---

link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=160 sentences=407, words=3003

---

1. react-transition-group

- React官方提供了react-transition-group (http://reactcommunity.org/react-transition-group/)给一个组件的显示和消失添加过渡动画
- 提供了用于定义进入和退出转换的简单组件

1.1 主要组件

组件 介绍 Transition 该组件与平台▉关(不▉定要结合CSS) CSSTransition 结合CSS，比较常▉ SwitchTransition 两个组件显▉和隐藏切换时使▉该组件 TransitionGroup 将多个动画组件包裹在其中，▉般▉于列表中元素的动画

1.2 安装

```
npm install react-transition-group react-bootstrap bootstrap --save
```

2. Transition

- Transition (http://reactcommunity.org/react-transition-group/transition)组件允许您使用简单的声明式 API描述随时间从一个组件状态到另一个组件状态的转换
- 默认情况下， Transition组件不会改变它呈现的组件的行为，它只跟踪组件的 进入和 退出状态。赋予这些状态以意义和效果取决于您
- 转换可以处于 4种主要状态
    - entering 进入中
    - entered 进入后
    - exiting 离开中
    - exited 离开后
- 过渡状态通过 in属性切换。当为 true时组件开始 进入阶段。在此阶段，组件将从其当前的过渡状态转移到 entering过渡期间，然后在 entered完成后进入该阶段
- in改为 false进行同样的事情，状态从移动 exiting到 exited

2.1 属性

属性名 类型 默认 含义 in boolean false 显示组件；触发进入或退出状态 children Function或element 必需 function可以使用子元素代替 React 元素。此函数使用当前转换状态(entering, entered, exiting,exited)调用，可用于将特定于上下文的属性应用于组件 timeout number 无 过渡的持续时间，以毫秒为单位

2.2 实现原理

- 我们可以向 Transition传递 in和 timeout属性，通过 in来控制组件是否显示，通过 timeout来控制显示或消失的时间间隔
- Transition会自动帮我们管理过渡状态(entering, entered, exiting,exited)
- Transition组件的 children是一个函数，当状态发生改变时，会把新的状态传递给 children函数参数，从而可以根据不同的状态渲染不同的样式

```
exited=>entering=>entered

entered=>exiting=>exited
```

2.3 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import 'bootstrap/dist/css/bootstrap.min.css';
import TransitionPage from './TransitionPage';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <TransitionPage />
);
```

2.4 TransitionPage/index.js

src\TransitionPage\index.js

```
import { useState } from 'react';
import { Container, Button } from 'react-bootstrap';
import { Transition } from '../react-transition-group';

const duration = 1000;

const defaultStyle = {
  opacity: 0,
  transition: `opacity ${duration}ms`
}
const transitionStyles = {
  entering: { opacity: 1 },
  entered: { opacity: 1 },
  exiting: { opacity: 0.1 },
  exited: { opacity: 0.1 }
}
function TransitionPage() {
  const [inProp, setInProp] = useState(false);
  return (
    <Container>
      <Transition in={inProp} timeout={duration}>
        {state => (
          <Button style={{ ...defaultStyle, ...transitionStyles[state] }}>
            {state}
          Button>
        )}
      Transition>
      <br />
      <button onClick={() => setInProp(!inProp)}>
        {inProp ? 'hide' : 'show'}
      button>
    Container>
  );
}
export default TransitionPage;
```

2.5 react-transition-group\index.js

src\react-transition-group\index.js

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

2.6 Transition.js

src\react-transition-group\Transition.js

```
import React from 'react'

export const ENTERING = 'entering'
export const ENTERED = 'entered'
export const EXITING = 'exiting'
export const EXITED = 'exited'

class Transition extends React.Component {
  constructor(props) {
    super(props)
    this.state = {

      status: this.props.in ? ENTERED : EXITED
    }
  }
  componentDidUpdate() {
    let { status } = this.state;
    console.log(status);

    if (this.props.in) {
      if (status !== ENTERING && status !== ENTERED) {
        this.updateStatus(ENTERING)
      }
    } else {
      if (status === ENTERING || status === ENTERED) {
        this.updateStatus(EXITING)
      }
    }
  }

  onTransitionEnd(timeout, callback) {
    if (timeout) {
      setTimeout(callback, timeout)
    }
  }

  performEnter() {
    const { timeout } = this.props
    this.setState({ status: ENTERING }, () => {
      this.onTransitionEnd(timeout, () => {
        this.setState({ status: ENTERED })
      })
    })
  }

  performExit() {
    const { timeout } = this.props
    this.setState({ status: EXITING }, () => {
      this.onTransitionEnd(timeout, () => {
        this.setState({ status: EXITED })
      })
    })
  }

  updateStatus(nextStatus) {
    if (nextStatus) {
      if (nextStatus === ENTERING) {
        this.performEnter()
      } else {
        this.performExit()
      }
    }
  }

  render() {
    const { children } = this.props
    const { status } = this.state
    return (
      children(status)
    )
  }
}

export default Transition
```

3. CSSTransition

- 如果您使用 CSS 过渡或动画，则应该使用它。它建立在Transition (http://reactcommunity.org/react-transition-group/css-transition) 组件之上，因此它继承了它的所有属性
- CSSTransition应用了一对类名在过渡的进场和离场状态

3.1 动画钩子

- Transition在管理组件的生命周期的时候给我们提供了动画钩子
- CSSTransition可以利用这些钩子函数为DOM节点添加类名

钩子名称 钩子含义 onEnter 进场动画开始执行时调用 onEntering 进场动画执行中调用 onEntered 进场动画执行完毕调用 onExit 退场动画开始执行时调用 onExiting 退场动画执行中时调用 onExited 退场动画执行完毕调用

3.2 classNames

- 在组件出现、进入、退出或完成过渡时应用于组件的动画类名
- 可以提供一个名称，每个阶段都会加上后缀
- 例如 classNames="fade"
- 进场时 fade-enter=>(马上)=>fade-enter fade-enter-active=>(1s后)=>fade-enter-done
    - enter 表示开始动画的初始阶段
    - enter-active 表示开始动画的激活阶段
    - enter-done 表示开始动画的结束阶段，也是样式的持久化展示阶段
- 退场时 fade-exit=>(马上)=>fade-exit fade-exit-active=>(1s后)=>fade-exit-done
    - exit 表示开始动画的初始阶段
    - exit-active 表示开始动画的激活阶段
    - exit-done 表示开始动画的结束阶段，也是样式的持久化展示阶段

3.3 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import 'bootstrap/dist/css/bootstrap.min.css';
import TransitionPage from './TransitionPage';
+import CSSTransitionPage from './CSSTransitionPage';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
+
);
```

3.4 CSSTransitionPage\index.js

src\CSSTransitionPage\index.js

```
import { useState } from 'react';
import { Container, Button } from 'react-bootstrap';
import { CSSTransition } from '../react-transition-group';
import './index.css'
const duration = 1000;
function CSSTransitionPage() {
  const [inProp, setInProp] = useState(false);
  return (
    <Container>
      <CSSTransition in={inProp} timeout={duration} classNames="fade">
        <Button className='fade'>
          fade
        Button>
      CSSTransition>
      <br />
      <button onClick={() => setInProp(!inProp)}>
        {inProp ? 'hide' : 'show'}
      button>
    Container>
  );
}
export default CSSTransitionPage;
```

3.4 CSSTransitionPage\index.css

src\CSSTransitionPage\index.css

```
.fade {
  opacity: .1;
}

.fade.fade-enter {
  opacity: .1;
}

.fade.fade-enter-active {
  opacity: 1;
  transition: opacity 1000ms;
}

.fade.fade-enter-done {
  opacity: 1;
}

.fade.fade-exit {
  opacity: 1;
}

.fade.fade-exit-active {
  opacity: .1;
  transition: opacity 1000ms;
}

.fade.fade-exit-done {
  opacity: .1;
}
```

3.6 react-transition-group\index.js

src\react-transition-group\index.js

```
export { default as Transition } from './Transition';
+export { default as CSSTransition } from './CSSTransition';
```

3.7 CSSTransition.js

src\react-transition-group\CSSTransition.js

```
import React from 'react'
import Transition from './Transition'
function CSSTransition(props) {
  const getClassNames = (status) => {
    const { classNames } = props
    return {
      base: `${classNames}-${status}`,
      active: `${classNames}-${status}-active`,
      done: `${classNames}-${status}-done`
    }
  }
  const onEnter = (node) => {
    const exitClassNames = Object.values(getClassNames('exit'));
    reflowAndRemoveClass(node, exitClassNames)
    const enterClassName = getClassNames('enter').base;
    reflowAndAddClass(node, enterClassName)
  }
  const onEntering = (node) => {
    const enteringClassName = getClassNames('enter').active
    reflowAndAddClass(node, enteringClassName)
  }
  const onEntered = (node) => {
    const enteringClassName = getClassNames('enter').active
    const enterClassName = getClassNames('enter').base
    reflowAndRemoveClass(node, [enterClassName, enteringClassName])
    const enteredClassName = getClassNames('enter').done
    reflowAndAddClass(node, enteredClassName)
  }

  const onExit = (node) => {
    const enteredClassNames = Object.values(getClassNames('enter'))
    reflowAndRemoveClass(node, enteredClassNames)
    const exitClassName = getClassNames('exit').base
    reflowAndAddClass(node, exitClassName)
  }

  const onExiting = (node) => {
    const exitingClassName = getClassNames('exit').active
    reflowAndAddClass(node, exitingClassName, true)
  }
  const onExited = (node) => {
    const exitingClassName = getClassNames('exit').active
    const exitClassName = getClassNames('exit').base
    reflowAndRemoveClass(node, [exitClassName, exitingClassName])
    const exitedClassName = getClassNames('exit').done
    reflowAndAddClass(node, exitedClassName)
  }
  return (
    <Transition
      onEnter={onEnter}
      onEntering={onEntering}
      onEntered={onEntered}
      onExit={onExit}
      onExiting={onExiting}
      onExited={onExited}
      in={props.in}
      timeout={props.timeout}
    >
      {props.children}
    Transition>
  )
}
export default CSSTransition;

function reflowAndAddClass(node, classes) {
  node.offsetWidth && (Array.isArray(classes) ? classes : [classes]).forEach((className) => node.classList.add(className))
}
function reflowAndRemoveClass(node, classes) {
  node.offsetWidth && (Array.isArray(classes) ? classes : [classes]).forEach((className) => node.classList.remove(className))
}
```

3.8 Transition.js

src\react-transition-group\Transition.js

```
import React from 'react'
+import ReactDOM from 'react-dom';
export const ENTERING = 'entering'//进入中
export const ENTERED = 'entered'//进入后
export const EXITING = 'exiting'//退出中
export const EXITED = 'exited'//退出后

class Transition extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      //过渡的状态
      status: this.props.in ? ENTERED : EXITED
    }
  }
  componentDidUpdate() {
    let { status } = this.state;
    console.log(status);
    //更新后当属性发生改变时更改状态
    if (this.props.in) {//in为true时执行进场动画
      if (status !== ENTERING && status !== ENTERED) {
        this.updateStatus(ENTERING)
      }
    } else {//in为false时执行离场动画
      if (status
        this.updateStatus(EXITING)
      }
    }
  }

  onTransitionEnd(timeout, callback) {
    if (timeout) {
      setTimeout(callback, timeout)
    }
  }

  performEnter() {
+   const { timeout, onEnter, onEntering, onEntered } = this.props
+   const node = ReactDOM.findDOMNode(this)
+   onEnter?.(node)
    this.setState({ status: ENTERING }, () => {
+     onEntering?.(node)
      this.onTransitionEnd(timeout, () => {
+       this.setState({ status: ENTERED }, () => onEntered?.(node))
      })
    })
  }

  performExit() {
+   const { timeout, onExit, onExiting, onExited } = this.props
+   const node = ReactDOM.findDOMNode(this)
+   onExit?.(node)
    this.setState({ status: EXITING }, () => {
+     onExiting?.(node)
      this.onTransitionEnd(timeout, () => {
+       this.setState({ status: EXITED }, () => onExited?.(node))
      })
    })
  }

  updateStatus(nextStatus) {
    if (nextStatus) {
      if (nextStatus
        this.performEnter()
      } else {
        this.performExit()
      }
    }
  }

  render() {
    const { children } = this.props
    const { status } = this.state
    return (
+     typeof children === 'function' ? children(status) : children
    )
  }
}
export default Transition
```

4. SwitchTransition

- 当您想控制状态转换之间的渲染时，可以使用它
- 在两个组件切换的时候会等待上一个组件离场以后再触发另一个组件的进场动画

4.1 实现原理

- 首次渲染把 `children`保存到 `state`的 `current`属性上并进行渲染
- 当组件从A切换到B的时候，在getDerivedStateFromProps把状态更新为 `EXITING`,此时继续渲染A组件并触发A组件的离场动画
- 当A组件离场对画结束后修改状态为 `ENTERING`并且清空 `state.current`,再次渲染B组件，并触发B组件的进场动画,动画结束后状态变为 `ENTERED`
- getDerivedStateFromProps
    - `getDerivedStateFromProps` 的作用就是为了让 `props` 能更新到组件内部 `state` 中
    - 首次挂载顺序 constructor=>getDerivedStateFromProps=>render=>componentDidMount
    - 更新时顺序 getDerivedStateFromProps=>render=>componentDidUpdate

4.2 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import 'bootstrap/dist/css/bootstrap.min.css';
import TransitionPage from './TransitionPage';
import CSSTransitionPage from './CSSTransitionPage';
+import SwitchTransitionPage from './SwitchTransitionPage';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
+
);
```

4.3 SwitchTransitionPage/index.js

src\SwitchTransitionPage/index.js

```
import { useState } from 'react';
import { SwitchTransition, CSSTransition } from '../react-transition-group';
import './index.css'
function SwitchTransitionPage() {
  const [state, setState] = useState(true);
  const buttonStyle = { width: '200px', height: '80px', fontSize: '40px', border: 'none', backgroundColor: 'green', color: 'white' }
  return (
    <SwitchTransition>
      <CSSTransition
        key={state}
        timeout={2000}
        classNames="panel"
      >
        <button style={buttonStyle} onClick={() => setState((state) => !state)}>
          {state ? "A" : "B"}
        button>
      CSSTransition>
    SwitchTransition>
  );
}
export default SwitchTransitionPage;
```

4.4 SwitchTransitionPage\index.css

src\SwitchTransitionPage\index.css

```
.panel {
  opacity: 0;
}

.panel-enter {
  opacity: 0;
  transform: translateX(-100%);
}

.panel-enter-active {
  opacity: 1;
  transform: translateX(0%);
  transition: opacity 2000ms, transform 2000ms;
}

.panel-enter-done {
  opacity: 1;
}

.panel-exit {
  opacity: 1;
  transform: translateX(0%);
}

.panel-exit-active {
  opacity: 0;
  transform: translateX(100%);
  transition: opacity 2000ms, transform 2000ms;
}

.panel-exit-done {
  opacity: 0;
}
```

4.5 react-transition-group\index.js

src\react-transition-group\index.js

```
export { default as Transition } from './Transition';
export { default as CSSTransition } from './CSSTransition';
+export { default as SwitchTransition } from './SwitchTransition';
```

4.7 SwitchTransition.js

src\react-transition-group\SwitchTransition.js

```
import React, { isValidElement, Component } from 'react';
import { ENTERING, ENTERED, EXITING } from './Transition';
import TransitionGroupContext from './TransitionGroupContext';
class SwitchTransition extends Component {
  constructor(props) {
    super(props)
    this.state = {
      status: ENTERED,
      current: null,
    }
    this.mounted = false
  }
  componentDidMount() {
    this.mounted = true
  }
  static getDerivedStateFromProps(props, state) {
    if (state.current && areChildrenDifferent(state.current, props.children)) {
      return {
        status: EXITING
      }
    }
    return {
      current: React.cloneElement(props.children, { in: true })
    }
  }
  changeState = (status, current = this.state.current) => {
    this.setState({ status, current })
  }
  render() {
    const { status, current } = this.state;
    const { children } = this.props;
    let component
    switch (status) {
      case ENTERING:
        component = React.cloneElement(children, {
          in: true,
          onEntered: () => {
            this.changeState(ENTERED)
          }
        })
        break
      case EXITING:
        component = React.cloneElement(current, {
          in: false,
          onExited: () => {
            this.changeState(ENTERING, null)
          }
        })
        break
      case ENTERED:
        component = current
        break
    }
    return (
      <TransitionGroupContext.Provider value={{ status: this.mounted ? ENTERING : ENTERED }}>
        {component}
      TransitionGroupContext.Provider>
    )
  }
}
function areChildrenDifferent(oldChildren, newChildren) {
  if (oldChildren === newChildren) return false;
  if (
    isValidElement(oldChildren) &&
    isValidElement(newChildren) &&
    oldChildren.key !== null &&
    oldChildren.key === newChildren.key
  ) {
    return false;
  }
  return true;
}
export default SwitchTransition;
```

4.8 CSSTransition.js

src\react-transition-group\CSSTransition.js

```
import React from 'react'
import Transition from './Transition'
function CSSTransition(props) {
  const getClassNames = (status) => {
    const { classNames } = props
    return {
      base: `${classNames}-${status}`,
      active: `${classNames}-${status}-active`,
      done: `${classNames}-${status}-done`
    }
  }
  const onEnter = (node) => {
    const exitClassNames = Object.values(getClassNames('exit'));//['fade-exit','fade-exit-active','fade-exit-done']
    reflowAndRemoveClass(node, exitClassNames)
    const enterClassName = getClassNames('enter').base;//fade-enter
    reflowAndAddClass(node, enterClassName)
  }
  const onEntering = (node) => {
    const enteringClassName = getClassNames('enter').active//fade-enter-active
    reflowAndAddClass(node, enteringClassName)
  }
  const onEntered = (node) => {
    const enteringClassName = getClassNames('enter').active//fade-enter-active
    const enterClassName = getClassNames('enter').base//fade-enter
    reflowAndRemoveClass(node, [enterClassName, enteringClassName])
    const enteredClassName = getClassNames('enter').done//fade-enter-done
    reflowAndAddClass(node, enteredClassName)
+   props.onEntered?.(node)
  }

  const onExit = (node) => {
    const enteredClassNames = Object.values(getClassNames('enter'))
    reflowAndRemoveClass(node, enteredClassNames)
    const exitClassName = getClassNames('exit').base
    reflowAndAddClass(node, exitClassName)
  }

  const onExiting = (node) => {
    const exitingClassName = getClassNames('exit').active
    reflowAndAddClass(node, exitingClassName, true)
  }
  const onExited = (node) => {
    const exitingClassName = getClassNames('exit').active
    const exitClassName = getClassNames('exit').base
    reflowAndRemoveClass(node, [exitClassName, exitingClassName])
    const exitedClassName = getClassNames('exit').done
    reflowAndAddClass(node, exitedClassName)
+   props.onExited?.(node)
  }
  return (

      {props.children}

  )
}
export default CSSTransition;

function reflowAndAddClass(node, classes) {
  //强制浏览器重绘
  node.offsetLeft && (Array.isArray(classes) ? classes : [classes]).forEach((className) => node.classList.add(className))
}
function reflowAndRemoveClass(node, classes) {
  node.offsetLeft && (Array.isArray(classes) ? classes : [classes]).forEach((className) => node.classList.remove(className))
}
```

4.9 Transition.js

src\react-transition-group\Transition.js

```
import React from 'react'
import ReactDOM from 'react-dom';
+import TransitionGroupContext from './TransitionGroupContext';
export const ENTERING = 'entering'//进入中
export const ENTERED = 'entered'//进入后
export const EXITING = 'exiting'//退出中
export const EXITED = 'exited'//退出后

class Transition extends React.Component {
+ static contextType = TransitionGroupContext
  constructor(props) {
    super(props)
    this.state = {
      //过渡的状态
      status: this.props.in ? ENTERED : EXITED
    }
  }
+ componentDidMount() {
+   if (this.context) {
+     const { status } = this.context
+     if (status === ENTERING) {
+       this.updateStatus(status)
+     }
+   }
+ }
  componentDidUpdate() {
    let { status } = this.state;
    //更新后当属性发生改变时更改状态
    if (this.props.in) {//in为true时执行进场动画
      if (status !== ENTERING && status !== ENTERED) {
        this.updateStatus(ENTERING)
      }
    } else {//in为false时执行离场动画
      if (status
        this.updateStatus(EXITING)
      }
    }
  }

  onTransitionEnd(timeout, callback) {
    if (timeout) {
      setTimeout(callback, timeout)
    }
  }

  performEnter() {
    const { timeout, onEnter, onEntering, onEntered } = this.props
    const node = ReactDOM.findDOMNode(this)
    onEnter?.(node)
    this.setState({ status: ENTERING }, () => {
      onEntering?.(node)
      this.onTransitionEnd(timeout, () => {
        this.setState({ status: ENTERED }, () => onEntered?.(node))
      })
    })
  }

  performExit() {
    const { timeout, onExit, onExiting, onExited } = this.props
    const node = ReactDOM.findDOMNode(this)
    onExit?.(node)
    this.setState({ status: EXITING }, () => {
      onExiting?.(node)
      this.onTransitionEnd(timeout, () => {
        this.setState({ status: EXITED }, () => onExited?.(node))
      })
    })
  }

  updateStatus(nextStatus) {
    if (nextStatus) {
      if (nextStatus
        this.performEnter()
      } else {
        this.performExit()
      }
    }
  }

  render() {
    const { children } = this.props
    const { status } = this.state
    return (
      typeof children
    )
  }
}

export default Transition
```

4.10 TransitionGroupContext.js

src\react-transition-group\TransitionGroupContext.js

```
import React from 'react';
export default React.createContext(null);
```

5. TransitionGroup

- TransitionGroup (http://reactcommunity.org/react-transition-group/transition-group)组件管理列表中的一组转换组件Transition和CSSTransition
- 与过渡组件一样， TransitionGroup它是一个状态机，用于随时间管理组件的安装和卸载

5.1 原理

- 该组件主要用来给一组元素添加进场和离场动画
- 原理是用新的 children和上次的 children进行对比，如果是增加元素就先添加进场动画，如果是删除元素就添加离场动画，等动画结束后再去进行真正的挂载和卸载操作

5.2 src\index.js

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import 'bootstrap/dist/css/bootstrap.min.css';
import TransitionPage from './TransitionPage';
import CSSTransitionPage from './CSSTransitionPage';
import SwitchTransitionPage from './SwitchTransitionPage';
+import TransitionGroupPage from './TransitionGroupPage';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
+
);
```

5.3 TransitionGroupPage\index.js

src\TransitionGroupPage\index.js

```
import React, { useState } from 'react';
import { Container, ListGroup, Button, } from 'react-bootstrap';
import { CSSTransition, TransitionGroup, } from '../react-transition-group';
import './index.css';
function TransitionGroupPage() {
  const [items, setItems] = useState([
    { id: '1', text: '吃饭' },
    { id: '2', text: '睡觉' },
    { id: '3', text: '打豆豆' }
  ]);
  return (
    <Container>
      <ListGroup>
        <TransitionGroup>
          {items.map(({ id, text }) => (
            <CSSTransition
              key={id}
              timeout={1000}
              classNames="item"
            >
              <ListGroup.Item>
                <Button
                  style={{ marginRight: '10px' }}
                  onClick={() =>
                    setItems(items =>
                      items.filter(item => item.id !== id)
                    )
                  }
                >
                  ×
                Button>
                {text}
              ListGroup.Item>
            CSSTransition>
          ))}
        TransitionGroup>
      ListGroup>
      <Button
        onClick={() => {
          setItems(items => [
            ...items,
            { id: Date.now(), text: items.length },
          ]);
        }}
      >
        Add Item
      Button>
    Container>
  );
}

export default TransitionGroupPage;
```

5.4 TransitionGroupPage\index.css

src\TransitionGroupPage\index.css

```
.item-enter {
  opacity: 0;
}

.item-enter-active {
  opacity: 1;
  transition: opacity 1000ms;
}

.item-exit {
  opacity: 1;
}

.item-exit-active {
  opacity: 0;
  transition: opacity 1000ms;
}
```

5.5 react-transition-group\index.js

src\react-transition-group\index.js

```
export { default as Transition } from './Transition';
export { default as CSSTransition } from './CSSTransition';
export { default as SwitchTransition } from './SwitchTransition';
+export { default as TransitionGroup } from './TransitionGroup';
```

5.6 TransitionGroup.js

src\react-transition-group\TransitionGroup.js

```
import React, { cloneElement } from 'react';
import TransitionGroupContext from './TransitionGroupContext';
import { ENTERING, ENTERED } from './Transition';
class TransitionGroup extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      children: {},
      status: ENTERED,
      firstRender: true,
      handleExited: this.handleExited
    };
  }
  static getDerivedStateFromProps(nextProps, { children, firstRender, handleExited }) {
    return {
      children: firstRender
        ? getInitialChildrenMapping(nextProps.children)
        : getNextChildrenMapping(nextProps, children, handleExited),
      firstRender: false
    };
  }

  componentDidMount() {
    this.setState({
      status: ENTERING
    });
  }

  handleExited = (child) => {
    this.setState((state) => {
      const children = { ...state.children };
      delete children[child.key];
      return { children };
    });
  }
  render() {
    const { children, status } = this.state;
    const component = Object.values(children);
    return (
      <TransitionGroupContext.Provider value={{ status }}>
        {component}
      TransitionGroupContext.Provider>
    );
  }
}

export default TransitionGroup;
function getChildrenMapping(children, mapFn = (c) => c) {
  const result = Object.create(null);
  React.Children.forEach(children, (c) => {
    result[c.key] = mapFn(c);
  });
  return result;
}

function getInitialChildrenMapping(children) {
  return getChildrenMapping(children, (c) => cloneElement(c, { in: true }));
}

function mergeChildMappings(prev, next) {
  return Object.keys(prev).length > Object.keys(next).length ? prev : next;
}

function getNextChildrenMapping(nextProps, prevChildrenMapping, handleExited) {
  const result = Object.create(null);
  const nextChildrenMapping = getChildrenMapping(nextProps.children);
  const mergeMappings = mergeChildMappings(prevChildrenMapping, nextChildrenMapping);
  Object.keys(mergeMappings).forEach((key) => {
    const isNext = key in nextChildrenMapping;
    const isPrev = key in prevChildrenMapping;

    if (!isPrev && isNext) {
      result[key] = React.cloneElement(nextChildrenMapping[key], { in: true });
    }

    if (isPrev && !isNext) {
      result[key] = React.cloneElement(prevChildrenMapping[key], {
        in: false,
        onExited() {
          handleExited(prevChildrenMapping[key]);
        },
      });
    }

    if (isNext && isPrev) {
      result[key] = React.cloneElement(nextChildrenMapping[key], { in: true });
    }
  });
  return result;
}
```