1. 长列表渲染

- 如果有海量数据在浏览器里一次性渲染会有以下问题
  - 计算时间过长，用户需要长时间等待，体验差
  - CPU处理时间过长，滑动过程中可能卡顿
  - GPU负载过高，渲染不过来会出现闪动
  - 内存占用过多，严重会引起浏览器卡死和崩溃
- 优化方法
  - 下拉底部加载更多实现懒加载，此方法随着内容越来越多，会引起大量的重排和重绘，依赖可能会卡顿
  - 虚拟列表 其实我们的屏幕可视区域是有限的，能看到的数据也是有限的,所以可以在用户滚动时，只渲染可视区域内的内容即可,不可见区域用空白占位填充, 这样的话页面中的DOM元素少，CPU、GPU和内存负载小

2.长列表组件

- react-virtualized
- react-window
- react-window.vercel.app

```
npm i react-window --save
```

3. 固定高度列表实战

3.1 src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FixedSizeList from './fixed-size-list';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FixedSizeList />);
```

3.2 fixed-size-list.js

src\fixed-size-list.js

```
import {FixedSizeList} from 'react-window';
import './fixed-size-list.css';
const Row = ({index,style})=>(
    <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>Row{index}</div>
)
function App(){
    return (
        <FixedSizeList
          className='List'
          height={200}
          width={200}
          itemSize={50}
          itemCount={1000}
        >
            {Row}
        </FixedSizeList>
    )
}
export default App;
```

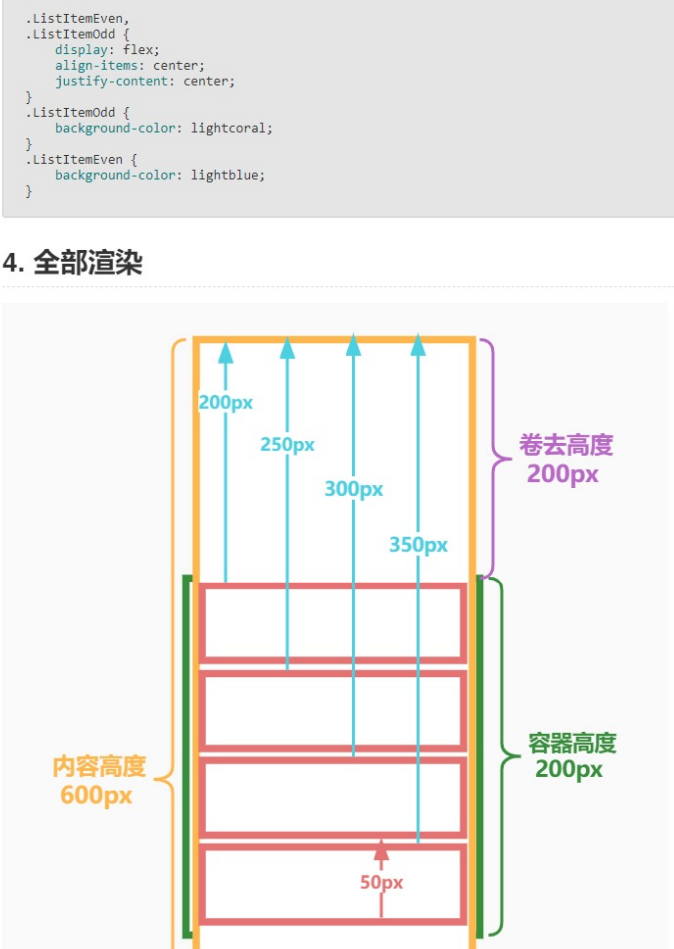
3.3 fixed-size-list.css

src\fixed-size-list.css

```
.List {
    border: 1px solid gray;
}

.ListItemEven,
.ListItemOdd {
    display: flex;
    align-items: center;
    justify-content: center;
}
.ListItemOdd {
    background-color: lightcoral;
}
.ListItemEven {
    background-color: lightblue;
}
```

4. 全部渲染



4.1 fixed-size-list.js

src\fixed-size-list.js

```
import {FixedSizeList} from './react-window';
import './fixed-size-list.css';
const Row = ({index,style})=>(
    <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>Row{index}</div>
)
function App(){
    return (
        <FixedSizeList
          className='List'
          height={200}
          width={200}
          itemSize={50}
          itemCount={1000}
        >
            {Row}
        </FixedSizeList>
    )
}
export default App;
```

4.2 react-window\index.js

src\react-window\index.js

```
export { default as FixedSizeList } from './FixedSizeList';
```

4.3 FixedSizeList.js

src\react-window\FixedSizeList.js

```
import createListComponent from './createListComponent';
const FixedSizeList = createListComponent({
    getItemSize: ({ itemSize }) => itemSize,//每个条目的高度
    getEstimatedTotalSize: ({ itemSize, itemCount }) => itemSize * itemCount, //获取预计的总高度
    getItemOffset: ({ itemSize }, index) => itemSize * index //获取每个条目的偏移量
});
export default FixedSizeList;
```

4.4 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset//获取每个条目的偏移量
}) {
    return class extends React.Component {
        render() {
            const {width,height,itemCount,children:ComponentType} = this.props;
            const containerStyle = {position:'relative',width,height,overflow:'auto', willChange: 'transform'};
            const contentStyle = {height:getEstimatedTotalSize(this.props),width:'100%'};
            const items = [];
            if(itemCount>0){
                for(let index = 0;index<itemCount;index++){
                    items.push(
                        <ComponentType key={index} index={index} style={this._getItemStyle(index)}/>
                    );
                }
            }
            return (
                <div style={containerStyle}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        _getItemStyle=(index)=>{
            const style = {
                position:'absolute',
                width:'100%',
                height:getItemSize(this.props),
                top:getItemOffset(this.props,index)
            };
            return style;
        }
    }
}
```

5. 渲染首屏

5.1 FixedSizeList.js

src\react-window\FixedSizeList.js

```
import createListComponent from './createListComponent';
const FixedSizeList = createListComponent({
    getItemSize: ({ itemSize }) => itemSize,//每个条目的高度
    getEstimatedTotalSize: ({ itemSize, itemCount }) => itemSize * itemCount, //获取预计的总高度
    getItemOffset: ({ itemSize }, index) => itemSize * index, //获取每个条目的偏移量
+   getStartIndexForOffset: ({ itemSize }, offset) => Math.floor(offset / itemSize),//获取起始索引
+   getStopIndexForStartIndex: ({ height, itemSize }, startIndex) => {//获取结束索引
+       const numVisibleItems = Math.ceil(height / itemSize);
+       return startIndex + numVisibleItems - 1;
    }
});
export default FixedSizeList;
```

5.2 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
+   getStartIndexForOffset,
+   getStopIndexForStartIndex
}) {
    return class extends React.Component {
+       state = { scrollOffset: 0 }
        render() {
            const { width, height, itemCount, children: ComponentType } = this.props;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props), width: '100%' };
            const items = [];
            if (itemCount > 0) {
+               const [startIndex, stopIndex] = this._getRangeToRender();
+               for (let index = startIndex; index <= stopIndex; index++) {
                    items.push(
                        <ComponentType key={index} index={index} style={this._getItemStyle(index)} />
                    );
                }
            }
            return (
                <div style={containerStyle}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        _getItemStyle = (index) => {
            const style = {
                position: 'absolute',
                width: '100%',
                height: getItemSize(this.props),
                top: getItemOffset(this.props, index)
            };
            return style;
        }
+       _getRangeToRender = () => {
+           const { scrollOffset } = this.state;
+           const startIndex = getStartIndexForOffset(this.props, scrollOffset);
+           const stopIndex = getStopIndexForStartIndex(this.props, startIndex);
+           return [startIndex, stopIndex];
+       }
    }
}
```

5. 监听滚动

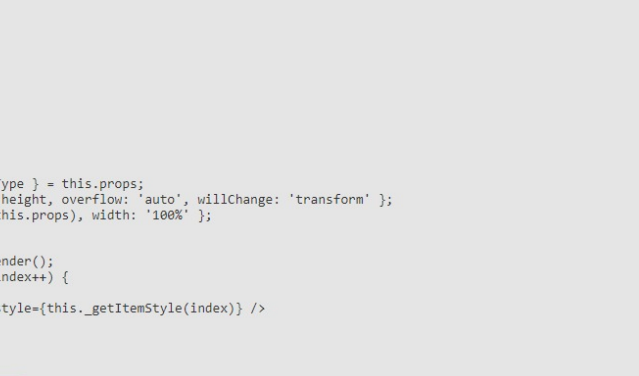
5.1 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex
}) {
    return class extends React.Component {
        state = { scrollOffset: 0 }
        render() {
            const { width, height, itemCount, children: ComponentType } = this.props;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
                    items.push(
                        <ComponentType key={index} index={index} style={this._getItemStyle(index)} />
                    );
                }
            }
            return (
+               <div style={containerStyle} onScroll={this.onScroll}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
+       onScroll = event => {
+           const { scrollTop } = event.currentTarget;
+           this.setState({ scrollOffset: scrollTop });
+       }
        _getItemStyle = (index) => {
            const style = {
                position: 'absolute',
                width: '100%',
                height: getItemSize(this.props),
                top: getItemOffset(this.props, index)
            };
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
            const startIndex = getStartIndexForOffset(this.props, scrollOffset);
            const stopIndex = getStopIndexForStartIndex(this.props, startIndex);
            return [startIndex, stopIndex]
        }
    }
}
```

6. overscan

- 过扫描实质上是切断图片的边缘，以确保所有重要的东西显示在屏幕上


Action safe area (93% W x 93% H)
Title safe area (80% W x 90% H)
CULDT @ ARU, CNY 2010

6.1 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex
}) {
    return class extends React.Component {
+       static defaultProps = {
+           overscanCount: 2
+       }
        state = { scrollOffset: 0 }
        render() {
            const { width, height, itemCount, children: ComponentType } = this.props;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
                    items.push(
                        <ComponentType key={index} index={index} style={this._getItemStyle(index)} />
                    );
                }
            }
            return (
                <div style={containerStyle} onScroll={this.onScroll}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        onScroll = event => {
            const { scrollTop } = event.currentTarget;
            this.setState({ scrollOffset: scrollTop });
        }
        _getItemStyle = (index) => {
            const style = {
                position: 'absolute',
                width: '100%',
                height: getItemSize(this.props),
                top: getItemOffset(this.props, index)
            };
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
+           const { itemCount, overscanCount } = this.props;
            const startIndex = getStartIndexForOffset(this.props, scrollOffset);
            const stopIndex = getStopIndexForStartIndex(this.props, startIndex);
            return [
+               Math.max(0, startIndex - overscanCount),
+               Math.max(0, Math.min(itemCount - 1, stopIndex + overscanCount)),
                startIndex, stopIndex]
        }
    }
}
```

7. VariableSizeList实战

7.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FixedSizeList from './fixed-size-list';
+import VariableSizeList from './variable-size-list';
const root = ReactDOM.createRoot(document.getElementById('root'));
+root.render(<VariableSizeList />);
```

7.2 variable-size-list.js

src\variable-size-list.js

```
import React from 'react';
import { VariableSizeList } from 'react-window';
import './variable-size-list.css';

const rowSizes = new Array(1000)
    .fill(true)
    .map(() => 25 + Math.round(Math.random() * 50));

const getItemSize = index => rowSizes[index];

const Row = ({ index, style }) => (
    <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>
        Row {index}
    </div>
)
const App = () => {
    return (
        <VariableSizeList
            className='List'
            height={200}
            width={200}
            itemSize={getItemSize}
            itemCount={1000}
        >
            {Row}
        </VariableSizeList>
    )
}
export default App;
```

7.3 variable-size-list.css

src\variable-size-list.css

```
.List {
    border: 1px solid gray;
}

.ListItemEven,
.ListItemOdd {
    display: flex;
    align-items: center;
    justify-content: center;
}
.ListItemOdd {
    background-color: lightcoral;
}
.ListItemEven {
    background-color: lightblue;
}
```

8. initInstanceProps

8.1 variable-size-list.js

src\variable-size-list.js

```
import React from 'react';
+import { VariableSizeList } from './react-window';
import './variable-size-list.css';

const rowSizes = new Array(1000)
    .fill(true)
    .map(() => 25 + Math.round(Math.random() * 50));

const getItemSize = index => rowSizes[index];

const Row = ({ index, style }) => (
    <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>
        Row {index}
    </div>
)
const App = () => {
    return (
        <VariableSizeList
            className='List'
            height={200}
            width={200}
            itemSize={getItemSize}
            itemCount={1000}
        >
            {Row}
        </VariableSizeList>
    )
}
export default App;
```

8.2 src\react-window\index.js

src\react-window\index.js

```
export { default as FixedSizeList } from './FixedSizeList';
+export { default as VariableSizeList } from './VariableSizeList';
```

8.3 VariableSizeList.js

src\react-window\VariableSizeList.js

```
import createListComponent from './createListComponent';
+const DEFAULT_ESTIMATED_SIZE = 50;
+const getEstimatedTotalSize = () => {}
+const VariableSizeList = createListComponent({
    getEstimatedTotalSize,
    getStartIndexForOffset: () => 0,
    getStopIndexForStartIndex: () => 0,
    getItemSize: () => 0,
    getItemOffset: () => 0,
+   initInstanceProps(props) {
+       const { estimatedItemSize } = props;
+       const instanceProps = {
+           estimatedItemSize: estimatedItemSize || DEFAULT_ESTIMATED_SIZE
+       }
+       return instanceProps;
+   }
});
+export default VariableSizeList;
```

8.4 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex,
+   initInstanceProps
}) {
    return class extends React.Component {
+       instanceProps = initInstanceProps&&initInstanceProps(this.props)
        static defaultProps = {
            overscanCount: 2
        }
    }
}
```

9. 预估总高度

9.1 src\react-window\VariableSizeList.js

src\react-window\VariableSizeList.js

```
import createListComponent from './createListComponent';
const DEFAULT_ESTIMATED_SIZE = 50;
+const getEstimatedTotalSize = ({ itemCount }, { estimatedItemSize }) => {
+   const numUnmeasuredItems = itemCount;//未测量的条目
+   const totalSizeOfUnmeasuredItems = numUnmeasuredItems * estimatedItemSize;//未测量条目的总高度
+   return totalSizeOfUnmeasuredItems;
+}
const VariableSizeList = createListComponent({
    getEstimatedTotalSize,
    getStartIndexForOffset: () => 0,
    getStopIndexForStartIndex: () => 0,
    getItemSize: () => 0,
    getItemOffset: () => 0,
    initInstanceProps(props) {
        const { estimatedItemSize } = props;
        const instanceProps = {
            estimatedItemSize: estimatedItemSize || DEFAULT_ESTIMATED_SIZE
        }
        return instanceProps;
    }
});
export default VariableSizeList;
```

9.2 src\react-window\createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    //......
}) {
    return class extends React.Component {
        render() {
            const { width, height, itemCount, children: ComponentType } = this.props;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
+           const contentStyle = { height: getEstimatedTotalSize(this.props, this.instanceProps), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
                    items.push(
                        <ComponentType key={index} index={index} style={this._getItemStyle(index)} />
                    );
                }
            }
            return (
                <div style={containerStyle} onScroll={this.onScroll}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        //......
    }
}
```

10. 动态计算高度

- lastMeasuredIndex 上次测试过高度的最大索引

10.1 VariableSizeList.js

src\react-window\VariableSizeList.js

```
import createListComponent from './createListComponent';
const DEFAULT_ESTIMATED_SIZE = 50;
+const getEstimatedTotalSize = ({ itemCount }, { estimatedItemSize, lastMeasuredIndex, itemMetadataMap }) => {
+  let totalSizeOfMeasuredItems = 0;//计算过的条目总大小
+  if (lastMeasuredIndex >= 0) {
+      const itemMetadata = itemMetadataMap[lastMeasuredIndex];
+      totalSizeOfMeasuredItems = itemMetadata.offset + itemMetadata.size;//测试过的总大小
+  }
+  const numUnmeasuredItems = itemCount - lastMeasuredIndex - 1;//未测量的条目
   const totalSizeOfUnmeasuredItems = numUnmeasuredItems * estimatedItemSize;//未测量条目的总高度
   return totalSizeOfMeasuredItems + totalSizeOfUnmeasuredItems;
}
+function findNearestItem(props, instanceProps, offset) {
+   const { lastMeasuredIndex } = instanceProps;
+   for (let index = 0; index <= lastMeasuredIndex; index++) {
+       const currentOffset = getItemMetadata(props, index, instanceProps).offset;
+       if (currentOffset >= offset) {
+           return index;
+       }
+   }
+   return 0;
+}
+function getItemMetadata(props, index, instanceProps) {
+   const { itemSize } = props;
+   const { itemMetadataMap, lastMeasuredIndex } = instanceProps;
+   if (index > lastMeasuredIndex) {
+       let offset = 0;//先计算上一个测试过的条目的下一个offset
+       if (lastMeasuredIndex >= 0) {
+           const itemMetadata = itemMetadataMap[lastMeasuredIndex];
+           offset = itemMetadata.offset + itemMetadata.size;
+       }
+       //计算从上一个条目到本次索引的offset和size
+       for (let i = lastMeasuredIndex + 1; i <= index; i++) {
+           let size = itemSize(i);
+           itemMetadataMap[i] = { offset, size };
+           offset += size;
+       }
+       instanceProps.lastMeasuredIndex = index;
+   }
+   return itemMetadataMap[index];
+}
const VariableSizeList = createListComponent({
    getEstimatedTotalSize,
+   getStartIndexForOffset: (props, offset, instanceProps) => findNearestItem(props, instanceProps, offset),
+   getStopIndexForStartIndex: (props, startIndex, scrollOffset, instanceProps) => {
+       const { itemCount, height } = props;
+       const itemMetadata = getItemMetadata(props, startIndex, instanceProps);
+       const maxOffset = scrollOffset + height;
+       let offset = itemMetadata.offset + itemMetadata.size;
+       let stopIndex = startIndex;
+       while (stopIndex < itemCount - 1 && offset < maxOffset) {
+           stopIndex++;
+           offset += getItemMetadata(props, stopIndex, instanceProps).size;
+       }
+       return stopIndex;
+   },
+   getItemSize: (props, index, instanceProps) => getItemMetadata(props, index, instanceProps).size,
+   getItemOffset: (props, index, instanceProps) => getItemMetadata(props, index, instanceProps).offset,
    initInstanceProps(props) {
        const { estimatedItemSize } = props;
        const instanceProps = {
            estimatedItemSize: estimatedItemSize || DEFAULT_ESTIMATED_SIZE,
+           itemMetadataMap: {},//存放每个条目的高度和偏移量
+           lastMeasuredIndex: -1//最后一个测量高度的索引
        }
        return instanceProps;
    }
});
export default VariableSizeList;
```

10.2 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
export default function createListComponent({}) {
    return class extends React.Component {
        _getItemStyle = (index) => {
            const style = {
                position: 'absolute',
                width: '100%',
+               height: getItemSize(this.props, index, this.instanceProps),
+               top: getItemOffset(this.props, index, this.instanceProps)
            };
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
            const { itemCount, overscanCount } = this.props;
+           const startIndex = getStartIndexForOffset(this.props, scrollOffset, this.instanceProps);
+           const stopIndex = getStopIndexForStartIndex(this.props, startIndex, scrollOffset, this.instanceProps);
            return [
                Math.max(0, startIndex - overscanCount),
                Math.max(0, Math.min(itemCount - 1, stopIndex + overscanCount)),
                startIndex, stopIndex]
        }
    }
}
```

11. 优化方案

11.1 缓存样式

11.1.1 createListComponent.js

src\react-window\createListComponent.js

```
    return class extends React.Component {
+       itemStyleCache = new Map()
        instanceProps = initInstanceProps&&initInstanceProps(this.props)
        _getItemStyle = (index) => {
+           let style;
+           if (this.itemStyleCache.has(index)) {
+               style = this.itemStyleCache.get(index);
+           } else {
                style = {
                    position: 'absolute',
                    width: '100%',
                    height: getItemSize(this.props, index, this.instanceProps),
                    top: getItemOffset(this.props, index, this.instanceProps)
                };
+               this.itemStyleCache.set(index, style);
+           }
            return style;
        }
    }
}
```

11.2 二分查找和指数扩充

11.2.1 VariableSizeList.js

src\react-window\VariableSizeList.js

```
+function findNearestItem(props, instanceProps, offset) {
+  const { itemMetadataMap, lastMeasuredIndex } = instanceProps;
+  const lastMeasuredItemOffset =
+    lastMeasuredIndex > 0 ? itemMetadataMap[lastMeasuredIndex].offset : 0;
+  if (lastMeasuredItemOffset >= offset) {
+    return findNearestItemBinarySearch(props, instanceProps, lastMeasuredIndex, 0, offset);
+  } else {
+    return findNearestItemExponentialSearch(
+      props,
+      instanceProps,
+      Math.max(0, lastMeasuredIndex),
+      offset
+    );
+  }
+  //return findNearestItemBinarySearch(props, instanceProps, lastMeasuredIndex, 0, offset);
+  //在源码里此处用的是二分查找，把时间复杂度从N=>logN
+  /* for (let index = 0; index <= lastMeasuredIndex; index++) {
+    const currentOffset = getItemMetadata(props, index, instanceProps).offset;
+    //currentOffset=当前条目的offset offset=当前容器向上卷去的高度
+    if (currentOffset >= offset) {
+      return index;
+    }
+  }
+  return 0; */
+}
+function findNearestItemExponentialSearch(props, instanceProps, index, offset) {
+  const { itemCount } = props;
+  let interval = 1;
+  while (
+    index < itemCount &&
+    getItemMetadata(props, index, instanceProps).offset < offset
+  ) {
+    index += interval;
+    interval *= 2;
+  }
+  return findNearestItemBinarySearch(props, instanceProps, Math.min(index, itemCount - 1), Math.floor(index / 2), offset);
+}
+const findNearestItemBinarySearch = (
+  props,
+  instanceProps,
+  high,
+  low,
+  offset
+) => {
+  while (low <= high) {
+    const middle = low + Math.floor((high - low) / 2);
+    const currentOffset = getItemMetadata(props, middle, instanceProps).offset;
+    if (currentOffset === offset) {
+      return middle;
+    } else if (currentOffset < offset) {
+      low = middle + 1;
+    } else if (currentOffset > offset) {
+      high = middle - 1;
+    }
+  }
+  if (low > 0) {
+    return low - 1;
+  } else {
+    return 0;
+  }
+};
```

11.3 IntersectionObserver

- IntersectionObserver接口(从属于Intersection Observer API)为开发者提供了一种可以异步监听目标元素与其祖先或视窗(viewport)交叉状态的手段。祖先元素与视窗(viewport)被称为根(root)
- 网页开发时，常常需要判断某个元素是否进入了视口(viewport,即用户能不能看到它，然后执行相应的逻辑
- 常见的方法是监听 `scroll` 事件，调用元素的 `getBoundingClientRect` 方法，得到它对应于视口左上角的坐标，再判断是否在视口之内。这种方法的缺点是，由于scroll事件密集发生，计算量很大，容易造成性能问题

11.3.1 createListComponent.js

src\react-window\createListComponent.js

```
function createListComponent({
  getEstimatedTotalSize,
  getItemSize,
  getItemOffset,
  getStartIndexForOffset,//根据向上卷去的高度计算开始索引
  getStopIndexForStartIndex,//根据开始索引和容器的高度计算结束索引
  initInstanceProps
}) {
  return class extends React.Component {
    constructor(props) {
      super(props);
      this.instanceProps = initInstanceProps && initInstanceProps(this.props)
      this.state = { scrollOffset: 0 }
+     this.outerRef = React.createRef();
+     this.oldFirstRef = React.createRef();
+     this.oldLastRef = React.createRef();
+     this.firstRef = React.createRef();
+     this.lastRef = React.createRef();
    }
    static defaultProps = {
      overscanCount: 2
    }
+   componentDidMount() {
+     this.observe(this.oldFirstRef.current = this.firstRef.current);
+     this.observe(this.oldLastRef.current = this.lastRef.current);
+   }
+   componentDidUpdate() {
+     if (this.oldFirstRef.current !== this.firstRef.current) {
+       this.oldFirstRef.current = this.firstRef.current;
+       this.observe(this.firstRef.current);
+     }
+     if (this.oldLastRef.current !== this.lastRef.current) {
+       this.oldLastRef.current = this.lastRef.current;
+       this.observe(this.lastRef.current);
+     }
+   }
+   observe = (dom) => {
+     let io = new IntersectionObserver((entries) => {
+       entries.forEach(this.onScroll);
+     }, { root: this.outerRef.current })
+     io.observe(dom);
+   }
    render() {
      const { width, height, children: Row } = this.props;
      const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
      const contentStyle = { width: '100%', height: getEstimatedTotalSize(this.props, this.instanceProps) };
      const items = [];
+     const [startIndex, stopIndex, originStartIndex, originStopIndex] = this.getRangeToRender();
      for (let index = startIndex; index <= stopIndex; index++) {
+       const style = this.getItemStyle(index);
+       if (index === originStartIndex) {
+         items.push(
+           <Row key={index} index={index} style={style} forwardRef={this.firstRef} />
+         );
+         continue;
+       } else if (index === originStopIndex) {
+         items.push(
+           <Row key={index} index={index} style={style} forwardRef={this.lastRef} />
+         );
+         continue;
+       }
        items.push(
          <Row key={index} index={index} style={style} />
        );
      }
      return (
+       <div style={containerStyle} ref={this.outerRef} >
          <div style={contentStyle} >
            {items}
          </div>
        </div>
      )
    }
    onScroll = () => {
      const { scrollTop } = this.outerRef.current;
      this.setState({ scrollOffset: scrollTop })
    }
    getRangeToRender = () => {
      const { scrollOffset } = this.state;
      const { itemCount, overscanCount } = this.props;
      const startIndex = getStartIndexForOffset(this.props, scrollOffset, this.instanceProps);
      const stopIndex = getStopIndexForStartIndex(this.props, startIndex, scrollOffset, this.instanceProps);
      return [
        Math.max(0, startIndex - overscanCount),
        Math.min(itemCount - 1, stopIndex + overscanCount),
        startIndex, stopIndex];
    }
    getItemStyle = (index) => {
      const style = {
        position: 'absolute',
        width: '100%',
        height: getItemSize(this.props, index, this.instanceProps),
        top: getItemOffset(this.props, index, this.instanceProps)
      }
      return style;
    }
  }
}
```

13. 动态高度列表

- react-window
- dynamic-size
- ResizeObserver
- react-virtual
- virtuoso

13.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FixedSizeList from './fixed-size-list';
import VariableSizeList from './variable-size-list';
+import DynamicSizeList from './dynamic-size-list'
const root = ReactDOM.createRoot(document.getElementById('root'));
+root.render(<DynamicSizeList />);
```

13.2 dynamic-size-list.js

src\dynamic-size-list.js

```
import React from 'react';
import { VariableSizeList } from './react-window';

const items = [];
for (let i = 0; i < 1000; i++) {
    const height = (30 + Math.floor(Math.random() * 20)) + 'px';
    const style = {
        height,
        width: '100%',
        backgroundColor: i % 2 ? 'green' : "orange",
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center'
    }
    items.push(<div style={style}>Row {i}</div>);
}

const Row = ({ index }) => items[index]
const App = () => {
    return (
        <VariableSizeList
            isDynamic={true}
            className='List'
            height={200}
            width={200}
            itemCount={1000}
        >
            {Row}
        </VariableSizeList>
    )
}
export default App
```

13.4 VariableSizeList.js

src\react-window\VariableSizeList.js

```
function getItemMetadata(props, index, instanceProps) {
    const { itemSize } = props;
    const { itemMetadataMap, lastMeasuredIndex } = instanceProps;
    if (index > lastMeasuredIndex) {
        let offset = 0;//先计算上一个测试过的条目的下一个offset
        if (lastMeasuredIndex >= 0) {
            const itemMetadata = itemMetadataMap[lastMeasuredIndex];
            offset = itemMetadata.offset + itemMetadata.size;
        }
        //计算从上一个条目到本次索引的offset和size
        for (let i = lastMeasuredIndex + 1; i <= index; i++) {
+           let size = itemSize ? itemSize(i) : DEFAULT_ESTIMATED_SIZE;
            itemMetadataMap[i] = { offset, size };
            offset += size;
        }
        instanceProps.lastMeasuredIndex = index;
    }
    return itemMetadataMap[index];
}
```

13.5 createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';

+class ListItem extends React.Component {
+   constructor(props) {
+       super(props);
+       this.domRef = React.createRef();
+       this.resizeObserver = null;
+   }
+   componentDidMount() {
+       if (this.domRef.current) {
+           const node = this.domRef.current.firstChild;
+           const { index, onSizeChange } = this.props;
+           this.resizeObserver = new ResizeObserver(() => {
+               onSizeChange(index, node);
+           });
+           this.resizeObserver.observe(node);
+       }
+   }
+   componentWillUnmount() {
+       if (this.resizeObserver && this.domRef.current.firstChild) {
+           this.resizeObserver.unobserve(this.domRef.current.firstChild);
+       }
+   }
+   render() {
+       const { index, style, ComponentType } = this.props;
+       return (
+           <div style={style} ref={this.domRef}>
+               <ComponentType index={index} />
+           </div>
+       )
+   }
+}
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex,
    initInstanceProps
}) {
    return class extends React.Component {
        itemStyleCache = new Map()
        instanceProps = initInstanceProps&&initInstanceProps(this.props)
        static defaultProps = {
            overscanCount: 2
        }
        state = { scrollOffset: 0 }
+       onSizeChange = (index, node) => {
+           const height = node.offsetHeight;
+           const { itemMetadataMap, lastMeasuredIndex } = this.instanceProps;
+           const itemMetadata = itemMetadataMap[index];
+           itemMetadata.size = height;
+           let offset = 0;
+           for (let i = 0; i <= lastMeasuredIndex; i++) {
+               const itemMetadata = itemMetadataMap[i];
+               itemMetadata.offset = offset;
+               offset = offset + itemMetadata.size;
+           }
+           this.itemStyleCache.clear();
+           this.forceUpdate();
+       }
        render() {
+           const { width, height, itemCount, children: ComponentType, isDynamic } = this.props;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props, this.instanceProps), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
+                   if (isDynamic) {
+                       items.push(
+                           <ListItem
+                               key={index} index={index}
+                               style={this._getItemStyle(index)}
+                               ComponentType={ComponentType}
+                               onSizeChange={this.onSizeChange}
+                           />
+                       );
+                   } else {
+                       items.push(
+                           <ComponentType
+                               key={index} index={index}
+                               style={this._getItemStyle(index)}
+                           />
+                       );
+                   }
                }
            }
            return (
                <div style={containerStyle} onScroll={this.onScroll}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        onScroll = event => {
            const { scrollTop } = event.currentTarget;
            this.setState({ scrollOffset: scrollTop });
        }
        _getItemStyle = (index) => {
            let style;
            if (this.itemStyleCache.has(index)) {
                style = this.itemStyleCache.get(index);
            } else {
                style = {
                    position: 'absolute',
                    width: '100%',
                    height: getItemSize(this.props, index, this.instanceProps),
                    top: getItemOffset(this.props, index, this.instanceProps)
                };
                this.itemStyleCache.set(index, style);
            }
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
            const { itemCount, overscanCount } = this.props;
            const startIndex = getStartIndexForOffset(this.props, scrollOffset, this.instanceProps);
            const stopIndex = getStopIndexForStartIndex(this.props, startIndex, scrollOffset, this.instanceProps);
            return [
                Math.max(0, startIndex - overscanCount),
                Math.max(0, Math.min(itemCount - 1, stopIndex + overscanCount)),
                startIndex, stopIndex]
        }
    }
}
```

14. 滚动状态

14.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FixedSizeList from './fixed-size-list';
import VariableSizeList from './variable-size-list';
import DynamicSizeList from './dynamic-size-list'
const root = ReactDOM.createRoot(document.getElementById('root'));
+root.render(<FixedSizeList />);
```

14.2 fixed-size-list.js

src\fixed-size-list.js

```
import { FixedSizeList } from './react-window';
import './fixed-size-list.css';
+const Row = ({ index, style, isScrolling }) => (
+   <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>
+       {isScrolling ? 'Scrolling' : `Row ${index}`}
+   </div>
+)

function App() {
    return (
        <FixedSizeList
            className='List'
            height={200}
            width={200}
            itemSize={50}
            itemCount={1000}
+           useIsScrolling
        >
            {Row}
        </FixedSizeList>
    )
}
export default App;
```

14.3 src\react-window\createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
+import { requestTimeout, cancelTimeout } from './timer';
+const IS_SCROLLING_DEBOUNCE_INTERVAL = 150;
class ListItem extends React.Component {
    constructor(props) {
        super(props);
        this.domRef = React.createRef();
        this.resizeObserver = null;
    }
    componentDidMount() {
        if (this.domRef.current) {
            const node = this.domRef.current.firstChild;
            const { index, onSizeChange } = this.props;
            this.resizeObserver = new ResizeObserver(() => {
                onSizeChange(index, node);
            });
            this.resizeObserver.observe(node);
        }
    }
    componentWillUnmount() {
        if (this.resizeObserver && this.domRef.current.firstChild) {
            this.resizeObserver.unobserve(this.domRef.current.firstChild);
        }
    }
    render() {
        const { index, style, ComponentType } = this.props;
        return (
            <div style={style} ref={this.domRef}>
                <ComponentType index={index} />
            </div>
        )
    }
}
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex,
    initInstanceProps
}) {
    return class extends React.Component {
        itemStyleCache = new Map()
        instanceProps = initInstanceProps && initInstanceProps(this.props)
        static defaultProps = {
            overscanCount: 2,
+           useIsScrolling: false
        }
+       state = { scrollOffset: 0, isScrolling: false }
        onSizeChange = (index, node) => {
            const height = node.offsetHeight;
            const { itemMetadataMap, lastMeasuredIndex } = this.instanceProps;
            const itemMetadata = itemMetadataMap[index];
            itemMetadata.size = height;
            let offset = 0;
            for (let i = 0; i <= lastMeasuredIndex; i++) {
                const itemMetadata = itemMetadataMap[i];
                itemMetadata.offset = offset;
                offset = offset + itemMetadata.size;
            }
            this.itemStyleCache.clear();
            this.forceUpdate();
        }
        render() {
+           const { width, height, itemCount, children: ComponentType, isDynamic, useIsScrolling } = this.props;
+           const { isScrolling } = this.state;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props, this.instanceProps), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
                    if (isDynamic) {
                        items.push(
                            <ListItem
                                key={index} index={index}
                                style={this._getItemStyle(index)}
                                ComponentType={ComponentType}
                                onSizeChange={this.onSizeChange}
+                               isScrolling={useIsScrolling && isScrolling}
                            />
                        );
                    } else {
                        items.push(
                            <ComponentType
                                key={index} index={index}
                                style={this._getItemStyle(index)}
+                               isScrolling={useIsScrolling && isScrolling}
                            />
                        );
                    }
                }
            }
            return (
                <div style={containerStyle} onScroll={this.onScroll}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        onScroll = event => {
            const { scrollTop } = event.currentTarget;
+           this.setState({ scrollOffset: scrollTop, isScrolling: true }, this._resetIsScrollingDebounced);
        }
+       _resetIsScrollingDebounced = () => {
+           if (this._resetIsScrollingTimeoutId) {
+               cancelTimeout(this._resetIsScrollingTimeoutId);
+           }
+           this._resetIsScrollingTimeoutId = requestTimeout(
+               this._resetIsScrolling,
+               IS_SCROLLING_DEBOUNCE_INTERVAL
+           );
+       };
+       _resetIsScrolling = () => {
+           this._resetIsScrollingTimeoutId = null;
+           this.setState({ isScrolling: false });
+       }
        _getItemStyle = (index) => {
            let style;
            if (this.itemStyleCache.has(index)) {
                style = this.itemStyleCache.get(index);
            } else {
                style = {
                    position: 'absolute',
                    width: '100%',
                    height: getItemSize(this.props, index, this.instanceProps),
                    top: getItemOffset(this.props, index, this.instanceProps)
                };
                this.itemStyleCache.set(index, style);
            }
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
            const { itemCount, overscanCount } = this.props;
            const startIndex = getStartIndexForOffset(this.props, scrollOffset, this.instanceProps);
            const stopIndex = getStopIndexForStartIndex(this.props, startIndex, scrollOffset, this.instanceProps);
            return [
                Math.max(0, startIndex - overscanCount),
                Math.max(0, Math.min(itemCount - 1, stopIndex + overscanCount)),
                startIndex, stopIndex]
        }
    }
}
```

15. 滚动到指定条目

15.1 src\fixed-size-list.js

src\fixed-size-list.js

```
import React from 'react';
import { FixedSizeList } from './react-window';
import './fixed-size-list.css';
const Row = ({ index, style, isScrolling }) => (
    <div className={index % 2 ? 'ListItemOdd' : 'ListItemEven'} style={style}>
        {isScrolling ? 'Scrolling' : `Row ${index}`}
    </div>
)

function App() {
+   const listRef = React.useRef();
    return (
+       <>
+           <button onClick={() => listRef.current.scrollToItem(50)}>滚动到50</button>
            <FixedSizeList
                className='List'
                height={200}
                width={200}
                itemSize={50}
                itemCount={1000}
                useIsScrolling
+               ref={listRef}
            >
                {Row}
            </FixedSizeList>
+       </>
    )
}
export default App;
```

15.2 FixedSizeList.js

src\react-window\FixedSizeList.js

```
import createListComponent from './createListComponent';
const FixedSizeList = createListComponent({
    getItemSize: ({ itemSize }) => itemSize,//每个条目的高度
    getEstimatedTotalSize: ({ itemSize, itemCount }) => itemSize * itemCount, //获取预计的总高度
    getItemOffset: ({ itemSize }, index) => itemSize * index, //获取每个条目的偏移量
    getStartIndexForOffset: ({ itemSize }, offset) => Math.floor(offset / itemSize),//获取起始索引
    getStopIndexForStartIndex: ({ height, itemSize }, startIndex) => {//获取结束索引
        const numVisibleItems = Math.ceil(height / itemSize);
        return startIndex + numVisibleItems - 1;
    },
+   getOffsetForIndex: (props, index) => {
+       const { itemSize } = props;
+       return itemSize * index;
+   }
});
export default FixedSizeList;
```

15.3 src\react-window\createListComponent.js

src\react-window\createListComponent.js

```
import React from 'react';
import { requestTimeout, cancelTimeout } from './timer';
const IS_SCROLLING_DEBOUNCE_INTERVAL = 150;
class ListItem extends React.Component {
    constructor(props) {
        super(props);
        this.domRef = React.createRef();
        this.resizeObserver = null;
    }
    componentDidMount() {
        if (this.domRef.current) {
            const node = this.domRef.current.firstChild;
            const { index, onSizeChange } = this.props;
            this.resizeObserver = new ResizeObserver(() => {
                onSizeChange(index, node);
            });
            this.resizeObserver.observe(node);
        }
    }
    componentWillUnmount() {
        if (this.resizeObserver && this.domRef.current.firstChild) {
            this.resizeObserver.unobserve(this.domRef.current.firstChild);
        }
    }
    render() {
        const { index, style, ComponentType } = this.props;
        return (
            <div style={style} ref={this.domRef}>
                <ComponentType index={index} />
            </div>
        )
    }
}
export default function createListComponent({
    getEstimatedTotalSize,//获取预计的总高度
    getItemSize,//每个条目的高度
    getItemOffset,//获取每个条目的偏移量
    getStartIndexForOffset,
    getStopIndexForStartIndex,
    initInstanceProps,
+   getOffsetForIndex
}) {
    return class extends React.Component {
+       outerRef = React.createRef();
        itemStyleCache = new Map()
        instanceProps = initInstanceProps && initInstanceProps(this.props)
        static defaultProps = {
            overscanCount: 2,
            useIsScrolling: false
        }
+       scrollTo = (scrollOffset) => {
+           this.setState({ scrollOffset: Math.max(0, scrollOffset) });
+       }
+       scrollToItem = (index) => {
+           const { itemCount } = this.props;
+           index = Math.max(0, Math.min(index, itemCount - 1))
+           this.scrollTo(
+               getOffsetForIndex(this.props, index)
+           )
+       }
+       componentDidUpdate() {
+           const { scrollOffset } = this.state;
+           this.outerRef.current.scrollTop = scrollOffset;
+       }
        state = { scrollOffset: 0, isScrolling: false }
        onSizeChange = (index, node) => {
            const height = node.offsetHeight;
            const { itemMetadataMap, lastMeasuredIndex } = this.instanceProps;
            const itemMetadata = itemMetadataMap[index];
            itemMetadata.size = height;
            let offset = 0;
            for (let i = 0; i <= lastMeasuredIndex; i++) {
                const itemMetadata = itemMetadataMap[i];
                itemMetadata.offset = offset;
                offset = offset + itemMetadata.size;
            }
            this.itemStyleCache.clear();
            this.forceUpdate();
        }
        render() {
            const { width, height, itemCount, children: ComponentType, isDynamic, useIsScrolling } = this.props;
            const { isScrolling } = this.state;
            const containerStyle = { position: 'relative', width, height, overflow: 'auto', willChange: 'transform' };
            const contentStyle = { height: getEstimatedTotalSize(this.props, this.instanceProps), width: '100%' };
            const items = [];
            if (itemCount > 0) {
                const [startIndex, stopIndex] = this._getRangeToRender();
                for (let index = startIndex; index <= stopIndex; index++) {
                    if (isDynamic) {
                        items.push(
                            <ListItem
                                key={index} index={index}
                                style={this._getItemStyle(index)}
                                ComponentType={ComponentType}
                                onSizeChange={this.onSizeChange}
                                isScrolling={useIsScrolling && isScrolling}
                            />
                        );
                    } else {
                        items.push(
                            <ComponentType
                                key={index} index={index}
                                style={this._getItemStyle(index)}
                                isScrolling={useIsScrolling && isScrolling}
                            />
                        );
                    }
                }
            }
            return (
+               <div style={containerStyle} onScroll={this.onScroll} ref={this.outerRef}>
                    <div style={contentStyle}>
                        {items}
                    </div>
                </div>
            )
        }
        onScroll = event => {
            const { scrollTop } = event.currentTarget;
            this.setState({ scrollOffset: scrollTop, isScrolling: true }, this._resetIsScrollingDebounced);
        }
        _resetIsScrollingDebounced = () => {
            if (this._resetIsScrollingTimeoutId) {
                cancelTimeout(this._resetIsScrollingTimeoutId);
            }
            this._resetIsScrollingTimeoutId = requestTimeout(
                this._resetIsScrolling,
                IS_SCROLLING_DEBOUNCE_INTERVAL
            );
        };
        _resetIsScrolling = () => {
            this._resetIsScrollingTimeoutId = null;
            this.setState({ isScrolling: false });
        }
        _getItemStyle = (index) => {
            let style;
            if (this.itemStyleCache.has(index)) {
                style = this.itemStyleCache.get(index);
            } else {
                style = {
                    position: 'absolute',
                    width: '100%',
                    height: getItemSize(this.props, index, this.instanceProps),
                    top: getItemOffset(this.props, index, this.instanceProps)
                };
                this.itemStyleCache.set(index, style);
            }
            return style;
        }
        _getRangeToRender = () => {
            const { scrollOffset } = this.state;
            const { itemCount, overscanCount } = this.props;
            const startIndex = getStartIndexForOffset(this.props, scrollOffset, this.instanceProps);
            const stopIndex = getStopIndexForStartIndex(this.props, startIndex, scrollOffset, this.instanceProps);
            return [
                Math.max(0, startIndex - overscanCount),
                Math.max(0, Math.min(itemCount - 1, stopIndex + overscanCount)),
                startIndex, stopIndex]
        }
    }
}
```

16.其它方案

- react-virtual
- virtuoso

16.1 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FixedSizeList from './fixed-size-list';
import VariableSizeList from './variable-size-list';
import DynamicSizeList from './dynamic-size-list'
+import Virtuoso from './Virtuoso'
const root = ReactDOM.createRoot(document.getElementById('root'));
+root.render(<DynamicSizeList />);
```

16.2 src\Virtuoso.js

src\Virtuoso.js

```
import React from 'react'
import { Virtuoso } from 'react-virtuoso'
const items = [];
for (let i = 0; i < 200; i++) {
  const height = (30+Math.random() * 20) + 'px';
  const style = {
    height,
    width: `100%`,
    backgroundColor:i%2?'green':"orange"
  }
  items.push(<div style={ style }>Row {i}</div>);
}
const App = () => (
    <Virtuoso
        style={{ height: '200px',width:'200px' }}
        totalCount={200}
        itemContent={index => items[index]}
    />
)
export default App;
```

16.3 ResizeObserver

- • ResizeObserver

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ResizeObserver</title>
</head>

<body>
    <img id="logo" />
    <script>
        let logo = document.getElementById('logo');
        console.log(logo.offsetHeight);
        const resizeObserver = new ResizeObserver(entries => {
            console.log(logo.offsetHeight);
        });
        resizeObserver.observe(logo);
        setTimeout(()=>{
            logo.src = 'https://img.zhufengpeixun.com/zfjglogo.png';
        },1000);
    </script>
</body>
</html>
```