Utility APIs

The following APIs don't have one-to-one equivalents in a real user interaction.
Their behavior is therefore an interpretation how the "perceived" user interaction might be translated to actual events on the DOM.

clear()

```
clear(element: Element): Promise<void>
```

This API can be used to easily clear an editable element.

1. Focus element
2. Select all contents as per browser menu
3. Delete contents as per browser menu

```
test('clear', async () => {
  render(<textarea defaultValue="Hello, World!" />)

  await userEvent.clear(screen.getByRole('textbox'))

  expect(screen.getByRole('textbox')).toHaveValue('')
})
```

The `Promise` is rejected if the element can not be focused or contents can not be selected.

selectOptions(), deselectOptions()

```
selectOptions(
    element: Element,
    values: HTMLElement | HTMLElement[] | string[] | string,
): Promise<void>
deselectOptions(
    element: Element,
    values: HTMLElement | HTMLElement[] | string[] | string,
): Promise<void>
```

```
test('selectOptions', async () => {
  render(
    <select multiple>
      <option value="1">A</option>
      <option value="2">B</option>
      <option value="3">C</option>
    </select>,
  )

  await userEvent.selectOptions(screen.getByRole('listbox'), ['1',
'C'])

  expect(screen.getByRole('option', {name:
'A'}).selected).toBe(true)
  expect(screen.getByRole('option', {name:
'B'}).selected).toBe(false)
  expect(screen.getByRole('option', {name:
'C'}).selected).toBe(true)
})
```

```
test('deselectOptions', async () => {
  render(
    <select multiple>
      <option value="1">A</option>
      <option value="2" selected>
        B
      </option>
      <option value="3">C</option>
    </select>,
  )
```

```
  await userEvent.deselectOptions(screen.getByRole('listbox'), '2')

  expect(screen.getByText('B').selected).toBe(false)
})
```

Note that this API triggers pointer events and is therefore subject to pointerEventsCheck.

type()

```
type(
    element: Element,
    text: KeyboardInput,
    options?: {
        skipClick?: boolean
        skipAutoClose?: boolean
        initialSelectionStart?: number
        initialSelectionEnd?: number
    }
): Promise<void>
```

Type into an input element.

> You should use `keyboard()` if you want to just simulate pressing buttons on the keyboard.
> You can use `type()` if you just want to conveniently insert some text into an input field or textarea.

1. Unless `skipClick` is `true`, click the element.
2. If `initialSelectionStart` is set, set the selection on the element. If `initialSelectionEnd` is not set, this results in a collapsed selection.
3. Type the given `text` per `keyboard()`.
4. Unless `skipAutoClose` is `true`, release all pressed keys.

```
test('type into an input field', async () => {
  render(<input defaultValue="Hello," />)
```

```
upload(
    element: HTMLElement,
    fileOrFiles: File | File[],
): Promise<void>
```

```
})
```

Change a file input as if a user clicked it and selected files in the resulting file upload dialog.

> Files that don't match an `accept` property will be automatically discarded, unless `applyAccept` is set to `false`.

```
test('upload file', async () => {
  render(
    <div>
      <label htmlFor="file-uploader">Upload file:</label>
      <input id="file-uploader" type="file" />
    </div>,
  )
  const file = new File(['hello'], 'hello.png', {type:
'image/png'})
  const input = screen.getByLabelText(/upload file/i)

  await userEvent.upload(input, file)

  expect(input.files[0]).toBe(file)
  expect(input.files.item(0)).toBe(file)
  expect(input.files).toHaveLength(1)
})

test('upload multiple files', async () => {
  render(
    <div>
      <label htmlFor="file-uploader">Upload file:</label>
      <input id="file-uploader" type="file" multiple />
    </div>,
  )
```

```
  const files = [
    new File(['hello'], 'hello.png', {type: 'image/png'}),
    new File(['there'], 'there.png', {type: 'image/png'}),
  ]
  const input = screen.getByLabelText(/upload file/i)

  await userEvent.upload(input, files)

  expect(input.files).toHaveLength(2)
  expect(input.files[0]).toBe(files[0])
  expect(input.files[1]).toBe(files[1])
})
```

Edit this page

*Last updated on **Aug 29, 2022** by **Anton Khitrenovich***