How to write a react renderer

introduction of egreact

1 Idea

Why ?

uncomfortable

- ❌ 🔔 no tip
- ❌ ⚫ any hacks
- ❌ 📦 extra packaging
- ❌ 📖 need compiler

From the current front-end development perspective, exml's poor authoring experience and isolated ecology have lagged behind the mainstream.

advantage

- react is borned for ui
- react ecosystems, such as react-router, state manage libraries, hooks libraries and test libraries
- flexible between template and jsx
- egret native ?

A similar and mature open source libray :

react-three

App.js

```
1   import { useRef, useState } from 'react'
2   import { Canvas, useFrame } from '@react
3
4   function Box(props) {
5     // This reference gives us direct acce
6     const ref = useRef()
7     // Hold state for hovered and clicked
8     const [hovered, hover] = useState(fals
9     const [clicked, click] = useState(fals
10    // Subscribe this component to the ren
11    useFrame((state, delta) => (ref.curren
12    // Return the view, these are regular
```

Open Sandbox

Console 0 Problems 0 React DevTools 0

egreat

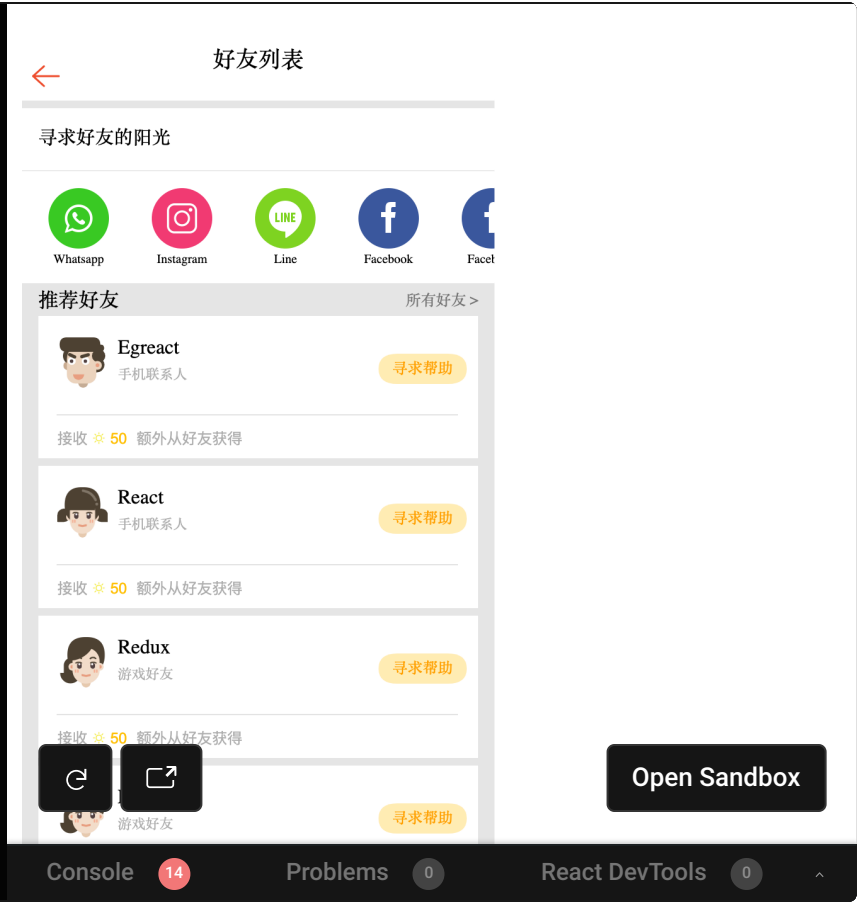
index.tsx

```
/**
* This is an auto-generated demo by dumi
* if you think it is not working as expe
* please report the issue at
* https://github.com/umijs/dumi/issues
**/

import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(
  <App />,
  document.getElementById('root'),
);
```



2 Implement

HostConfig

There is an interface named `HostConfig` to define a renderer

Think about react-dom: **createElement, appendChild, insertBefore and removeChild.**

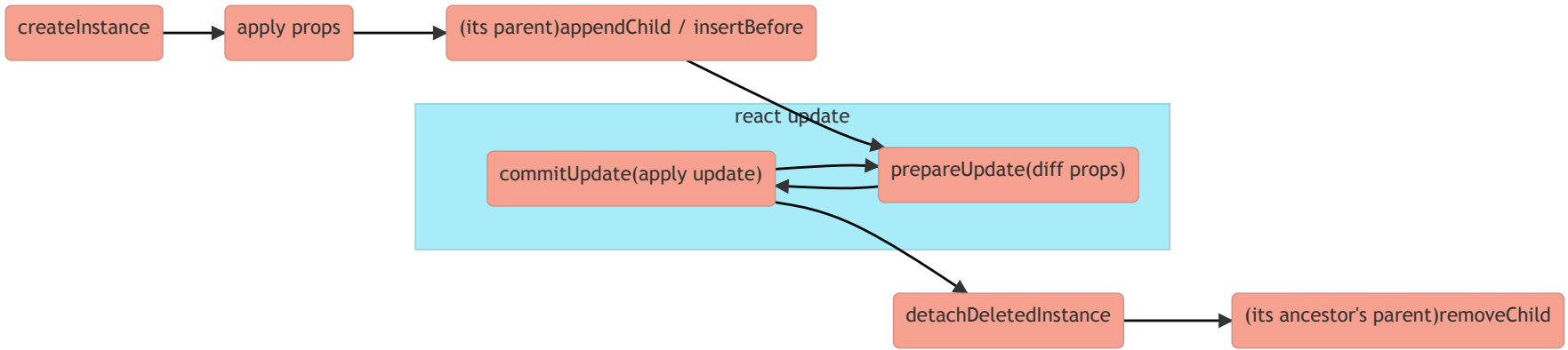
```
const HostConfig = {
  // create a host instance
  createInstance,

  // host actions
  appendChild,
  insertBefore,
  removeChild,
  detachDeletedInstance, // clean effects

  // props diff and apply
  prepareUpdate,
  commitUpdate,

  ...others
}
```

Lifecycle of a host instance



- diff props
- apply updates

Description for a host component

Expandsive Prop

```
interface IPropHandler{
    __Class: new ( ...args: any) => any, // constructor
    [key in string]: PropSetter, // function for updating
    [key in `__diff_${string}`]: DiffHandler // function for diff
}
```

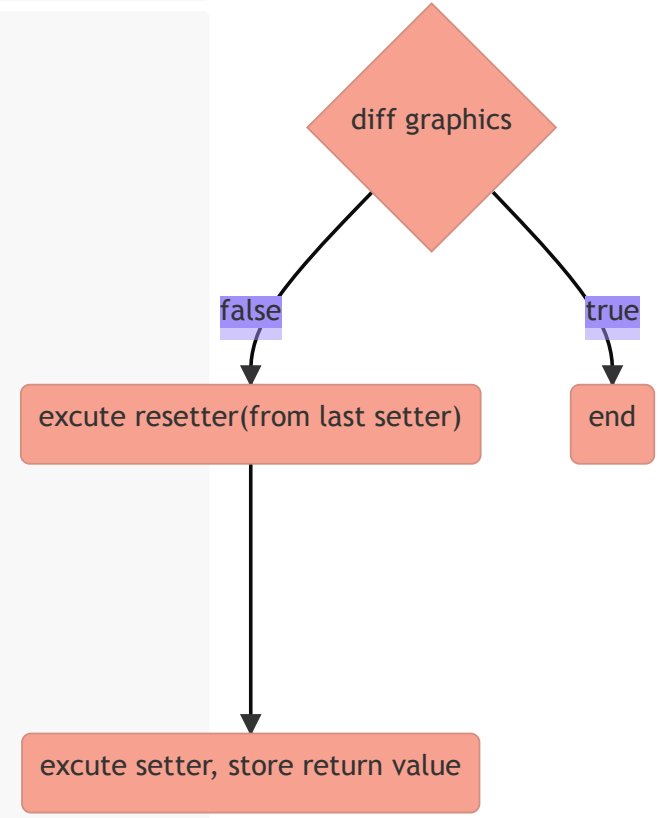
clear last prop setter effect, inspired by `useEffect`

```
type PropResetter = void | ((removed: boolean) => void)
type PropSetter = (args:any) => PropResetter
```

example

```
<shape graphics={[['beginFill', 0x000000],['drawRect', 0, 0, 300, 100],['endFill']]}></shape>
```

```
import { Setters } from 'egreact'
const shape = {
  ...Setters.egret.displayObject,
  __Class: egret.Shape,
  graphics: ({
    newValue,
    instance,
  }:{ newValue: ['string', ...any[]] | Function,instance: { graphics: egret.Graphics }}) => {
    if (is.arr(newValue)) {
      for (const action of newValue) {
        instance.graphics[action[0]](...action.slice(1))
      }
      return () => instance.graphics.clear()
    } else if (is.fun(newValue)) {
      return newValue(instance.graphics, instance)
    }
  },
  __diff_graphics: (np: any, op: any) => {
    if (is.arr(np) && is.arr(op)) {
      np = np.flat(1)
      op = op.flat(1)
      if (np.length !== op.length) return false
      for (let i = 0; i < np.length; i++) {
        if (np[i] !== op[i]) return false
      }
      return true
    } else return np === op
  }
}
```



example

1. declare jsx
2. call `extend` to let egreact add a host component

```
import { TransProp, extend } from 'egreact'
import shape from './shape'
declare global {
  namespace JSX {
    shape: TransProp<typeof shape>
  }
}
extend({
  Shape: shape
})
```

3 Accessibility

entry

- `<Egreact></Egreact>`

A component in react-dom context, `<Egreact>` will `runEgret`

```
import { Egreact, EgreactLink } from "egreact";
function App() {
  return
    (<div>
      <Egreact>
        <eui-group layout="vertical" layout-gap={20}>
          {/** egreact context ... **/}
        </eui-group>
      </Egreact>
    </div>)
}
```

entry

- `createEgreactRoot`

Writing react component and use it like a skin

```
import React from 'react'
import { createEgreactRoot } from 'egreact'
class EgreactRender extends egret.DisplayObjectContainer {
  root = createEgreactRoot(this)
  constructor(reactNode: React.ReactNode) {
    super()
    this.addEventListener(egret.Event.ADDED, () => this.root.render(reactNode), this)
    this.addEventListener(egret.Event.REMOVED, () => this.root.unmount(), this)
  }
}
const displayObjectContainer = new egret.displayObjectContainer();
displayObjectContainer.addChild(new EgreactRender(
    <sprite graphics={[['beginFill', 0x000000],['drawRect', 0, 0, 300, 100],['endFill']]}>
      <textField size={16}>Hello, egreact</textField>
    </sprite>
))
```

primitive

- `primitive`

insert egret instance/class in egreat context

```
<displayObjectContainer>
  <primitive
    object={container}
    key={container.$hashCode}
    onTouchTap={()=> setX((x) => x + 50)}
    x={x}>
    <eui-rect fillColor={0x888888} width={100} height={100}></eui-rect>
  </primitive>
  <primitive
    constrctor={ButtonSkin}
    borderRadius={50}
    strokeWeight={2}
    isStroke={true}
  />
</displayObjectContainer>
```

If sub component is a prop...

syntax sugar in exml

```
<e:Scroller>
    <e:Group/>
</e:Scroller>
```

original syntax

```
<e:Scroller> <e:viewport> <e:Group/> </e:viewport> </e:Scroller>
```

in egreact

```
<eui-scroller>
  <eui-group attach="viewport"/>
</eui-scroller>
```

use `attach` to change the way of adding from `scroller.addChild(group)` to `scroller.viewport = group`

If prop is an object...

```
const layout = new eui.VerticalLayout();
layout.gap = 10;
group.layout = layout;
```

exml

```
<e:Group>
  <e:layout>
    <e:VerticalLayout gap="10" />
  </e:layout>
</e:Group>
```

why a prop look like a sub component?

If prop is an object...

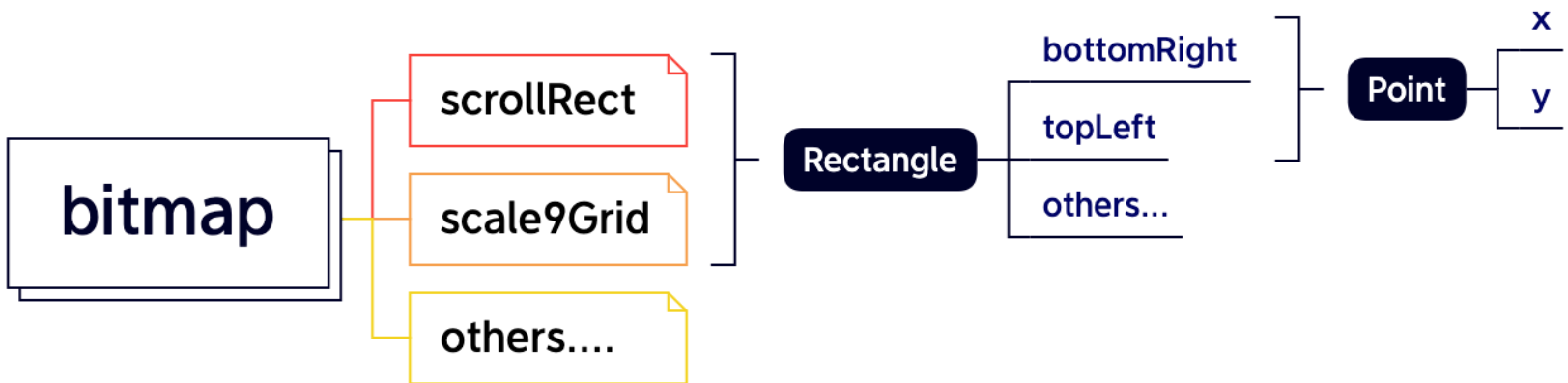
```
const layout = new eui.VerticalLayout();
layout.gap = 10;
group.layout = layout;
```

Prop should be described at tag...

```
<eui-group layout="vertical" layout-gap={10}></eui-group>
```

Same interface

Prop on different host components has the same interface.



interface prop

```
scrollRect
scrollRect-width
scrollRect-height
scrollRect-others ...
scrollRect-bottomRight
scrollRect-bottomRight-x
scrollRect-bottomRight-y
scrollRect-topLeft
scrollRect-topLeft-x
scrollRect-topLeft-y

scale9Grid
scale9Grid-width
scale9Grid-height
scale9Grid-others ...
scale9Grid-bottomRight
scale9Grid-bottomRight-x
scale9Grid-bottomRight-y
scale9Grid-topLeft
scale9Grid-topLeft-x
scale9Grid-topLeft-y
```

`${name}-${first}-${second}-${third} ...`

interface prop

```
const pointProp = {
  __Class: egret.Point,
  __setter: Point.setter,
  __diff: Point.diff,
  x: NormalProp.num,
  y: NormalProp.num,
}

const rectangleProp = {
  __Class: egret.Rectangle,
  __setter: Rectangle.setter,
  __diff: Rectangle.diff,
  bottomRight: pointProp,
  topLeft: pointProp,
  ...others
}
```

Generate props recursively

- what about the type?

interface prop

```typescript
// flat object, such as `{ a: { b: string }, b: number }` will be translated into `{ a-b: string, b: number }`

type ToUnionOfFunction<T> = T extends any ? (x: T) => any : never
// { a-b: string } | { b: number } => { a-b: string } & { b: number }
// { a-b: string, b: number }
type UnionToIntersection<T> = ToUnionOfFunction<T> extends (x: infer P) => any ? P : never

// { a: { b: string }, b: number } => { a-b: string } | { b: number }
type FlattenObjectToIntersection<T extends object, S extends string> = {
  [K in Exclude<keyof T, Symbol>]: T[K] extends object
    ? FlattenObjectToIntersection<T[K], `${S}${K}-`>
    : { [_ in `${S}${K}`]: T[K] }
}[Exclude<keyof T, Symbol>]

// { a: { b: string }, b: number } => { a-b: string, b: number }
type FlattenObject<T extends object> = UnionToIntersection<FlattenObjectToIntersection<T, ''>>
```

Think about `<eui-group layout-gap={10} layout="vertical"></eui-group>`

If `layout` update from `vertical` to `horizontal`, will effect `layout-gap` ?

prefix prop update effect

1. sort props by the length of keys to ensure the order.

   props: `['layout-gap', 'layout']` => `['layout', 'layout-gap']`

   changes: `[]`

2. push `layout` change to `changes`

   changes: `[] ⇒ [['layout', any change]]`

prefix prop update effect

when traverse at `layout-gap`

pre\sub	no change	update	remove
update	after	after	-/before
remove	-	after	-/before

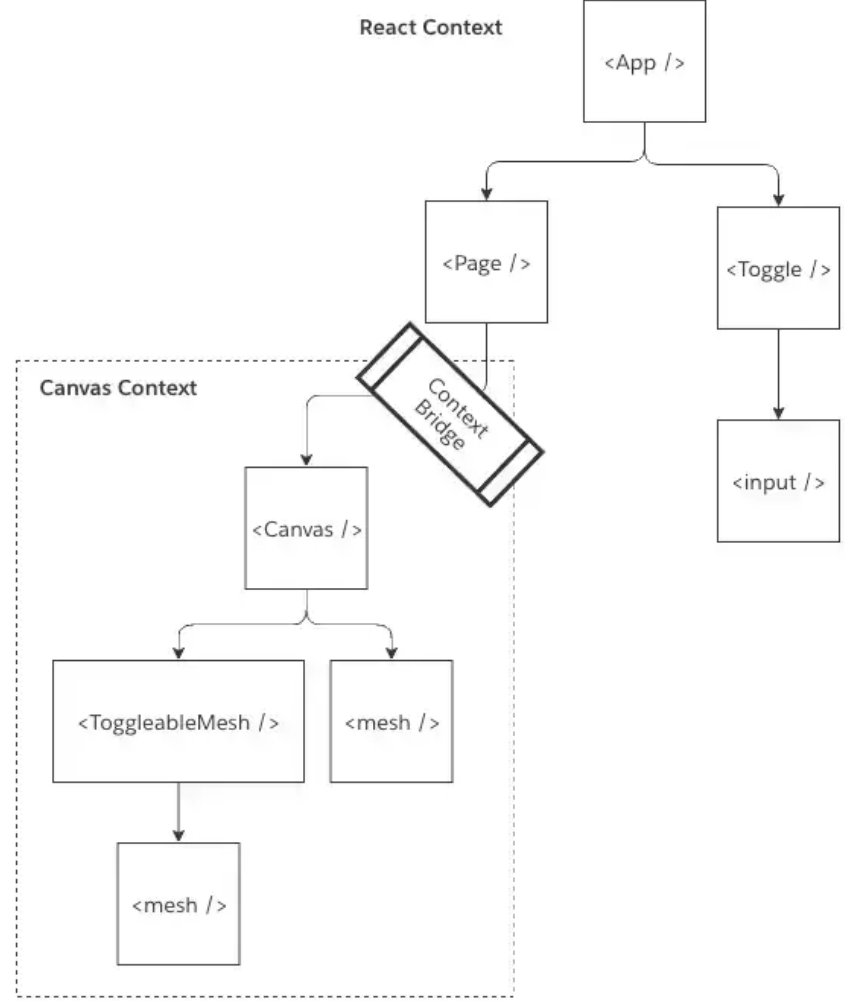
- when `sub` is `remove`, its change must be inserted before `pre`
- when `pre` is `update`, its `sub` must be applied again

If use contexts...

```
1   const App = () => {
2     const count = useSelector((state: RootState) => state.counter.value)
3     const dispatch = useDispatch()
4     return (
5       <div>
6         <h2 onClick={() => dispatch(increment())}>
7           i have been click {count} times!
8         </h2>
9         <ErrorBoundary>
10          <Egreact contextsFrom={false}>
11            <SubComponent />
12          </Egreact>
13        </ErrorBoundary>
14      </div>
15    )
16  }
17  const SubComponent = () => {
18    const dispatch = useDispatch()
19    return <sprite onTouchTap={() => dispatch(increment())} />
20  }
21  export default () =>(<Provider store={store}><App/></Provider>)
```

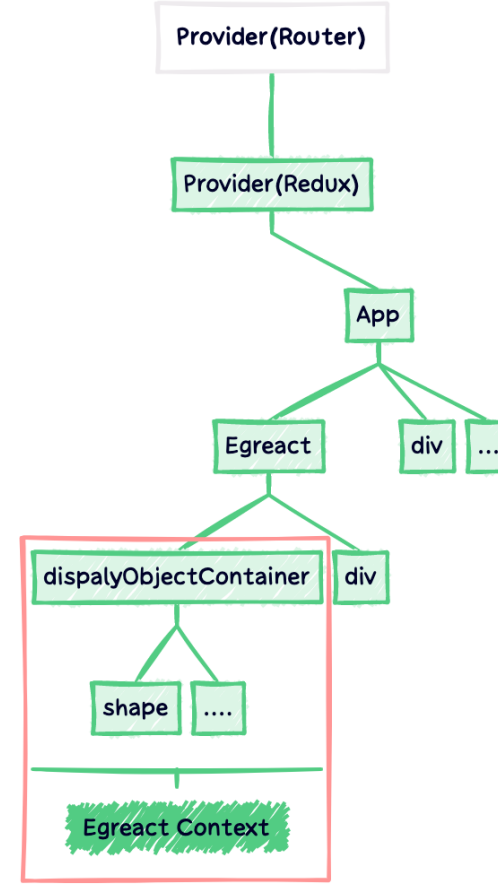
Context Bridge

1. collect contexts from fiber
2. collect values from contexts by useContext
3. generator Provider for new renderer



collect contexts from fiber

```
1   export function collectContextsFromDom(dom: HTMLElement) {
2     const fiberKey = Object.keys(dom).find(
3       (key) => key.startsWith('__react') && dom[key]?.stateNode === dom,
4     )
5     let fiber = dom[fiberKey]
6     const contexts: React.Context<any>[] = []
7     while (fiber) {
8       if (fiber.type?._context) {
9         contexts.push(fiber.type._context)
10      }
11      fiber = fiber.return
12    }
13    return contexts
14  }
```



context bridge

collect values from contexts by useContext

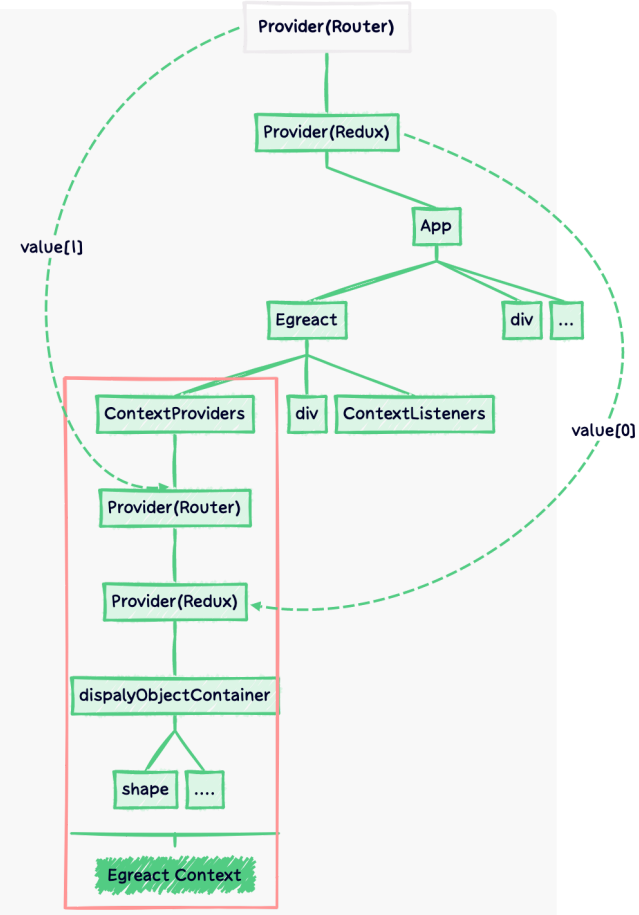
```
export const ContextListeners = memo(
  ({ contexts, values }: { contexts: React.Context<any>[]; values: CallBackArray }) => (
    <>
      {contexts.map((context, index) => (
        <ContextListener key={index} context={context} values={values} index={index} />
      ))}
    </>
  ),
)

export const ContextListener = memo(
  ({ context, values, index }: { context: React.Context<any>; values: CallBackArray; index: number }) => {
    values[index] = useContext(context)
    return null;
  },
)
```

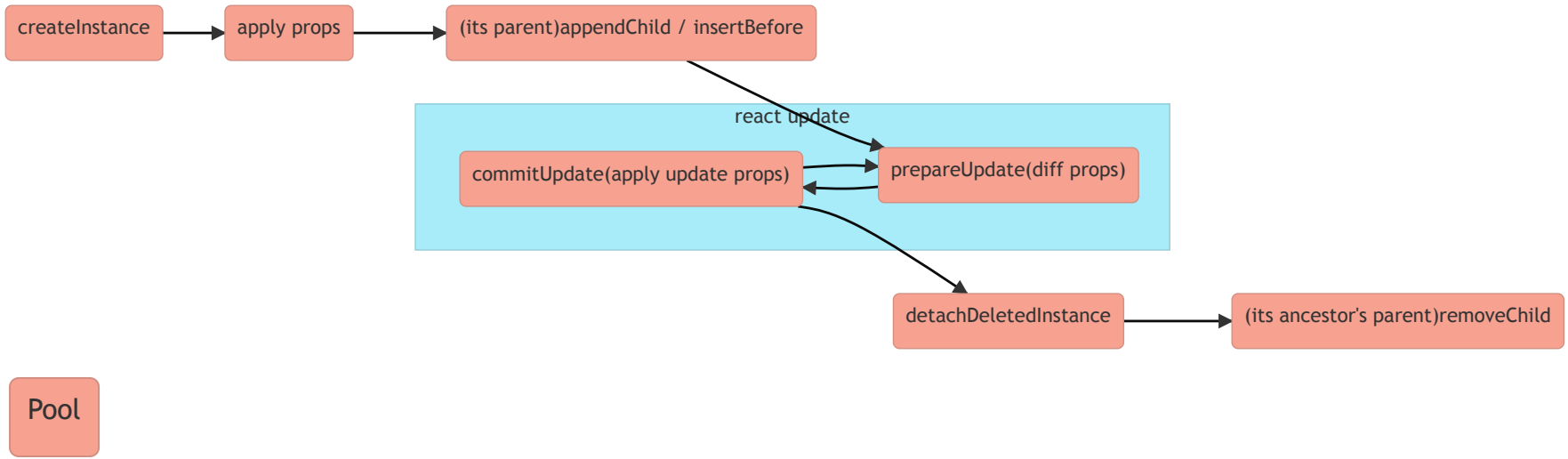
why not `contexts.map((context, index) => (values[index] = useContext(context))` in `ContextListeners` directly?

context bridge

```
1   export const ContextProviders = memo(
2     ({
3       contexts,
4       values,
5       children,
6     }: {
7       contexts: React.Context<any>[]
8       values: CallBackArray
9       children: React.ReactNode
10    }) => {
11      const [, update] = useState({})
12      useEffect(() => {
13        values.setCallback(() => update({}))
14        return () => values.setCallback(() => void 0)
15      }, [])
16
17      return contexts.reduce(
18        (child, Context, index) =>
19          (<Context.Provider value={values[index]}>
20            {child}
21          </Context.Provider>),
22        children,
23      ) as JSX.Element
24    },
25  )
```

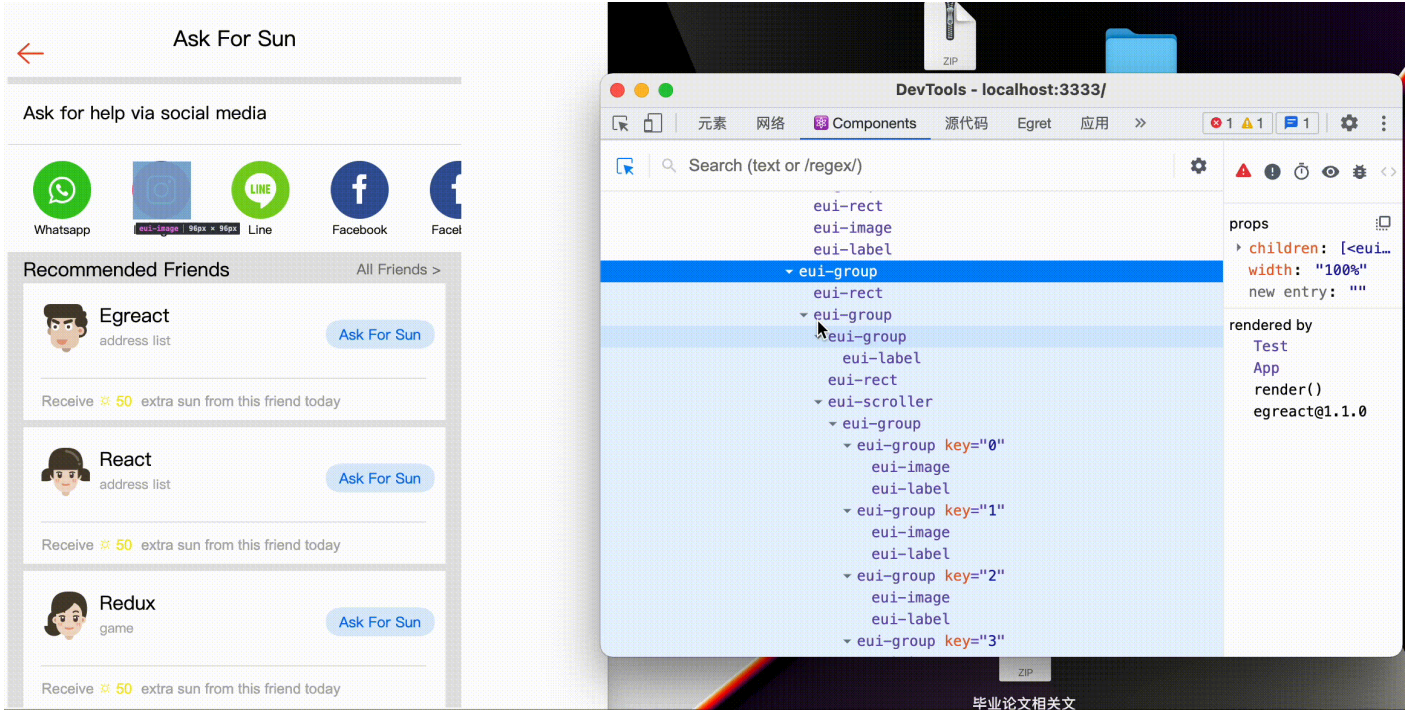


Pool


createInstance
apply props
(its parent)appendChild / insertBefore
react update
commitUpdate(apply update props)
prepareUpdate(diff props)
detachDeletedInstance
(its ancestor's parent)removeChild
Pool

react devtool

`reconciler.injectIntoDevTools`: view component tree



How picker implement?

1. listener `onMouseEnter` at window
2. get dom when emit envent
3. call `getComputedStyle` and dom methods, such as `getBoundingClientRect`, to computed the position and size of the shadow, then show shadow
4. call `findFiberByHostInstance`, next find corresponding node in component tree by fiber, finally jump to the position of the node

adapt

1. proxy `window.listener`
2. proxy `window.getComputedStyle`
3. mock dom attributes/methods is used in devtool

react devtool

1. judge is the mouse event point in canvas

```
function proxyHandler(e: MouseEvent){
  const { pageX: x, pageY: y } = e
  const r = document.querySelector('.egret-player > canvas').getBoundingClientRect()
  r.x += window.scrollX
  r.y += window.scrollY
  const isInCanvas = r.x > r.x && x < r.x + r.width && y > r.y && y < r.y + r.height
  ...
```

2. find the most suitable egret host instance

```
const scale = r.width / egret.lifecycle.stage.stageWidth
const target = findTargetByPosition(egret.lifecycle.stage, (x - r.x) / scale, (y - r.y) / scale)
```

3. mock a new event, change target from dom to egret instance

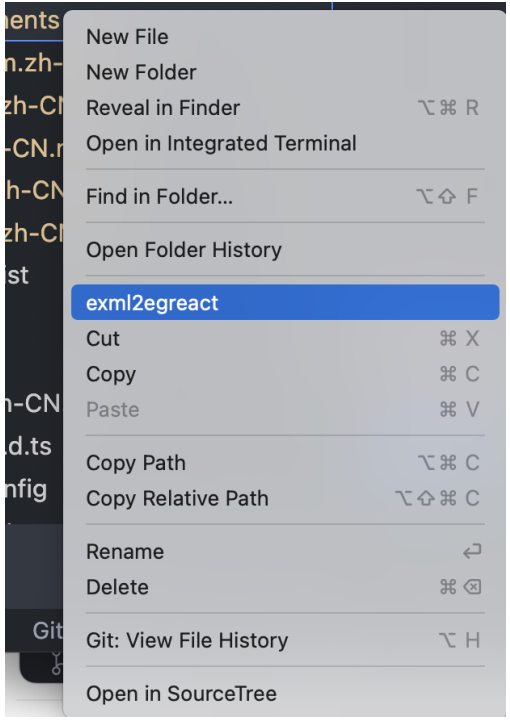
```
e = {
  ...e,
  preventDefault: e.preventDefault.bind(e),
  stopPropagation: e.stopPropagation.bind(e),
  target,
}
listener.call(this, e) // react devtool handler
```

react devtool

```
/**
 *
 * @copyright Copyright egret inject.
 * @description 寻找显示对象中符合舞台位置的最深的子显示对象（包含自身）
 */
export function findTargetByPosition(
  displayObject: egret.DisplayObject,
  stageX: number,
  stageY: number,
): egret.DisplayObject | null {
  if (!displayObject.visible) { return null }
  const matrix = displayObject.$getInvertedConcatenatedMatrix()
  const x = matrix.a * stageX + matrix.c * stageY + matrix.tx
  const y = matrix.b * stageX + matrix.d * stageY + matrix.ty
  const rect = displayObject.$scrollRect ? displayObject.$scrollRect : displayObject.$maskRect
  if (rect && !rect.contains(x, y)) { return null }
  if (this?.$mask && !displayObject.$mask.$hitTest(stageX, stageY)) { return null }
  const children = displayObject.$children
  let notTouchThrough = false
  if (children) {
    for (let index = children.length - 1; index >= 0; index--) {
      const child = children[index]
      if (child.$maskedObject) { continue }
      var target = findTargetByPosition(child, stageX, stageY)
      if (target && target.ispTouchThrough !== true) {
        notTouchThrough = true
        break
      }
    }
  }
  if (target) { return target }
  if (notTouchThrough) { return displayObject }
  return displayObject.$hitTest(stageX, stageY)
}
```

vscode plugin - exml2egreact

search in vscode plugin marketplace, then



exml2egreact

```
<?xml version="1.0" encoding="utf-8"?>
<e:Skin class="skins.RadioButtonSkin" states="up,down,disabled,upAndSelected,downAndSelected,disabledAndSelected"
    <e:Group width="100%" height="100%">
        <e:layout>
            <e:HorizontalLayout verticalAlign="middle"/>
        </e:layout>
        <e:Image fillMode="scale" alpha="1" alpha.disabled="0.5" alpha.down="0.7"
                 source="radiobutton_unselect_png"
                 source.upAndSelected="radiobutton_select_up_png"
                 source.downAndSelected="radiobutton_select_down_png"
                 source.disabledAndSelected="radiobutton_select_disabled_png"/>
        <e:Label id="labelDisplay" size="20" textColor="0×707070"
                 textAlign="center" verticalAlign="middle"
                 fontFamily="Tahoma"/>
    </e:Group>
</e:Skin>
```

exml2egreact

```
import React, { useRef, useState, useEffect } from "react";

export default function RadioButtonSkin({ context }) {
  const { currentState } = context;

  const labelDisplayRef = useRef<eui.Label>(null!);

  useEffect(() => {
    context.labelDisplay = labelDisplayRef.current;
  });

  return (
    <>
      <eui-group width="100%" height="100%" layout="horizontal">
        <eui-image
          fillMode="scale"
          alpha={{ disabled: 0.5, down: 0.7 }[currentState] ?? 1}
          source={
            {
              upAndSelected: "radiobutton_select_up_png",
              downAndSelected: "radiobutton_select_down_png",
              disabledAndSelected: "radiobutton_select_disabled_png"
            }[currentState] ?? "radiobutton_unselect_png"
          }
        />
        <eui-label
          size={20}
          textColor={0x707070}
          textAlign="center"
          verticalAlign="middle"
          fontFamily="Tahoma"
          ref={labelDisplayRef}
        />
      </eui-group>
    </>
  );
```

4 Engineering

Engineering

- 💼 pnpm monorepo
- 📦 rollup + tsc
- 🧪 jest + @testing-library/react (99%)
- 🎺 github action
- 📄 dumi (https://xingxinglieo.github.io/egreact/)

Q&A