React Components communication

Different patterns for communicating between react components

Our goals

- Learn different strategies and patterns to communicate between react components

Props - Communication from parent to child

- Props are used to communicate from parent to child by passing data from parent to child
- In the child the props are read only
- you can specify which props you expect to get in the child with the static propTypes
- you can add static defaultProps if you want to specify a default value for the props in the child

Props - Communication from child to parent - EX

- By using the props the parent can pass the child a callback
- the child can call that callback and pass a value back to the parent
- EX: Create a parent child components and pass value and callback to the child
- The child will have a button and will call the callback passing a random number
- The parent will display that number

Props - Communication from child to parent - EX

- By using the props the parent can pass the child a callback
- the child can call that callback and pass a value back to the parent
- EX: Create a parent child components and pass value and callback to the child
- The child will have a button and will call the callback passing a random number
- The parent will display that number

Brothers communication - lifting state - EX

- lifting to the state of the common ancestor is a technique to communicate between brothers
- Create parent component with two children under him: child1, child2
- The goal of the ex is to pass message from child1 to child2 and from child2 back to child 1
- each of those components will contain a form with text input and button to pass message to the other child
- Each of those child will contain list of previous messages that was sent to him.
- We pass the messages by using the state of the parent

Context - passing subtree data

- For our current communication problem we will have a parent, child, and grandchild components
- The parent would like to pass data to the grandchild
- One way to do this with the tools we learned is pass the props from the parent to the child and then pass the props from the child to the grandchild, but what if our tree is more complex than that
- Using context we can share data for part of our tree of components
- It is easy to abuse this feature and create global data which will prevent reuse of components, avoid this when other communications are possible

Context - passing subtree data

- For our current communication problem we will have a parent, child, and grandchild components
- The parent would like to pass data to the grandchild
- One way to do this with the tools we learned is pass the props from the parent to the child and then pass the props from the child to the grandchild, but what if our tree is more complex than that
- Using context we can share data for part of our tree of components
- It is easy to abuse this feature and create global data which will prevent reuse of components, avoid this when other communications are possible
- We use this when we have data shared by many components

Context - createContext

- You create the context using
  - React.createContext(defaultValue)
- You wrap the tree you want to provide the context to using
  - <MyContext.Provider value={/* not the default value*/}>

    …

    </MyContext.Provider>
- class components will be able to access the context using
  - this.context
  - Will only be available on components that define static contextType
- function component need to use MyContext.Consumer with the render prop pattern

Context -Changing context - EX

- You change the context by passing a function from the component that started the context, and the value will be in the state
- Our toggle function will be called and change the state which will cause the context to change in all the children.
- EX: create parent, child, grand, grandgrand components
  - the parent will pass a context with message value that will be displayed in all children
  - The parent will also pass a function to change the message
  - the grandgrand will contain a form with text input to change the message

Communication remote siblings

- There are times when we have an elaborate tree of components and we want to communicate between 2 siblings that are far from each other in the tree
- state lifting will not do the trick cause the common parent is really high up the tree
- Context is not the proper solution here as well since that data is not shared by many components
- In those cases we can use 2 approaches for communication
  - Redux - we will cover in later chapter
  - Using a service

Communication with a service - EX

- In order for us to use a service as means of communication we will have to use some sort of library that provides us with way to push data to listeners
- There are also common API's in the browser (CustomEvent) and in node (EventEmitter) but we would like to use a universal API
- RXJS is a library that solves async events using the Observable pattern
- EX:
  - Create a singleton service and using RXJS create an instance of a subject that will pass a message
  - using that service and subject, pass data between 2 sibling components

Summary

- In this lesson by using communication between components we covered some useful extra react patterns we can use
- Props and props callback we will use to communicate between parent and child
- state lifting to the parent will help us communicate between 2 siblings using their parent
- Context will share data between many components
- render prop that we learned before can also be used to communicate between parent and distant child
- RXJS can also push data to distant components