购物车列表 (动画)

使用ReactCSSTransitionGroup

```
npm i react-addons-css-transition-group
```

`src/shop/index.js`

```
import React, { Component } from 'react'
import ReactCSSTransitionGroup from 'react-addons-css-transition-group'

import './shop.css'

class AnimateShoppingList extends Component {
  constructor () {
    super()
    this.state = {
      items: [
        {
          id: 1,
          name: 'Milk'
        },
        {
          id: 2,
          name: 'Yogurt'
        },
        {
          id: 3,
          name: 'Orange Juice'
        }
      ]
    }
  }

  // 添加条目
  handleChange (evt) {
    if (evt.key === 'Enter') {
      let newItem = {
        id: Date.now(),
        name: evt.target.value
      }

      let newItems = this.state.items.concat(newItem)

      this.setState({
        items: newItems
      })
    }
  }

  // 删除条目
  handleRemove (index) {
```

```
    let { items } = this.state
    items.splice(index, 1)
    this.setState({
      items: items
    })
  }

  render () {
    let shoppingItems = this.state.items.map((item, i) => {
      return (
        <div key={ item.id } className="item"
          onClick={ this.handleRemove.bind(this, i) }
        >
          { item.name }
        </div>
      )
    })
    return (
      <div>
        {/*
          // 映射到CSS中包含实际动画定义的类名
          transitionName="example"
          // 动画持续的时间
          transitionEnterTimeout={ 300 }
          transitionLeaveTimeout={ 300 }

          // 【可选】  用于在组件初始挂在时执行一个额外的过渡阶段
          transitionAppear={ true }
          transitionAppearTimeout={ 300 }
        */}
        <ReactCSSTransitionGroup
          transitionName="example"
          transitionEnterTimeout={ 300 }
          transitionLeaveTimeout={ 300 }
          transitionAppear={ true }
          transitionAppearTimeout={ 300 }
        >
          { shoppingItems }
        </ReactCSSTransitionGroup>
        <input type="text" value={ this.state.newItem }
          onKeyDown={ this.handleChange.bind(this) }
        />
      </div>
    )
  }
}

export default AnimateShoppingList
```

src/shop/shop.css

```
input {
  padding: 5px;
  width: 120px;
  margin-top: 10px;
}

```

```css
.item {
  background-color: #efefef;
  cursor: pointer;
  display: block;
  margin-bottom: 1px;
  padding: 0 12px;
  width: 120px;
}

/* 进入之前 */
.example-enter {
  opacity: 0;
  transform: translateX(-250px);
}
/* 进入时 */
.example-enter.example-enter-active {
  opacity: 1;
  transform: translateX(0);
  transition: 0.3s;
}

/* 离开之前 */
.example-leave {
  opacity: 1;
  transform: translateX(0);
}
/* 离开时 */
.example-leave.example-leave-active {
  opacity: 0;
  transform: translateX(250px);
  transition: 0.3s;
}
.example-appear {
  opacity: 0;
  transform: translateX(-250px);
}
.example-appear.example-appear-active {
  opacity: 1;
  transform: translateX(0);
  transition: 0.5s;
}
```

拖放

安装 React Dnd 2

```
npm i react-dnd react-dnd-html5-backend -S
```

> React DnD 库 提供了三个高阶组件，他们必须应用于你的应用程序的不同组件中，这三个高阶组件分别是DragSource, DropTarget, DragDropContext

- DragSource

  > 会返回指定组件的增强版，是组建成为一个可被拖拽的元素

- DropTarget

  > 同样会返回增强版的组件，使其有能力处理被拖放其内部的元素

- DragDropContext

  > 封装了发生拖放交互行为的父元素，在交互场景背后

DragSource 和 DropTarget 高阶组件

- type

  > 指定了组件的名称，在复杂的UI中，可能会出现多个不同类型的拖拽源和多个不同类型的放置目标之间的交互，所以需要给每个源或目标设置一个特定的标识

- spec 对象

  > 描述了增强组件是如何响应拖拽和放置事件的，他是一个包含了若干函数的普通JavaScript对象，这些甘薯会在拖拽交互发生时被调用
  >
  > 对于DragSource 有
  >
  > 1. beginDrag
  > 2. endDrag
  >
  > 对于DropTarget
  >
  > 1. canDrag
  > 2. onDrag

- collect函数

  > DragSource 和 DropTarget 的封装都向其内部组件中提供了属性注入
  >
  > React DnD 并非直接在组件中注入所有的prop属性，而是通过collect 函数来控制哪些属性需要进行注入以及如何进行注入，可以在注入前对属性进行预处理，改变其名称

开始

`shopDnD/Container.js`

```
import React, { Component } from 'react'
// 引入 包裹 目标容器 及 源组件的 最外层高阶组件
import { DragDropContext } from 'react-dnd'
// 当做参数传入
import HTML5Backend from 'react-dnd-html5-backend'
// 目标容器
import ShoppingCart from './ShoppingCart'
// 拖拽源组件
import Snack from './Snack'

class Container extends Component {
  render () {
    return (
      <div>
        <Snack name="Chips" />
        <Snack name="Cupcake" />
        <Snack name="Donut" />
        <Snack name="Doritos" />
```

```
        <Snack name="Popcorn" />
        <ShoppingCart />
      </div>
    )
  }
}

export default DragDropContext(HTML5Backend)(Container)

```

shopDnD/ShoppingCart.js

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// 引入目标容器的高阶组件
import { DropTarget } from 'react-dnd'

// spec 对象 【需要传入的】
const ShoppingCartSpec = {
  drop () {
    return {
      name: 'ShoppingCart'
    }
  }
}

let collect = (connect, monitor) => {
  return {
    // 包裹目标容器的方法
    connectDropTarget: connect.dropTarget(),
    // 可以使用任意的名字
    isOver: monitor.isOver(),
    canDrop: monitor.canDrop()
  }
}

// 目标容器组件
class ShoppingCart extends Component {

  static propTypes = {
    connectDropTarget: PropTypes.func.isRequired,
    isOver: PropTypes.bool.isRequired,
    canDrop: PropTypes.bool.isRequired
  }

  render () {
    // 会通过参数传入进来
    const { canDrop, isOver, connectDropTarget } = this.props
    // 如果已经拖拽完成已经放下
    const isActive = canDrop && isOver

    let backgroundColor = '#FFFFFF'
    if (isActive) {
      backgroundColor = '#F7F7BD'
    } else {
      backgroundColor = '#F7F7F7'
    }
```

```
    const style = {
      backgroundColor: backgroundColor
    }
    return (
      // 拖放目标容器 使用connectDropTarget进行包裹
      connectDropTarget(
        <div className="shopping-cart" style={ style }>
          {
            isActive?
            'Hummmm, snack!': 'Drag here to order!'
          }
        </div>
      )
    )
  }
}

//  传入参数 type spec 对象 collect函数
export default DropTarget("snack", ShoppingCartSpec, collect)(ShoppingCart)

```

shopDnD/Shack.js

```
import React, {Component } from 'react'
import PropTypes from 'prop-types'
// 拖拽源高阶组件
import { DragSource }  from 'react-dnd'

const snackSpec = {
  // 拖拽这个组件开始
  beginDrag(props) {
    return {
      name: props.name
    }
  },
  // 拖拽这个组件结束
  endDrag(props, monitor) {
    // 拖拽的这一项
    const dragItem = monitor.getItem()
    // 拖拽的目标容器
    const dropResult = monitor.getDropResult()

    // 如果目标容易存在
    if (dropResult) {
      console.log(`You dropped ${dragItem.name} into ${dropResult.name}`)
    }
  }
}

let collect = (connect, monitor) => {
  return {
    // 拖拽的这一项 拖拽源头
    connectDragSource: connect.dragSource(),
    // 是否拖动了出来
    isDragging: monitor.isDragging()
  }
}
```

```
35
36  class Snack extends Component {
37    static propTypes = {
38      name: PropTypes.string.isRequired,
39      isDragging: PropTypes.bool.isRequired,
40      connectDragSource: PropTypes.func.isRequired
41    }
42    render () {
43      const { name, isDragging, connectDragSource } = this.props
44
45      // 如果拖动了出来 则改变其透明度
46      let opacity = isDragging? 0.4 : 1
47
48      const style = {
49        opacity: opacity
50      }
51
52      return (
53        // 当成参数传入函数
54        connectDragSource(
55          <div className="snack" style={ style }>
56            {
57              name
58            }
59          </div>
60        )
61      )
62    }
63  }
64
65  export default DragSource('snack', snackSpec, collect)(Snack)
```