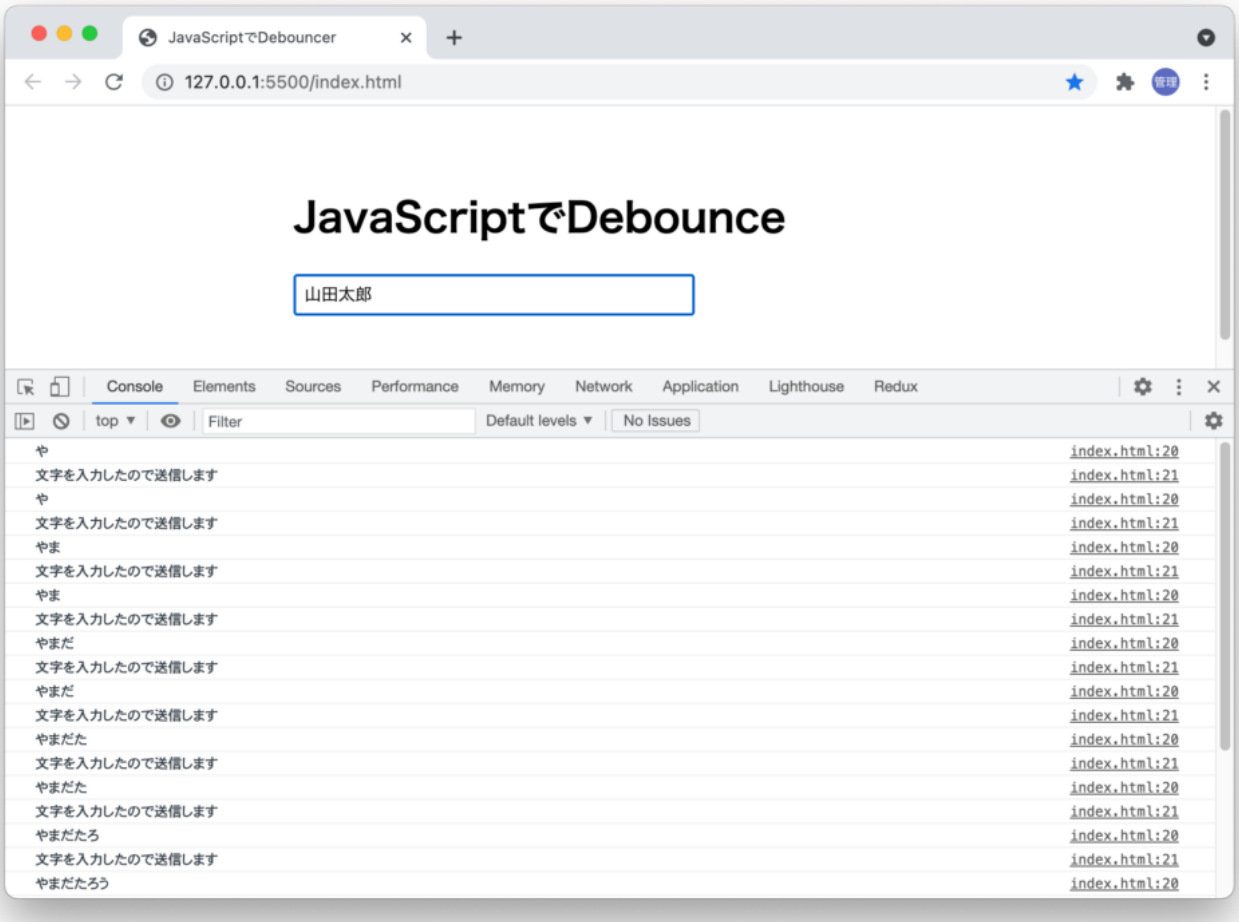
```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>JavaScriptでDebouncer</title>
  </head>
```

```
    <body>
      <div style="width: 500px; margin: 4em auto">
        <h1>JavaScriptでDebounce</h1>
        <input
          type="text"
          placeholder="名前を入力してください。"
          onkeyup="sendData()"
          style="padding: 0.5em; width: 300px"
          id="name"
        />
      </div>
      <script>
        const sendData = () => {
          console.log(document.getElementById('name').value);
          console.log('文字を入力したので送信します');
        };
      </script>
    </body>
  </html>
```

文字の入力ができるように画面中央にinput要素を表示させています。input要素への文字の入力はonkeyupイベントを利用してキャッチします。イベントをキャッチするとsendData関数を実行しコンソールにメッセージが表示されます。sendData関数の中ではgetElementByIdを利用してinput要素にアクセスして入力した文字を取得しています。

> 今回のsendData関数の中ではサーバへのデータ送信は行なっていませんが実際にこの中にデータへの送信処理を追加することになります。

ブラウザのデベロッパーツールを開いてinput要素に”山田太郎”と入力します。一文字打つごとにコンソールにメッセージが表示されることが確認できます。



input要素に文字列を入力

”山田太郎”と入力するだけでもすごい数のメッセージが表示されることがわかります。文字を入力するごとにメッセージを表示させないようにdebounce機能を追加します。

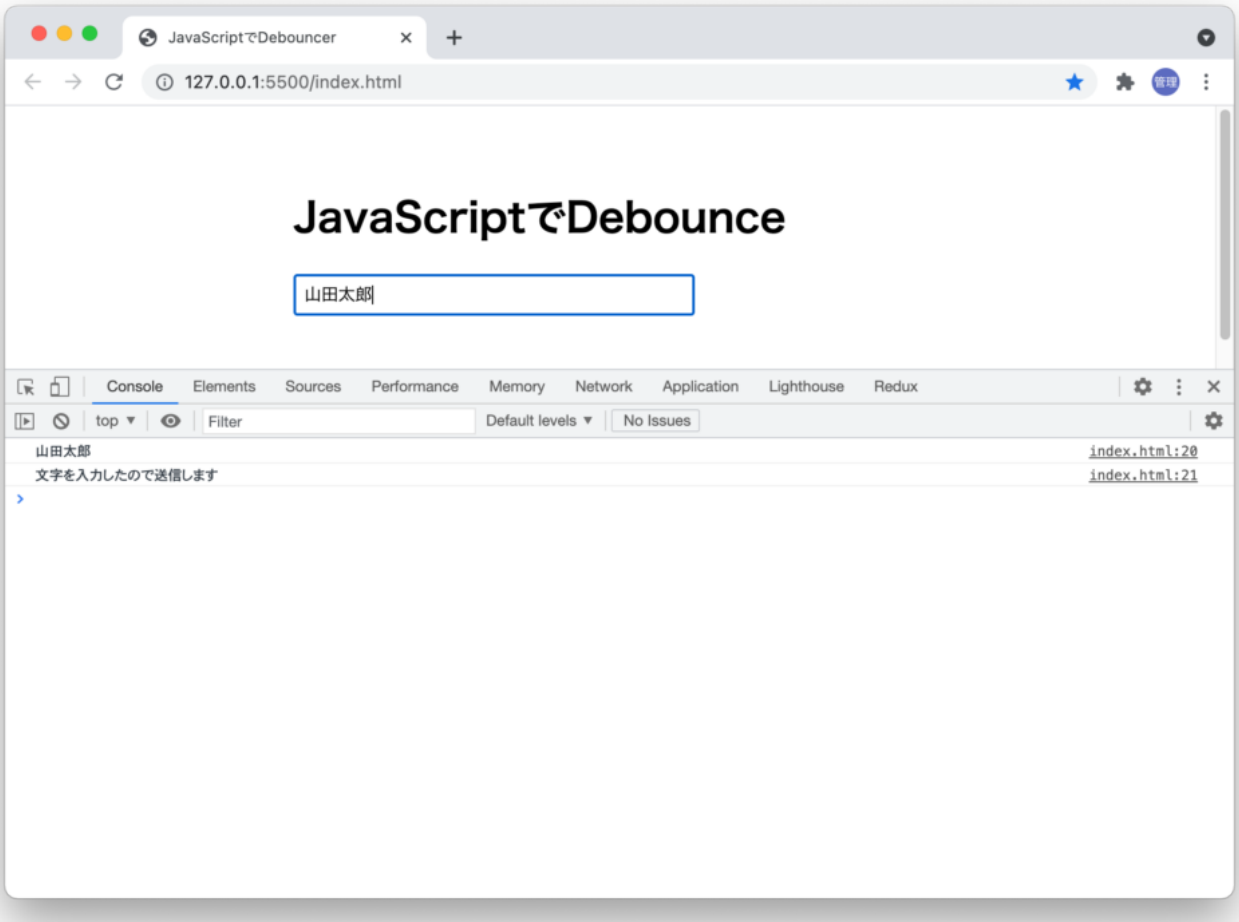
debounceを利用した場合

先ほど作成したindex.htmlファイルにdebounceのコード追加します。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>JavaScriptでDebouncer</title>
  </head>
  <body>
    <div style="width: 500px; margin: 4em auto">
```

```
      <h1>JavaScriptでDebounce</h1>
      <input
        type="text"
        placeholder="名前を入力してください。"
        onkeyup="debounceSendData()"
        style="padding: 0.5em; width: 300px"
        id="name"
      />
    </div>
    <script>
      const sendData = () => {
        console.log(document.getElementById('name').value);
        console.log('文字を入力したので送信します');
      };

      const debounce = (func, wait = 1000) => {
        let timerId;
        return (...args) => {
          if (timerId) {
            clearTimeout(timerId);
          }
          timerId = setTimeout(() => {
            func.apply(null, args);
          }, wait);
        };
      };

      const debounceSendData = debounce(() => sendData());
    </script>
  </body>
</html>
```

コードの説明の前に上記のコードを利用して”山田太郎”を入力してみましょう。先ほどとは異なり入力途中にキーボードを打つのを止めなければ”山田太郎”を入力した後一定時間が経過してコンソールにメッセージが表示されます。



debounce関数を利用した場合

このようにdebounceを利用することで処理を劇的に少なくすることが理解できたかと思います。

debounce関数の中身を確認

debounceの動きが文字の入力の例を通して理解できたと思うので次は追加したdebounce関数のコードの内容を確認しましょう。

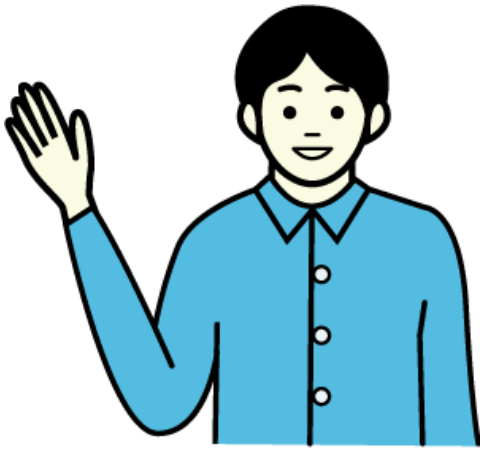
```
const debounce = (func, wait = 1000) => {
  let timerId;
  return (...args) => {
    if (timerId) {
      clearTimeout(timerId);
    }
    timerId = setTimeout(() => {
```

```
        func.apply(null, args);
      }, wait);
    };
  };
```

debounce関数にはfuncとwaitの2つの引数を取ることができます。今回の例ではfuncにはsendData関数が入り、waitには何も指定しなかったのでデフォルトの1000msが設定されます。

ユーザが1文字を入力するとtimerIdには何も設定されていないのでsetTimeoutが実行されます。1文字目を入れて1000ms経過するとsetTimeoutのスケジュールによりfunc関数(sendData)が実行されコンソールにメッセージが表示されます。連続で2文字を打った場合は2文字目でdebounceが実行された時にtimerIdが値を持っているのでclearTimeoutにより実行したsetTimeoutによるスケジュールはキャンセルされ、その後また新たにsetTimeoutが実行されます。キーを打ち続けてる間は一つ前の文字を入力した時に実行されたsetTimeoutによるスケジュールをキャンセルしているでfunc関数が実行されることはありません。キーを打つのをやめると最後の文字の入力時のsetTimeoutで実行したスケジュールがclearTimeoutによってキャンセルされないため1000ms後にfunc関数が実行されます。

> lodashのdebounceはオプションが複数あるためこれほどシンプルなコードではありませんがsetTimeoutを利用して制御しているという点では同じです。



最後に...argsはどの場所に設定した場合の引数なのかを確認します。argsはargmentsの略です。...argsの引数がどこの場所の引数なのかを確認するためにonkeyupで実行しているdebounceSendDataに引数を設定します。

```
<input
  type="text"
  placeholder="名前を入力してください。"
```

```
    onkeyup="debounceSendData('aaa','bbbb')"
    style="padding: 0.5em; width: 300px"
    id="name"
  />
```

debounce関数の中でconsole.logを利用してargsが表示されるか確認します。

```
const debounce = (func, wait = 1000) => {
  let timerId;
  return (...args) => {
    console.log(args);
    if (timerId) {
      clearTimeout(timerId);
    }
    timerId = setTimeout(() => {
      func.apply(null, args);
    }, wait);
  };
};
```

input要素で入力を行うとコンソールに配列で[`aaa',`bbb']が表示されます。setTimeout関数の中で実行されるapplyでは配列を第二引数で渡すことができるため[`aaa',`bbb']がfunc関数の引数として渡されることになります。

実際にapply関数で実行するfuncに引数を渡すと下記のようになります。

```
<body>
  <div style="width: 500px; margin: 4em auto">
    <h1>JavaScriptでDebounce</h1>
    <input
      type="text"
      placeholder="名前を入力してください。"
      onkeyup="debounceSendData('aaa','bbbb')"
      style="padding: 0.5em; width: 300px"
      id="name"
```

```
    />
  </div>
  <script>
    const sendData = (arg1, arg2) => {
      console.log(document.getElementById('name').value);
      console.log(`文字を入力したので送信します。${arg1} ${arg2}`);
    };

    const debounce = (func, wait = 1000) => {
      let timerId;
      return (...args) => {
        if (timerId) {
          clearTimeout(timerId);
        }
        timerId = setTimeout(() => {
          func.apply(null, args);
        }, wait);
      };
    };

    const debounceSendData = debounce((arg1, arg2) => sendData(arg1, arg2
  </script>
</body>
```

実行するとコンソールには”文字を入力したので送信します。aaa bbbb”が表示されます。

Reactによるdebounceの設定

debounceの理解とVanilla JavaScriptによるdebounceの動作確認が完了したので次にReactでのdebounceの動作確認を行います。自作したdebounceで設定を行いますが設定方法はlodashのdebounceを利用した時と同じです。React HookのuseCallback, useRef, useMemoを利用するのでReactで利用する場合はそれらのHookの理解も必要になります。

Reactでの動作確認を行うためにcreate-react-appでプロジェクトを作成してから実行します。

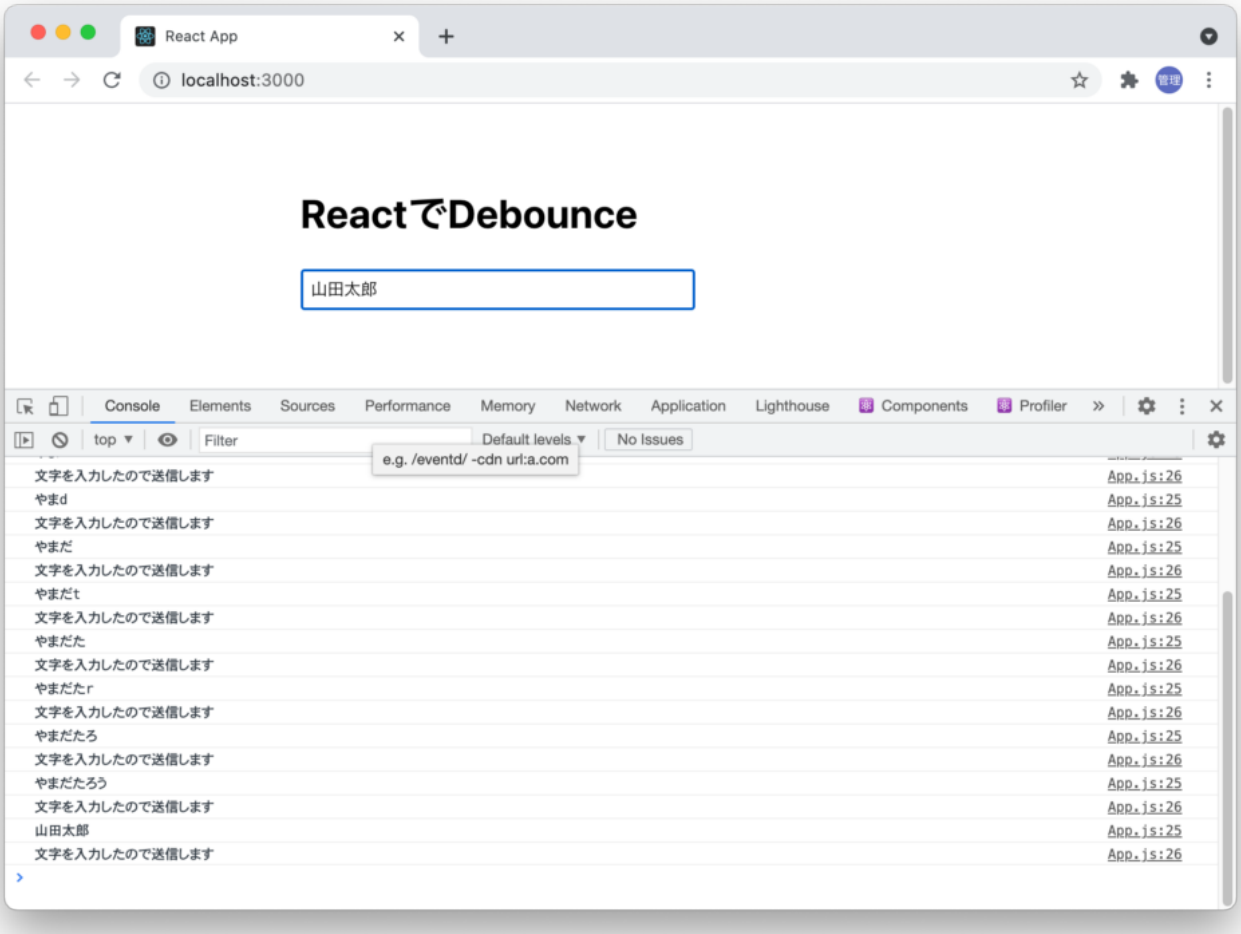
debounceを利用しない場合

動作確認にはsrcのApp.jsファイルのみ利用します。Reactでも最初はdebounceを利用しない場合の動作確認を行います。

```
import { useState } from 'react';
function App() {
  const [data, setData] = useState('');

  const handleChange = (e) => {
    setData(e.target.value);
    console.log(e.target.value);
    console.log('文字を入力したので送信します');
  };

  return (
    <div style={{ width: '500px', margin: '4em auto' }}>
      <h1>ReactでDebounce</h1>
      <input
        type="text"
        placeholder="名前を入力してください。"
        onChange={handleChange}
        style={{ padding: '0.5em', width: '300px' }}
        value={data}
      />
    </div>
  );
}

export default App;
```

React HookのuseStateを利用してinput要素で入力した文字をdataに保存します。
onChangeイベントを利用してhandleChange関数を設定しています。

動作確認を行うとdebounceを設定していないのでJavaScriptでdebounceを利用しない時と同じように複数回のメッセージが表示されます。JavaSciptの場合はkeyupイベントだったので少し表示されるメッセージが変わりますがかなりの数のメッセージが表示されます。



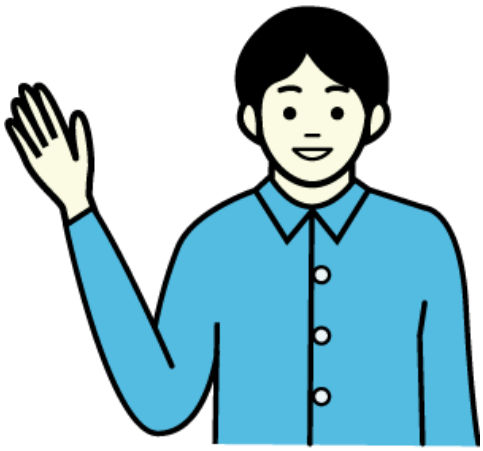
Reactでdebounceを利用しない場合

debounceを設定した場合

JavaSciptの時に利用したdebounceコードをそのまま利用します。debounceのコードに変更はないのでdebounceのコードに関する説明は省略します。

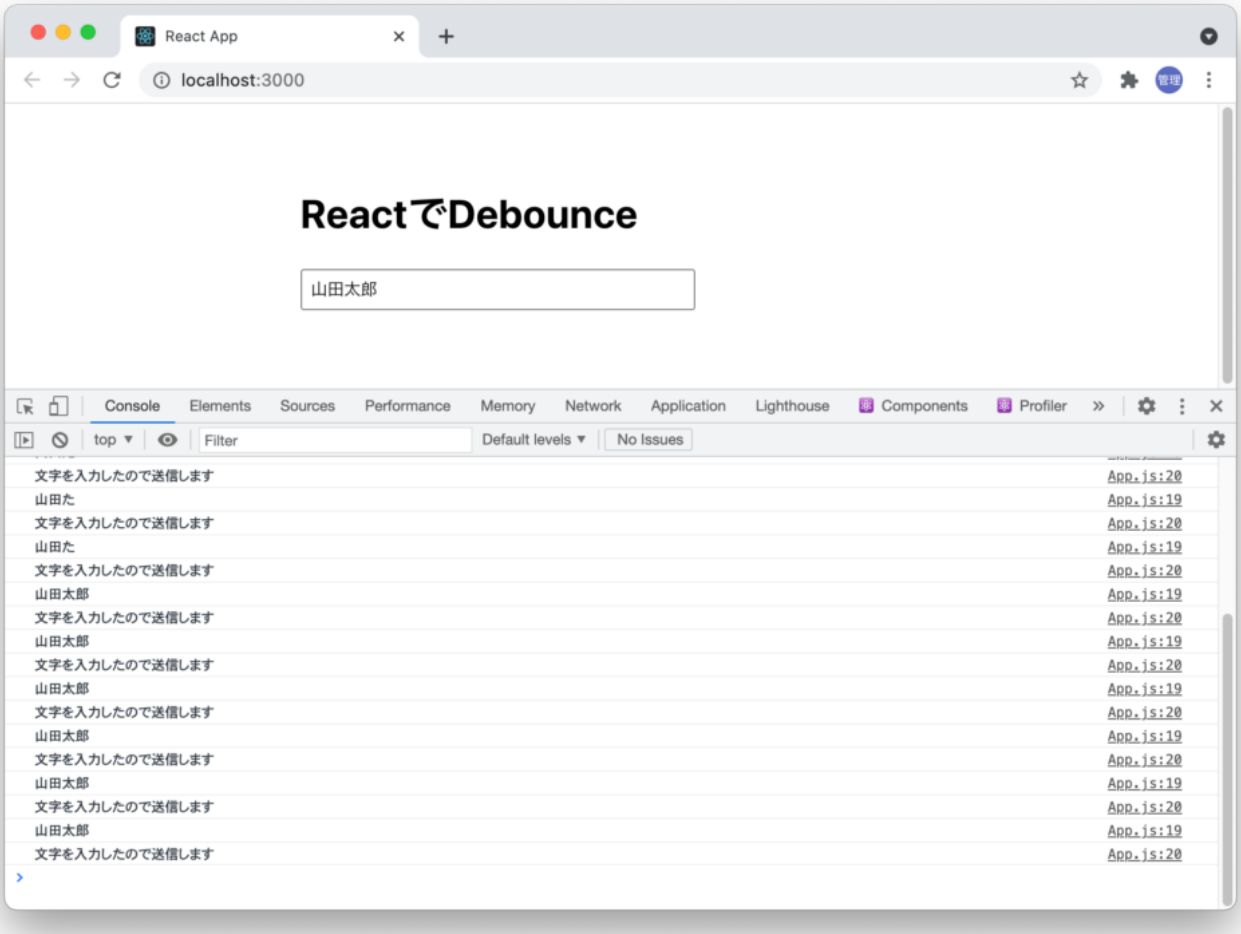
App.jsのコードにはいくつか変更を加えます。handleChange関数の中にdebouncehandelChangeを追加し、debouncehandleChange関数でdebounce関数を実行しています。debouncehandleChange関数にはイベントeを渡しています。イベントeはdebounce関数のcallbackの引数として受け取ることができるので受け取ったeからe.target.valueで入力した値を取得しています。

debounce関数の中で..argsとapply関数を利用しなければdebouncehandleChangeからイベントeを受け取ることはできません。



```
const debounce = (func, wait = 1000) => {
  let timerId;
  return (...args) => {
    if (timerId) {
      clearTimeout(timerId);
    }
    timerId = setTimeout(() => {
      func.apply(null, args);
    }, wait);
  };
};

const debouncehandleChange = debounce((e) => {
  console.log(e.target.value);
  console.log('文字を入力したので送信します');
});

const handleChange = (e) => {
  setData(e.target.value);
  debouncehandleChange(e);
};
```

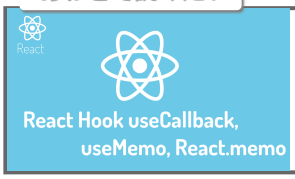
コードの更新が完了したらブラウザからinput要素に文字を入力してみましょう。コンソールに表示されるように現在の設定ではうまくdebounceを行うことができません。



debounceを設定して入力

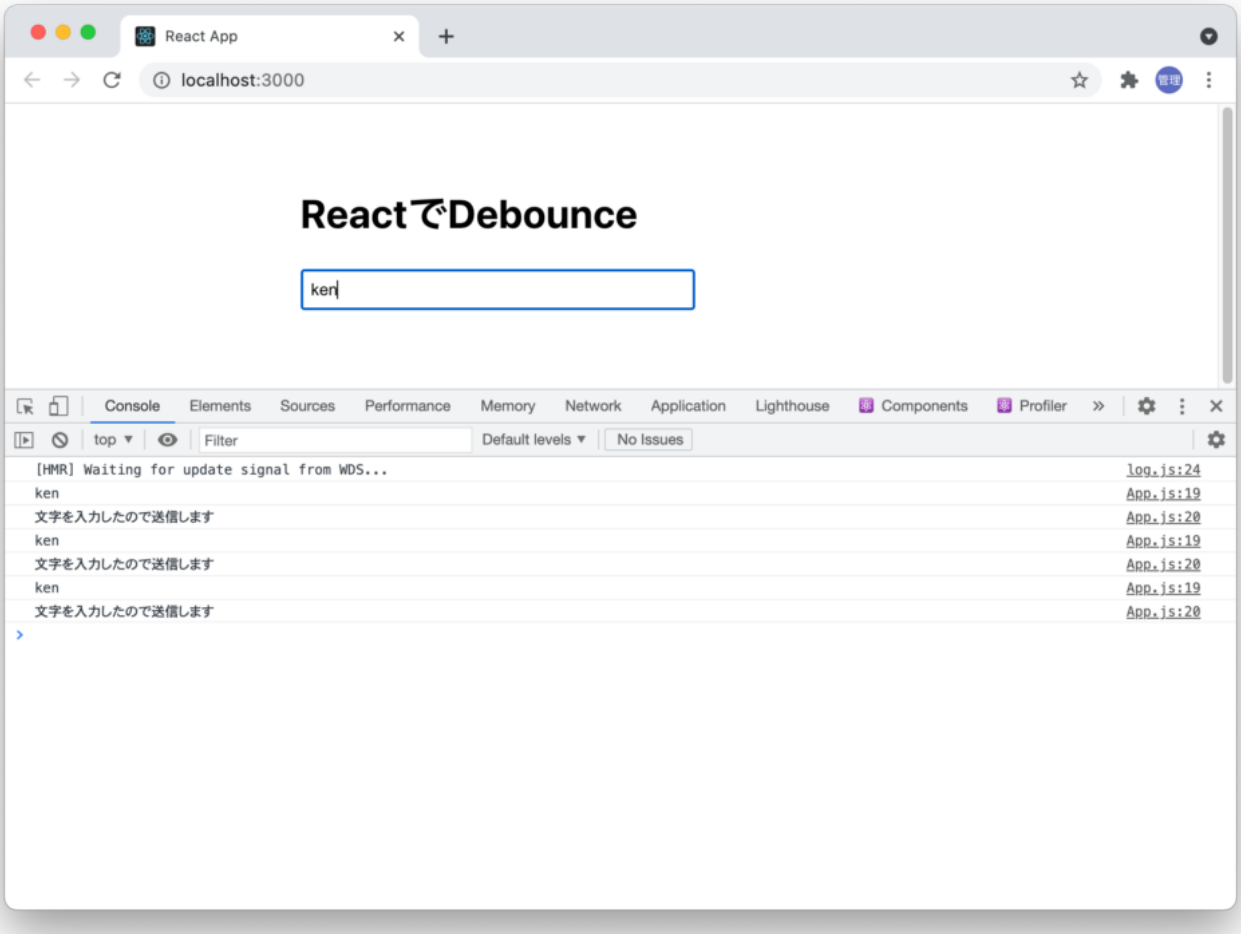
useStateを利用して状態(ここではdata)を更新する場合にコンポーネントの再描写が行われdebouncehandleChangeが新規で再描写された回数分作成され実行されることになります。

あわせて読みたい



Reactの再描写(re-render)って何??useCallback, useMemo, memoと一緒に理解。

再描写でdebouncehandleChangeが実行されているのか確認を行いますが日本語では変換があるのでわかりにくいのでkenをinput要素に入力してください。文字の入力分だけdebouncehandleChangeが実行され、それぞれ独立にメッセージが表示されます。キーの入力を止めてから1000ms後の値が表示されるのですべてkenと表示されます。



再描写の確認

再描写ごとに関数を実行させないための方法(実行した関数を再描写しても保持)としてReactではuseCallbakとuseMemo, useRefがあります。まずはuseCallbackから確認しましょう。

useCallbackの設定

reactからuseCallbackのimportを行います。debounce関数をuseCallbackで包みます。設定はこれで完了です。

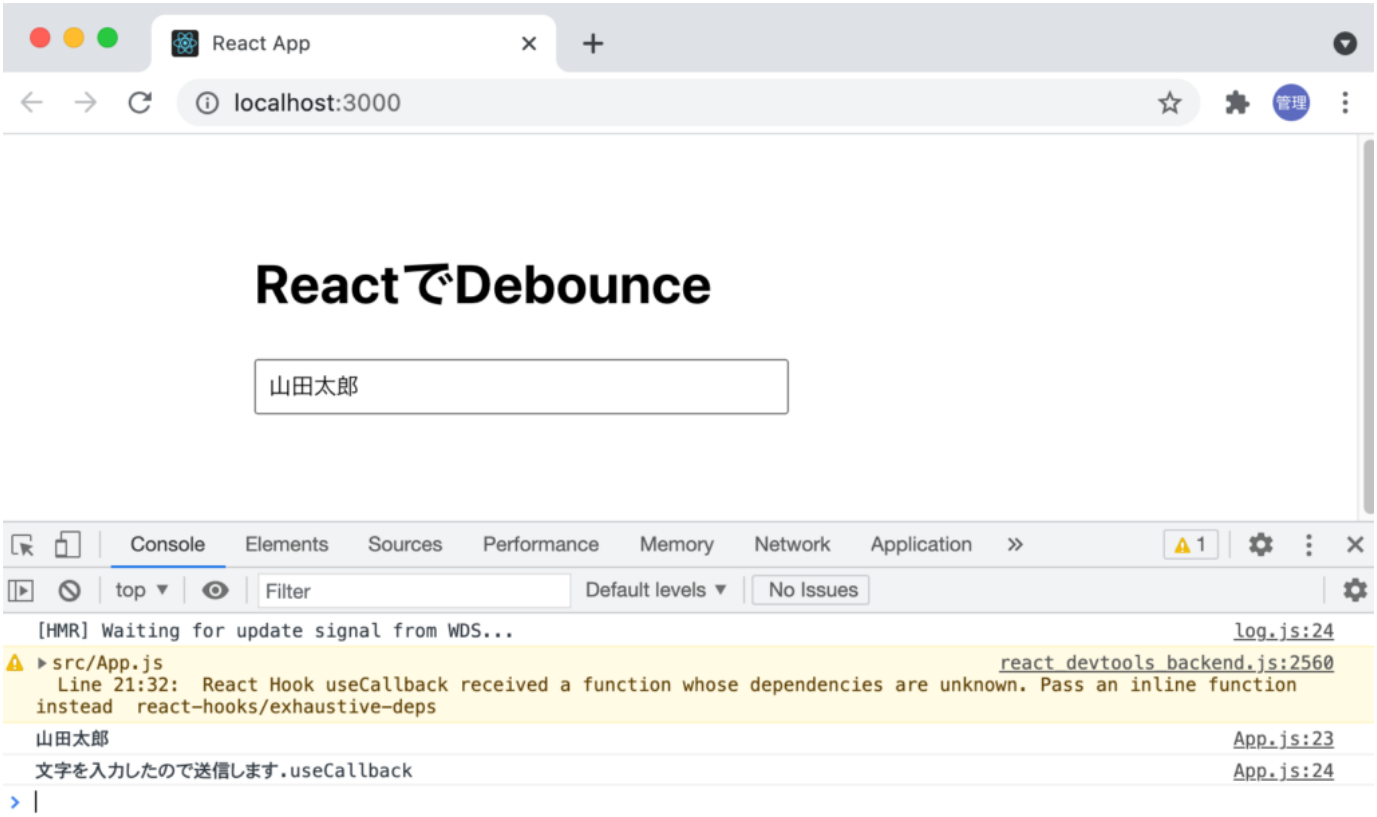
```
import { useState, useCallback } from 'react';
//略
  const debouncehandleChange = useCallback(
    debounce((e) => {
      console.log(e.target.value);
      console.log('文字を入力したので送信します.useCallback');
```

```
    }, 1000),
    []
  );
  //略
```

更新して保存するとコンソールにWarningが発生していることが確認できます。
useCallbakではinline functionを渡さないといけないという警告です。

```
Compiled with warnings.

src/App.js
  Line 21:32:  React Hook useCallback received a function whose dependenc:

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.
```

ブラウザ側にもReact Hook useCallback received a function whose dependencies are unknown. Pass an inline function instead react-hooks/exhaustive-depsのWarningが出力されていますがdebounce自体の動作はうまくいきます。



warningが発生

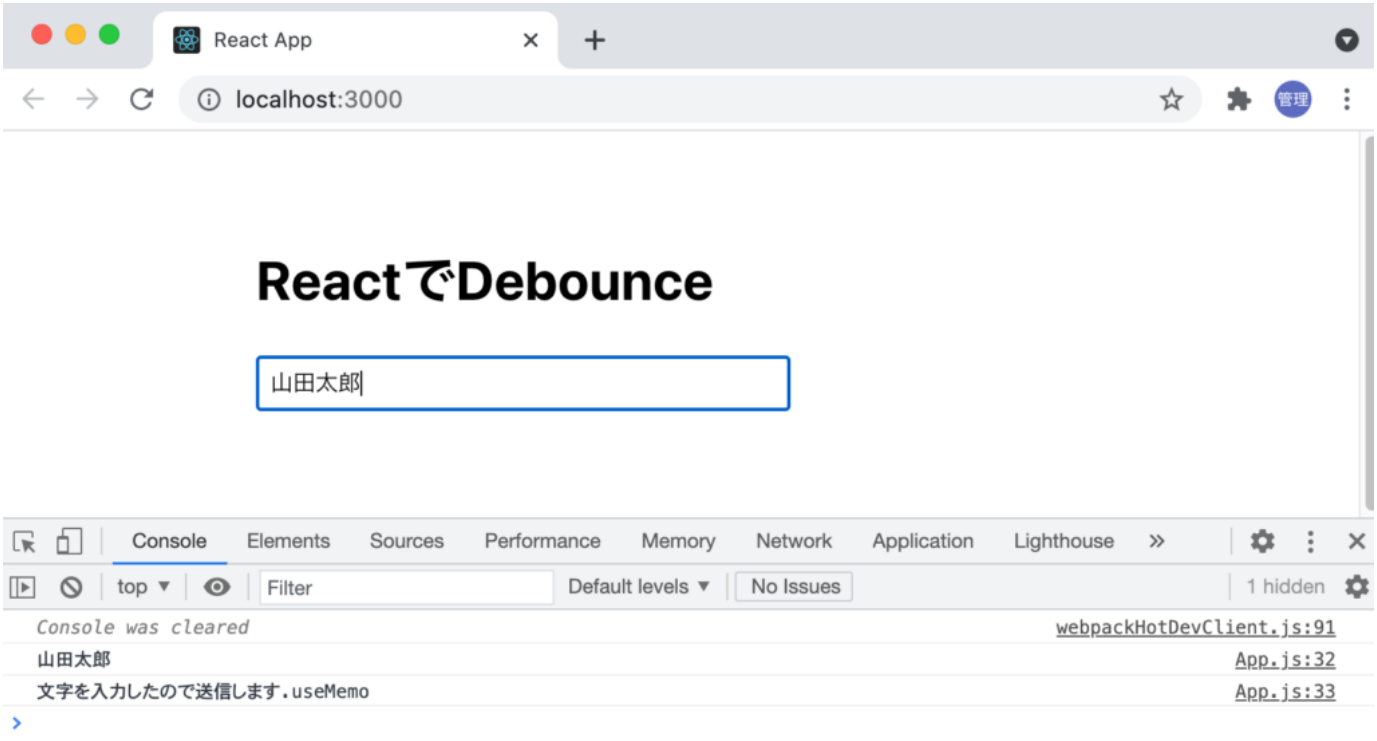
警告が出たままでは利用することはできないので次はuseMemoを利用して動作確認を行います。

useMemoの設定

importするHookをuseCallbackからuseMemoに変更します。debouncehandleChangeの関数の中身も一部変更を加えます。アロー関数を追加しています。

```
import { useState, useMemo } from 'react';
//略
  const debouncehandleChange = useMemo(
    () =>
      debounce((e) => {
        console.log(e.target.value);
        console.log('文字を入力したので送信します.useMemo');
      }, 1000),
    []
  );
  //略
```

更新を保存してもuseCallbackのように警告がでることはありません。useMemoを利用することで問題なくdebouncehandleChangeが何度も作成されることがなくなりdebounceが動作することが確認できます。



useMemoを利用した動作確認

lodashのdebounceを利用した場合

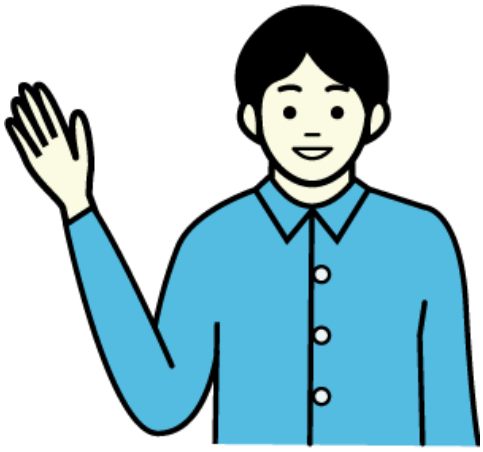
ここまでは自作のdebounceを利用してきましたが、現在のコードでlodashのdebounceが動作するのか確認するためにuseMemoでlodashのdebounceを利用して動作確認を行ってみましょう。Reactではlodashをインストールすることなくそのまま利用することができます。

```
import { useState, useMemo } from 'react';
import { debounce } from 'lodash';
function App() {
  const [data, setData] = useState('');

  // const debounce = (func, wait = 1000) => {
  //   let timerId;
  //   return (...args) => {
  //     if (timerId) {
  //       clearTimeout(timerId);
  //     }
  //     timerId = setTimeout(() => {
```

```
    //       func.apply(null, args);
    //     }, wait);
    //   };
    // };

    const debouncehandleChange = useMemo(
      () =>
        debounce((e) => {
          console.log(e.target.value);
          console.log('文字を入力したので送信します.useMemo');
        }, 1000),
      []
    );

    const handleChange = (e) => {
      setData(e.target.value);
      debouncehandleChange(e);
    };
  //略
```

動作確認を行うと自作のdebounceと同じ動作をすることが確認できます。lodashでもuseMemoを利用した方法が問題ないことが確認できました。

lodashのdebounceでuseCallbackを利用しても React Hook useCallback received a function whose dependencies are unknown. Pass an inline function instead react-hooks/exhaustive-depsは発生します。

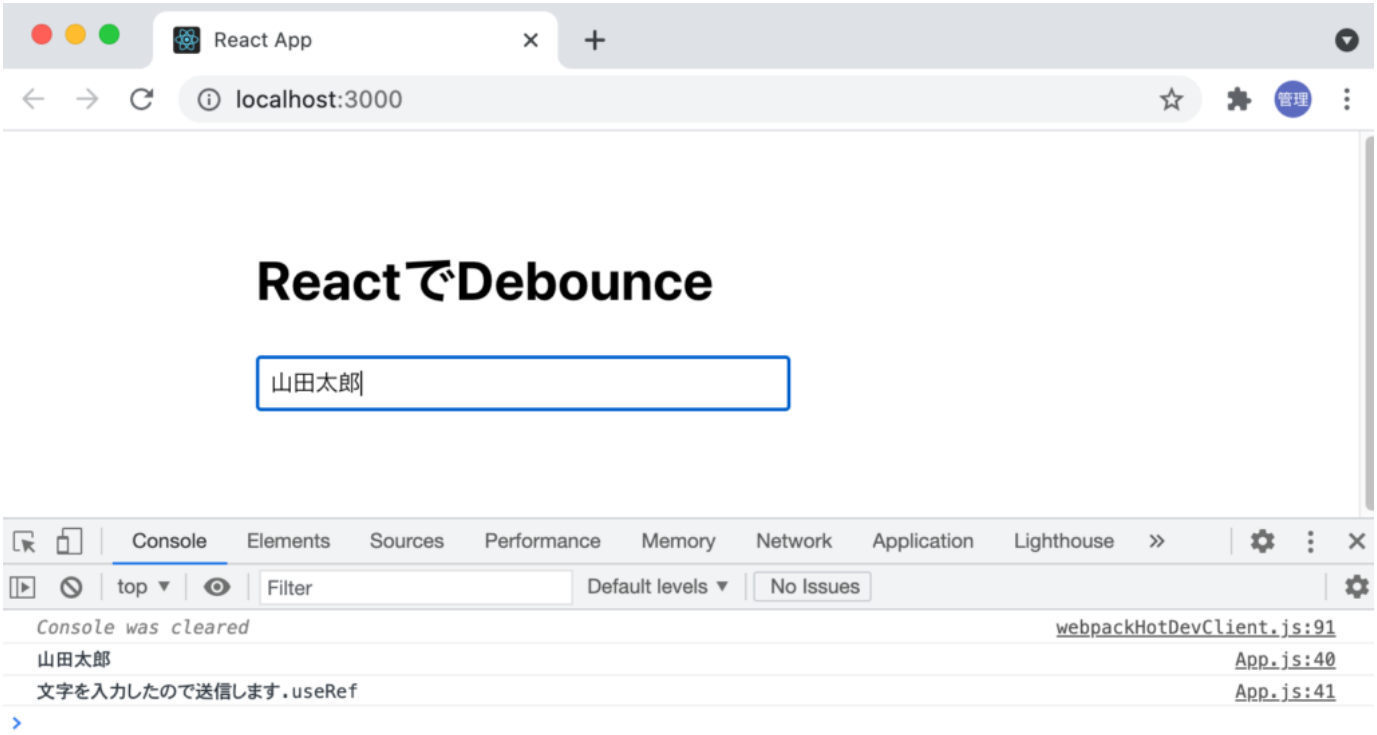


useRefの設定

useMemoの次はuseRefを利用して動作確認を行います。useRefを利用するのでReact HookのuseRefをimportします。debouncehandleChangeの中身もuseMemoから変更を行い、useCallbackの時と同じ記述になります。

```
import { useState, useRef } from 'react';
//略
  const debouncehandleChange = useRef(
    debounce((e) => {
      console.log(e.target.value);
      console.log('文字を入力したので送信します.useRef');
    }, 1000)
  ).current;

  //略
```

動作確認するとuseRefもuseMemoと同様に警告なしでdebounceが動作することが確認できます。



useRefを使ってdebounceを設定

Reactでは今回のコードでdebounceを利用する場合はuseMemo, useRefが利用できることがわかりました。useCallbakについてもinline functionを利用する場合は利用することができます。

Vue.jsによるdebounceの設定

debounceの理解とVanilla JavaScript, Reactによるdebounceの動作確認が完了したので次にVue.jsでのdebounceの動作確認を行います。

Vue.jsの環境はVue CLIを利用してVue3のプロジェクトを作成し動作確認を行なっています。

debounceを利用しない場合

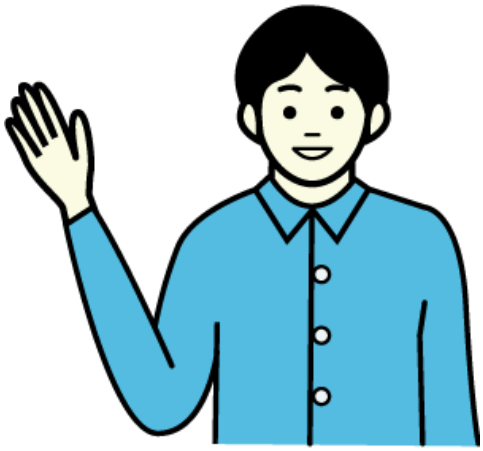
動作確認にはsrcフォルダのApp.vueファイルのみ利用します。Vue.jsでも最初はdebounceを利用しない場合の動作確認を行います。

```
<template>
  <div style="width: 500px; margin: 4em auto">
    <h1>Vue.jsでDebounce</h1>
    <input
      type="text"
      placeholder="名前を入力してください。"
      v-model="data"
      style="padding: 0.5em; width: 300px"
      id="name"
      @input="handleChange"
    />
  </div>
</template>

<script>
export default {
  data() {
    return {
      data: '',
    };
  },
  methods: {
```

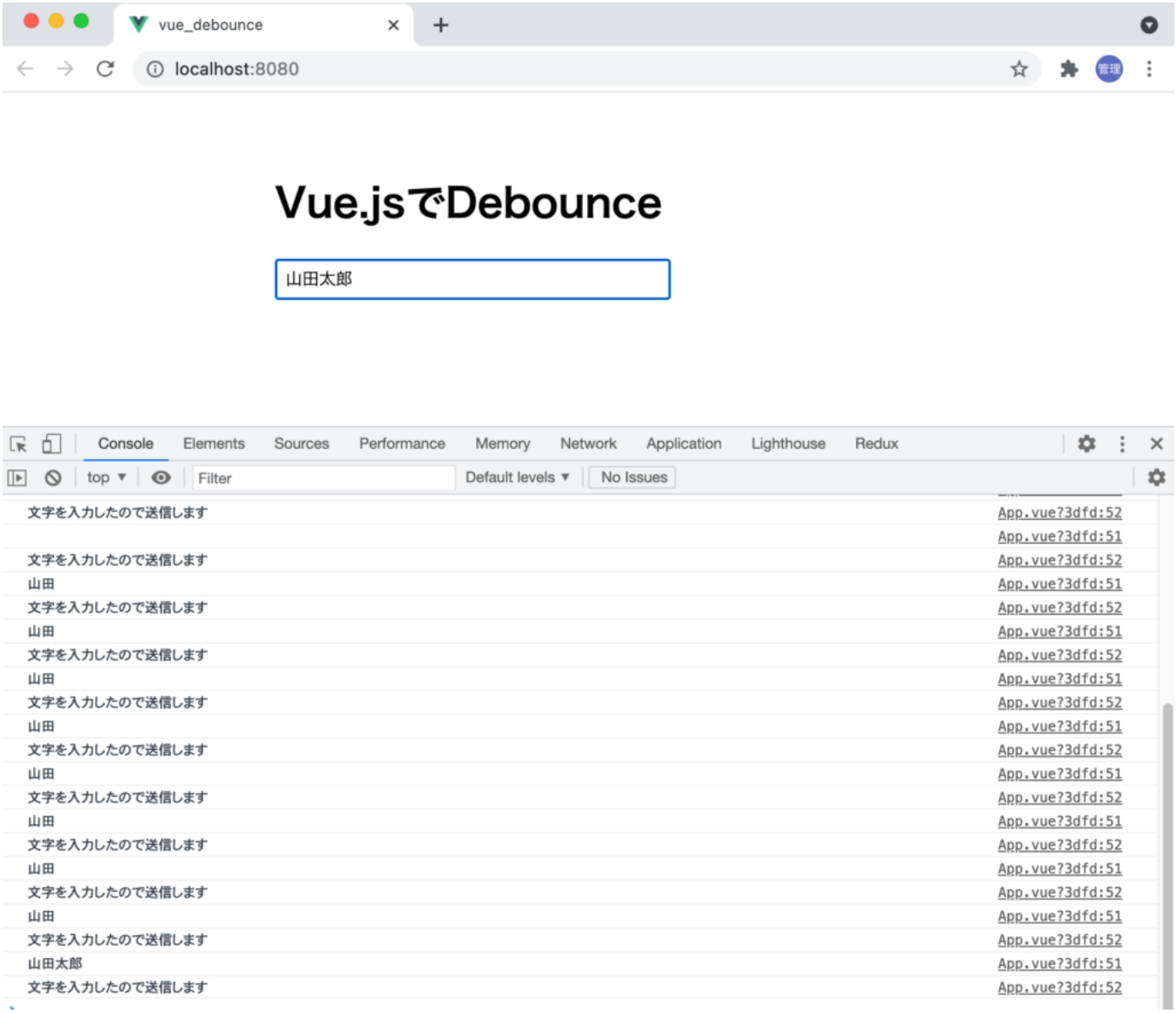
```
      sendData() {
        console.log(this.data);
        console.log('文字を入力したので送信します');
      },
      handleChange: function() {
        this.sendData();
      },
    },
  };
  </script>
```

データプロパティのdataを設定し、input要素にv-modelを設定しています。inputイベントを設定しているので文字を入力するたびにhandleChangeメソッドが実行されます。handleChangeメソッドの中ではsendDataメソッドが実行されます。

Vue.jsのchangeイベントを利用するとinput要素に入力を行いカーソルを外した際にイベントが発生するのでinputイベントを利用しています。inputイベントを利用しても日本語の変換前のアルファベットは表示されません。



動作確認を行うとdebounceを設定していないのでJavaScriptでdebounceを利用しない時と同じようにメッセージが複数回表示されます。JavaScript, Reactの時と表示されるメッセージは異なりますがかなりの数のメッセージが表示されます。



debounceを設定しない場合

debounceを設定した場合

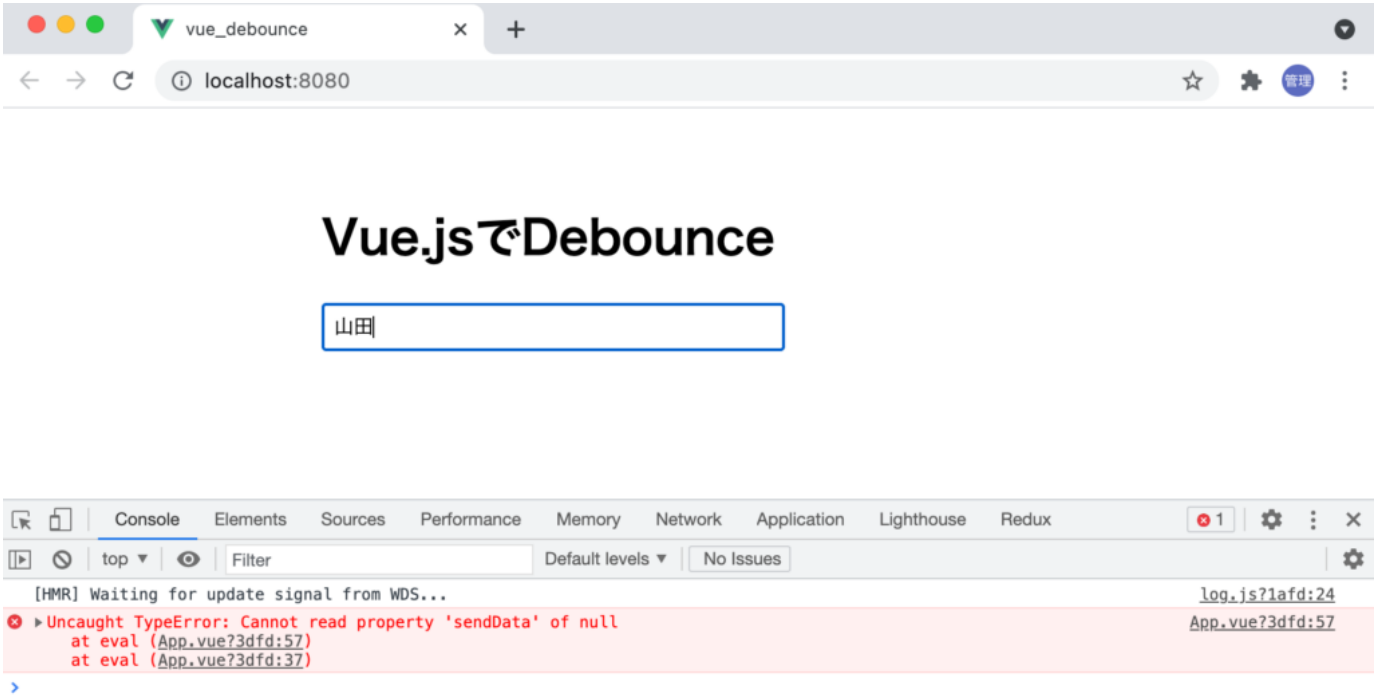
JavaSciptの時に利用したdebounceコードをそのまま利用します。debounceのコードに変更はないのでdebounceのコードに関する説明は省略します。(後ほど変更が必要になります)

Vue.jsでは追加したdebounceでhandleChangeメソッドで設定していた関数を包みます。

```
<template>
  <div style="width: 500px; margin: 4em auto">
    <h1>Vue.jsでDebounce</h1>
```

```
      <input
        type="text"
        placeholder="名前を入力してください。"
        v-model="data"
        style="padding: 0.5em; width: 300px"
        id="name"
        @input="handleChange"
      />
    </div>
  </template>

  <script>
  const debounce = (func, wait = 1000) => {
    let timerId;
    return (...args) => {
      if (timerId) {
        clearTimeout(timerId);
      }
      timerId = setTimeout(() => {
        func.apply(null, args);
      }, wait);
    };
  };

  export default {
    data() {
      return {
        data: '',
      };
    },
    methods: {
      sendData() {
        console.log(this.data);
        console.log('文字を入力したので送信します');
      },
      handleChange: debounce(function() {
        this.sendData();
      }, 1000),
    },
  };
  </script>
```

動作確認を行うと”Uncaught TypeError: Cannot read property ‘sendData’ of null”エラーが発生します。



cannot read property エラー発生

エラーからsetDataが実行できないことがわかります。現在のdebounceのコードではthisが取得できていないことが原因です。thisにはVue インスタンスが含まれる必要があります。

この問題に対応するためにdebounce関数のreturnで実行しているアロー関数を標準関数に変更し、func.applyで設定しているnullをthisに変更します。標準関数とアロー関数でのthisの取り扱いの違いを理解する必要があります。

```
const debounce = (func, wait = 1000) => {
  let timerId;
  return function(...args){
    if (timerId) {
      clearTimeout(timerId);
    }
    timerId = setTimeout(() => {
      func.apply(this, args);
    }, wait);
  };
};
```

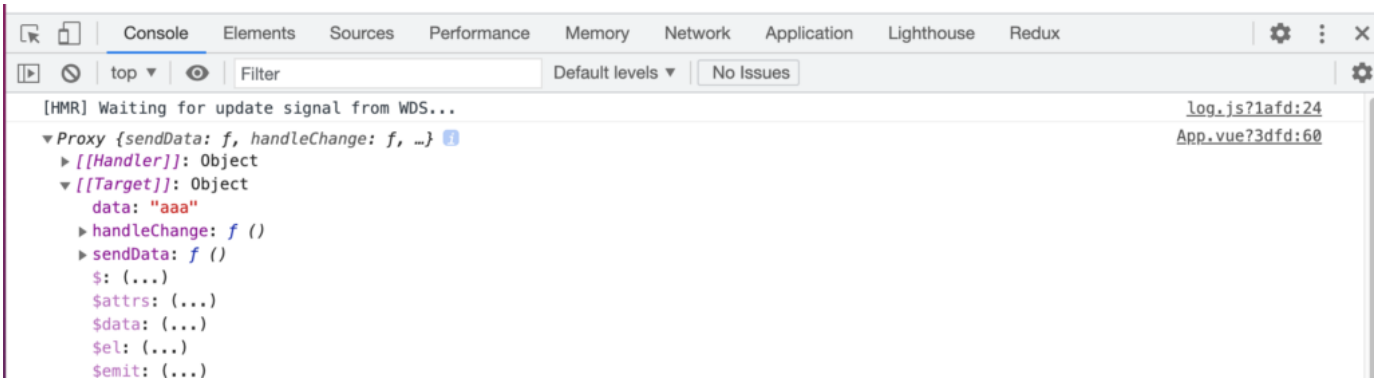
あわせて読みたい

【初心者向け】JavaScriptのbindって何？？を理解する(call, appyも一緒に)

設定後debounce関数に設定した関数でthisが正しく取得しているか確認するために console.log(this)を設定します。

```
handleChange: debounce(function() {
  // this.sendData();
  console.log(this);
}),
```

input要素に文字を入れてデベロッパーツールのコンソールに以下のproxyが表示されれば Vueインスタンスが取得できているので問題ありません。中にはデータプロパティも handleChange, sendDataメソッドも確認できます。

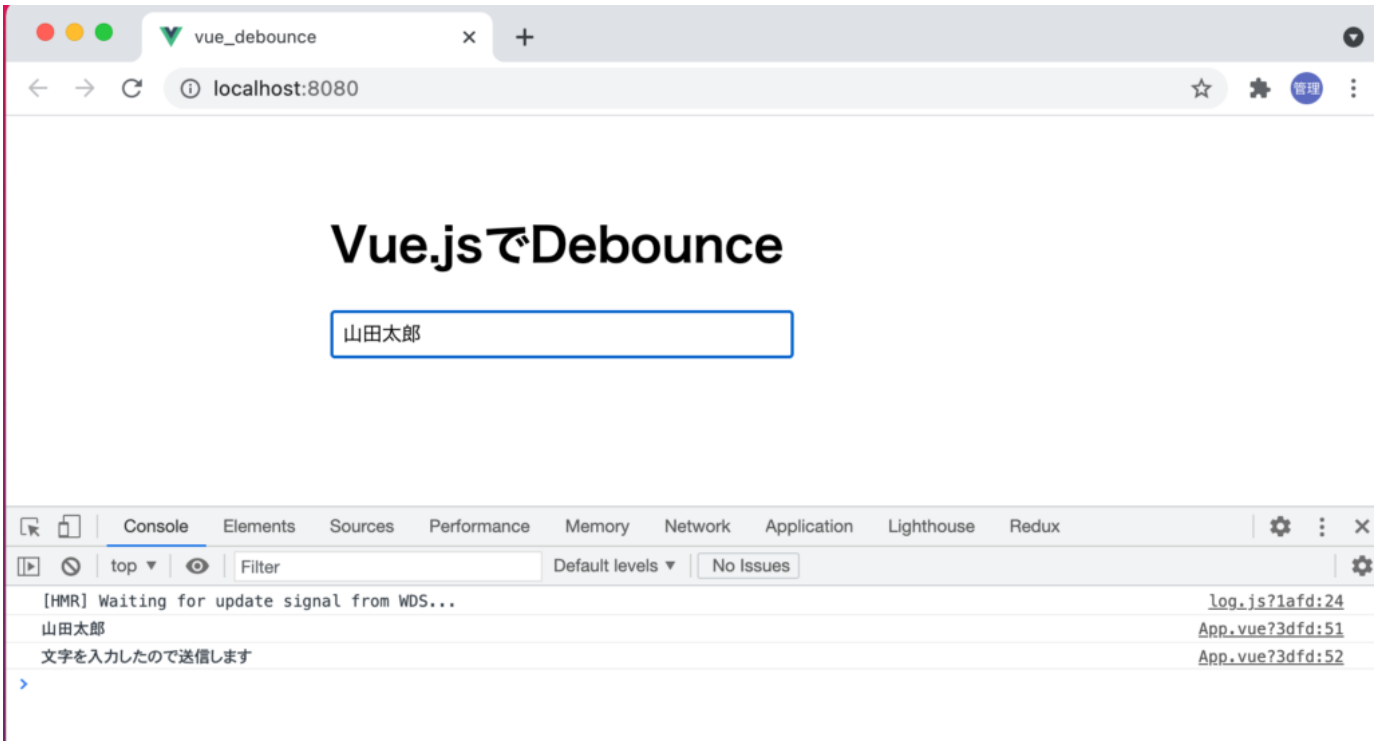


Vueインスタンスの確認

再度this.sendDataに戻してdebounceが動作することを確認してください。

```
handleChange: debounce(function() {
  this.sendData();
}),
```

山田太郎を入力してしばらくするとコンソールに山田太郎が表示されます。Vue.jsでもdebounceが動作することを確認することができました。



Vue.jsでdebounceが正しく動作

lodashのdebounceを利用した場合

Vue.jsの場合はlodashのインストールを行う必要があります。

```
% npm install lodash
```

インストールしたlodashのdebounceをインストールするだけであとは何も変わりません。lodashのdebounceの場合は時間を設定しない場合は0msとなり文字入力と同時に実行されるので適切な時間を設定する必要があります。

```
<script>
import { debounce } from 'lodash';
```

```
export default {
  data() {
    return {
      data: '',
    };
  },
  methods: {
    sendData() {
      console.log(this.data);
      console.log('文字を入力したので送信します');
    },
    handleChange: debounce(function() {
      this.sendData();
    },1000),
  },
};
</script>
```

ここまで読み進めてもらえればdebounceの理解も深まったのではないでしょうか。実際に利用するときにはlodashなどのライブラリの利用するこになるとは思いますが実際の仕組みが理解できていることは重要だと思いますのでぜひこの知識を役立ててください。