Scroll Infinito

Para deixar o scroll infinito é usado o intersection observer API. Permite observar mudanças na visibilidade de um elemento em relação a um contêiner pai ou a viewport do navegador, a janela da pagina.

Para utilizar o observer é preciso criar uma constante declarando ele:
`const observer = new IntersectionObserver(callback, options)`

O observer recebe dois parametros, uma função callback que vai executar toda vez / quando o elemento alvo aparecer no elemento pai / na tela e um objeto options contendo a configuração do callback.

Callback: A função callback recebe dois parâmetros. O entries, que é uma lista de objetos que contem dados do elemento observado, cada item do array representa um elemento observado e contem informações como:

- **entry.target:** O elemento DOM observado
- **entry.isIntersecting:** Se esta visivel na viewport (ou root)
- **entry.intersectionRatio:** Porcentagem visivel
- **entry.boundingClientRect:** Posição e tamanho do elemento
- **entry.intersectionRect:** Parte visivel
- **entry.rootBounds:** Tamanho da root (geralmente a viewport)
- **entry.time:** Timestamp da interseção

O entry representa os elementos, mas quando tem apenas 1 elemento sendo observado ele representa o primeiro item:
`function callback([entry]) {}`

O segundo parâmetro é o observer, que é o próprio `const observer` que foi declarado anteriormente.

Na função callback é preciso verificar se o elemento apareceu na tela com o if, usando o entry.isIntersecting, se for true é preciso desconectar o observer, ou seja, fazer o elemento não ser mais observado, se não o callback vai ficar executando toda vez que sair e entrar de novo:

```
function callback([entry], observer) {
    if(entry.isIntersecting) {
        observer.disconnect()
    }
}
```

Além de desconectar, é aqui que a função que vai fazer aparecer mais elementos é adicionada (em baixo do disconnect). Por exemplo, é aqui que é adicionada / atualizada uma lista, um fetch é feito ou os dois, etc.

Options: O options é um objeto e possui 3 parâmetros principais de configuração:

root: Recebe como valor o contêiner que vai englobar o elemento observado, o elemento pai que possui a barra de rolagem. Geralmente é o próprio document, ou seja, a própria pagina. Para selecionar a própria pagina é colocado o valor de null.

rootMargin: É a distancia a mais ou a menos que o callback será disparado quando entrar em interseção com o root. É como um margin, quando esse margin aparece na tela a função callback é executada. É basicamente a área que executa o callback

threshold: Define a proporção da área de interseção, ou seja, quanto o elemento terá que aparecer para o callback ser executado. É um valor

percentual que vai de 0 a 1 (0% a 100%), sendo que 1 (100%) é o elemento todo e 0.5 (50%) é metade do elemento.

Configuração padrão do options:
`options = {root: null, rootMargin: "0px", threshold: 1.0}`

useEffect: É preciso usar o useEffect porque o intersection observer precisa ser criado, configurado e destruído corretamente, o useEffect é o lugar ideal para isso porque roda quando o componente é montado, aparece na tela e pode limpar (desconectar) quando o componente desmonta. Resumindo, ele é bom porque permite executar a logica dentro dele quando a pagina carrega e no final permite retornar um disconnect, ou seja, o useEffect é um lugar controlado:

```
useEffect(() => {
    const options = {}

    function callback() {}

    const observer = new IntersectionObserver(callback, options)

    observer.observe(elementRef.current)

    return () => observer.disconnect()
}, [])
```

useRef: Faz referencia a um elemento
`const containerRef = useRef()` ← retorna o elemento h1

```
<h1 ref={containerRef}>Titulo</h1>
```

useRef Dinamico: É possível ir adicionando elementos e referenciá-los de forma dinâmica. O segredo é o {} dentro do useRef e a função no parametro ref={}, que permite passar a função como parametro para o componente e retornar o parametro desse valor, que é a img no componente Img.jsx:

App.jsx

```
const imgRefs = useRef({})

<Img key={index} src={src} ref={el => imgRefs.current[index] =
el} />
```

Img.jsx

```
function Img({ src, ref } ) {
    return (
        <>
            <img src={src} alt="img" ref={ref} />
        </>
    )
}
```

Pegar o ultimo elemento referenciado:

```
 const lastImg = imgRefs.current[srcList.length - 1]

if(lastImg) {
    observer.observe(lastImg)
}
```

O código varia muito quando se fala de renderizar elementos infinitos, ou muitos elementos na tela, mas a base é pegar um elemento e botar um observador nele, para quando ele aparecer na janela algo ser executado (um callback), é esse callback que faz outro algo aparecer, só que esse algo precisa ter um observador para aparecer o 2° algo quando o 1°

aparecer na janela. Assim por diante. O algo que aparece sempre tem que ter um observador.

Os elementos que vão aparecendo geralmente são resultado de um fetch ou de uma lista.

Código Basico

```
useEffect(() => {
    const options = {root: null, rootMargin: "0px", threshold: 1.0}

    function callback() {
        if(entry.isIntersecting) {
            observer.disconnect()
            getNewImgSrc() ← Função que adiciona itens em uma lista
        }
    }

    observer = new IntersectionObserver(callback, options)
    observer.observe(titleRef.current)

    return () => observer.disconnect()
}, [])

return (
    <h1 ref={titleRef}>Titulo</h1>
)
```

Exemplo

```
import Img from './assets/Img'
import { useRef, useEffect, useState } from 'react'

function App() {
  const [srcList, setSrcList] = useState([])
```

```
const titleRef = useRef()
const imgRefs = useRef({})

function getNewImgSrc() {
  fetch(`https://api.dicebear.com/9.x/pixel-art/svg?seed=${new Date().getTime()}`)
  .then(resp => resp.blob())
  .then((blob) => {
    if(srcList.length > 0) {
      setSrcList([...srcList, URL.createObjectURL(blob)])
    }

    else {
      setSrcList([URL.createObjectURL(blob)])
    }
  })
  .catch(err => console.log(err))
}

useEffect(() => {
  const options = {root: null, rootMargin: "0px", threshold: 1.0}

  function callback([entry], observer) {
    if(entry.isIntersecting) {
      observer.disconnect()
      getNewImgSrc()
    }
  }

  const observer = new IntersectionObserver(callback, options)

  observer.observe(titleRef.current)

  return () => observer.disconnect()
}, [])

useEffect(() => {
  const observer = new IntersectionObserver(([entry]) => {
    if(entry.isIntersecting) {
      observer.disconnect()
      getNewImgSrc()
    }
  }, {root: null, rootMargin: "0px", threshold: 1.0})
```

```
    const lastImg = imgRefs.current[srcList.length - 1]

    if(lastImg) {
      observer.observe(lastImg)
    }

    return () => observer.disconnect()
  }, [srcList])

  return (
    <div className='container'>
      <h1>Role para ver as imagens</h1>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>
      <p>&darr;</p>

      <div>
        <h1 ref={titleRef}>Imagens</h1>

        {
          srcList.length > 0 &&
            srcList.map((src, index) => (
            <Img key={index} src={src} ref={el => imgRefs.current[index] = el} />
          ))
        }
      </div>
    </div>
  )
}

export default App
```