Peter Kim Follow

May 25, 2017 · 4 min read · Listen

Controlled and Uncontrolled Input Values in React

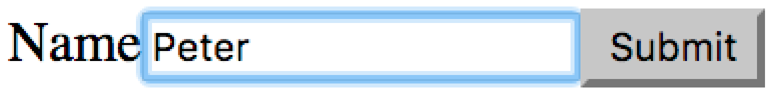
Forms play an important role everywhere you go: doctor's office, post office, work office, internet office, you name it. They contain information that is meant to be transferred between two parties. Many times, without a form, information cannot be stored and referred to.

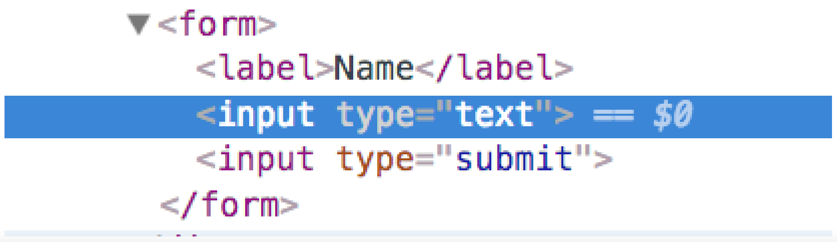
With React, there are two input value options that you can go with when you deal with

With uncontrolled input g of any states. What you submit is what you get.

```
class Form extends React.Component{

  formHandler(event){
  event.preventDefault();
  const name = this.input.value
  //do something with "name"

  }

  render(){
    return(
      <div >
        <form onSubmit={this.formHandler.bind(this)}>
          <label>Name</label>
            <input type='text' ref={(userInput) => this.input =
              userInput}/>
            <input type='submit'/>
        </form>
      </div>
    )
  }
}
```

The "ref" that we are using here is a special attribute provided by the Component — it takes a callback and when used on an HTML element, the argument it takes is the underlying DOM element (which in this case is an input). Filling this form out with the name "Peter":





```
▼<form>
   <label>Name</label>
   <input type="text"> == $0
   <input type="submit">
 </form>
```

> *Quick Note: With uncont {value here}) inside the input element will override any values that are typed in so in order to work around that, React provides another attribute called defaultValue (defaultValue={value here}) that will pre-populate the input field with the defaultValue without overriding any value input by the user.*

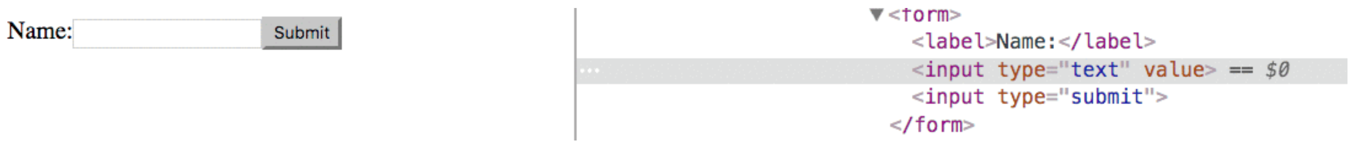
This is a very basic and the most simple way of setting up a form. Although it may be quicker to set this up, it limits the potential of React. We will get into that a little later.

Controlled

With controlled inputs, there is always some sort of change going on. The value of the input field will almost always be a prop of the class component (usually a state). Going along with this, there will also be a callback that is necessary to handle the changes going on with the input field.

```
class Form extends React.component{
 constructor(){
   super()
    this.state={
     name: ""
    }
 }

 nameChange(event){
  const inputName = event.target.value
   this.setState({
     name: inputName
   })
 }

 render(){
   return(
    <div>
     <form>
       <label>Name:</label>
       <input type="text" value={this.state.name} onChange=
       {this.nameChange.bind(this)}/>
       <input type="submit"/>
     </form>
    </div>
   )
```

Here, we have the comp nput field. The onChange handles the change in the input field and immediately updates the state of the component to the input of the user.



```
▼<form>
   <label>Name:</label>
   <input type="text" value> == $0
   <input type="submit">
 </form>
```

This is what makes it a controlled value.

When to use uncontrolled values vs controlled values

Uncontrolled values are useful when the form is very basic with minimalistic features. They are also good options when dealing with libraries or other languages that don't interact well with or don't use React. The downside of using uncontrolled values is the fact that there is not much you can do with it — making it quite limited in functionality.

Controlled values are useful for things such as validation or instant user feedback. You can imagine writing an error code that alerts the user that the input from the user is invalid or doesn't meet certain criteria (passwords for instance) which disappears once it is valid. The downside to using controlled values is that it requires much more code and there is a need to take care of the changes using callbacks. On top of that, every callback that you pass it through, needs to be a React component to process the changes. It's possible to see how changing your pre-existing code framework to React would make this a tedious task.

There isn't a ***right*** or ***wrong*** way to use these features. It all comes down to how the architect will use these forms and what will happen to the information that is being passed along. The design can get complicated when using controlled values but that allows for more functionality and a better user experience.

References:
A blog on controlled and uncontrolled form inputs
React uncontrolled components and React controlled components