1 产品 | 公有云上的分析利器
2 分析型 | 数据仓库的新
3 LakeHouse 范式的开拓者
4 开源 | Apache Doris
5 实践 | Apache Doris 在 Pusher 的实践
6 Apache SeaTunnel 在 Pusher 的实践
7 Apache SeaTunnel 的实践
8 Apache SeaTunnel 的实践
9 Apache SeaTunnel 的开发实践
10 Apache SeaTunnel 的开发实践
11 开源 | Apache Doris 的开源生态
12 开源 | Apache Doris 的开源生态
13 开源 | Doris 的开源生态实践

一、以前 Scrollable 组件

Scrollable 组件是用户一个可滚动的区域，当内容溢出容器时出现，同时这个区域支持 Viewport 视图的裁剪。此外，通过使用 ScrollCache 缓存已经渲染过的内容，性能表现更优秀。Viewport 是一个可视窗口，它决定了哪些内容会被渲染，哪些内容会被裁剪。

1. Scrollable 组件的内部实现

首先，Scrollable 基于 ScrollableWrapper 进行实现，这是用户日常使用的公共组件 (Public API)。

```
// Scrollable.tsx
export const Scrollable = memo(({ children, ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  const [viewportRect, setViewportRect] = useRef(null);
  // ...
  const [scrollX, setScrollX] = useRef(0);
  const [scrollY, setScrollY] = useRef(0);
  const [isScrollingX, setIsScrollingX] = useRef(false);
  const [isScrollingY, setIsScrollingY] = useRef(false);
  const [isDragging, setIsDragging] = useRef(false);
  const [isHoverX, setIsHoverX] = useRef(false);
  const [isHoverY, setIsHoverY] = useRef(false);
  const [isHoverScrollbarX, setIsHoverScrollbarX] = useRef(false);
  const [isHoverScrollbarY, setIsHoverScrollbarY] = useRef(false);
  // ...
  return <ScrollableWrapper ref={ref} />
})
```

然后，ScrollableWrapper 内部会分别实现 Viewport 和 ScrollCache 两个核心模块。

```
// ScrollableWrapper.tsx
export const ScrollableWrapper = memo(({ children, ...
  // ...
  const [viewportRect, setViewportRect] = useRef(null);
  // ...
  const [scrollX, setScrollX] = useRef(0);
  const [scrollY, setScrollY] = useRef(0);
  // ...
  return <ScrollableWrapperContext.Provider>
    <Viewport
      ref={viewportRef}
      onRectChange={setViewportRect}
      onScrollX={setScrollX}
      onScrollY={setScrollY}
      onIsScrollingX={setIsScrollingX}
      onIsScrollingY={setIsScrollingY}
      onIsDragging={setIsDragging}
      onIsHoverX={setIsHoverX}
      onIsHoverY={setIsHoverY}
      onIsHoverScrollbarX={setIsHoverScrollbarX}
      onIsHoverScrollbarY={setIsHoverScrollbarY}
      />
    <ScrollCache />
    {children}
  </ScrollableWrapperContext.Provider>
})
```

最后，ScrollCache 组件以非事件驱动方式工作，监听 onRenderEnd 回调，Viewport 内部的渲染完成后，ScrollCache 会将当前内容快照保存，以备后续使用。

2. 分析 viewportRender 模块

对于这个组件的内部模块来说，最关键的一环，对于一个可滚动，那么通过以下：

- 首先 getOffset 获取偏移量，然后根据偏移量，计算 startIndex 和 endIndex 的值。
- 接下来 getRenderRange 根据这两个值，计算出一个需要渲染的索引区间。
- 最后 getItem 根据这个索引区间，调用 renderItem 函数，生成实际的子组件。

```
// viewportRender.ts
export function*(*) { // 生成器函数
  const { offset, viewportSize, ... } = state;
  // 1. 计算偏移量
  const startOffset = getOffset(state, ...);
  // 2. 计算渲染范围
  const { startIndex, endIndex } = getRenderRange(state, ...);
  // 3. 生成子组件
  for (let i = startIndex; i <= endIndex; i++) {
    yield renderItem(i, ...);
  }
}
```

最后，通过 ScrollCache 缓存已经渲染过的内容，这样 ScrollCache 会直接使用缓存，而不需要重新计算。

```
// ScrollCache.tsx
export const ScrollCache = memo(({ ...
  // ...
  const [cache, setCache] = useRef(null);
  // ...
  return null;
})
```

二、以前 Viewport 组件

Viewport 是用户视图，它决定了哪些内容会被渲染，哪些内容会被裁剪。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
})
```

1. Viewport 的结构

Viewport 组件内部可以拆分为两部分，一部分是 RectCollectorWrapper 模块，负责监听 DOM 元素的尺寸变化和位置，通过 ResizeObserver 和 IntersectionObserver 来实现监听。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
  // ...
  <RectCollectorWrapper_ref={ref} />
  // ...
  <div_ref={ref} />
})
```

另一部分是 ScrollableWrapper 负责处理滚动事件，通过 RectCollectorWrapper 模块来实现 ScrollableWrapper 的尺寸变化，通过 ScrollableWrapper 的 onRectChange 回调，将 Viewport 的尺寸变化通知给父组件 ScrollableWrapper。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
  // ...
  <ScrollableWrapper_onRectChange={setRect} />
  // ...
  <div_ref={ref} />
})
```

2. Viewport 的 offset 模块

Viewport 的 offset 模块，是 Viewport 的核心模块，这个模块的作用是，当 ScrollableWrapper 的 onScrollX 或 onScrollY 事件发生时，通过 ScrollableWrapper 的 onRectChange 回调，将 Viewport 的 offset 变化通知给父组件 ScrollableWrapper。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
  // ...
  <ScrollableWrapper_onScrollX={setScrollX} />
  <ScrollableWrapper_onScrollY={setScrollY} />
  // ...
  <div_ref={ref} />
})
```

三、以前 ScrollableWrapper 组件

ScrollableWrapper 是 Scrollable 的内部实现，用户日常使用的是 Scrollable 组件，ScrollableWrapper 是 Scrollable 的内部实现。

```
// ScrollableWrapper.tsx
export const ScrollableWrapper = memo(({ ...
  // ...
  return <div_ref={ref} />
})
```

1. 以前 ScrollableWrapper 的结构

ScrollableWrapper 的结构，主要分为两部分，一部分是 RectCollectorWrapper 模块，通过 RectCollectorWrapper 模块来监听 ScrollableWrapper 的尺寸变化，通过 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Viewport。

```
// ScrollableWrapper.tsx
export const ScrollableWrapper = memo(({ ...
  // ...
  return <div_ref={ref} />
  // ...
  <RectCollectorWrapper_ref={ref} />
  // ...
  <div_ref={ref} />
})
```

另一部分是 ScrollableWrapper 的 onScrollX 和 onScrollY 事件，通过 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Viewport。

```
// ScrollableWrapper.tsx
export const ScrollableWrapper = memo(({ ...
  // ...
  return <div_ref={ref} />
  // ...
  <div_ref={ref} />
})
```

2. ScrollableWrapper 的渲染逻辑

最后总结，我们来回顾 ScrollableWrapper 的渲染逻辑，首先通过 RectCollectorWrapper 模块，监听 ScrollableWrapper 的尺寸变化，通过 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Viewport。

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Viewport。

```
// ScrollableWrapper.tsx
export const ScrollableWrapper = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  const [scrollX, setScrollX] = useRef(0);
  const [scrollY, setScrollY] = useRef(0);
  // ...
  return <div_ref={ref} />
  // ...
  <RectCollectorWrapper_ref={ref} />
  // ...
  <div_ref={ref} />
})
```

3. 以前 Viewport 的渲染逻辑

Viewport 的渲染逻辑，主要分为两部分，一部分是 RectCollectorWrapper 模块，通过 RectCollectorWrapper 模块来监听 Viewport 的尺寸变化，通过 onRectChange 回调，将 Viewport 的尺寸变化通知给父组件 ScrollableWrapper。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
  // ...
  <RectCollectorWrapper_ref={ref} />
  // ...
  <div_ref={ref} />
})
```

另一部分是 Viewport 的 offset 模块，通过 onScrollX 和 onScrollY 回调，将 Viewport 的 offset 变化通知给父组件 ScrollableWrapper。

```
// Viewport.tsx
export const Viewport = memo(({ ...
  // ...
  const [ref, setRef] = useRef<HTMLDivElement>(null);
  // ...
  return <div_ref={ref} />
  // ...
  <ScrollableWrapper_onScrollX={setScrollX} />
  <ScrollableWrapper_onScrollY={setScrollY} />
  // ...
  <div_ref={ref} />
})
```

四、Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

五、Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

六、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

七、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

八、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

九、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十一、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十二、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十三、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十四、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十五、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十六、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十七、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十八、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

十九、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十一、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十二、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十三、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十四、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十五、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十六、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十七、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十八、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

二十九、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十一、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十二、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十三、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十四、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十五、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十六、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十七、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十八、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

三十九、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

四十、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

四十一、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

四十二、以前 Scrollable 的 Viewport 的渲染逻辑

最后总结，我们来回顾 Scrollable 的渲染逻辑，首先通过 ScrollableWrapper 的 onRectChange 回调，将 ScrollableWrapper 的尺寸变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
  return <ScrollableWrapper />
})
```

接下来，通过 ScrollableWrapper 的 onScrollX 和 onScrollY 回调，将 ScrollableWrapper 的 scrollX 和 scrollY 变化通知给父组件 Scrollable。

```
// Scrollable.tsx
export const Scrollable = memo(({ ...
  // ...
```