React Transition Group

Getting Started

Installation

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

# yarn
yarn add react-transition-group
```

CDN / External

Since react-transition-group is fairly small, the overhead of including the library in your application is negligible. However, in situations where it may be useful to benefit from an external CDN when bundling, link to the following CDN: https://unpkg.com/react-transition-group/dist/react-transition-group.min.js

Components

Transition

The Transition component lets you describe a transition from one component state to another *over time* with a simple declarative API. Most commonly it's used to animate the mounting and unmounting of a component, but can also be used to describe in-place transition states as well.

By default the `Transition` component does not alter the behavior of the component it renders, it only tracks "enter" and "exit" states for the components. It's up to you to give meaning and effect to those states. For example we can add styles to a component when it enters or exits:

```
import Transition from 'react-transition-group/Transition';

const duration = 300;

const defaultStyle = {
  transition: `opacity ${duration}ms ease-in-out`,
  opacity: 0,
}

const transitionStyles = {
  entering: { opacity: 1 },
  entered:  { opacity: 1 },
};

const Fade = ({ in: inProp }) => (
  <Transition in={inProp} timeout={duration}>
    {(state) => (
      <div style={{
        ...defaultStyle,
        ...transitionStyles[state]
      }}>
        I'm A fade Transition!
      </div>
    )}
  </Transition>
);
```

As noted the `Transition` component doesn't *do* anything by itself to its child component. What it does do is track transition states over time so you can update the component (such as by adding styles or classes) when it changes states.

There are 4 main states a Transition can be in:

- `ENTERING`
- `ENTERED`
- `EXITING`
- `EXITED`

Transition state is toggled via the `in` prop. When `true` the component begins the "Enter" stage. During this stage, the component will shift from its current transition state, to `'entering'` for the duration of the transition and then to the `'entered'` stage once it's complete. Let's take the following example:

```
state= { in: false };

toggleEnterState = () => {
  this.setState({ in: true });
}

render() {
  return (
    <div>
      <Transition in={this.state.in} timeout={500} />
      <button onClick={this.toggleEnterState}>Click to Enter</button>
    </div>
  );
}
```

When the button is clicked the component will shift to the `'entering'` state and stay there for 500ms (the value of `timeout`) when finally switches to `'entered'`.

When `in` is `false` the same thing happens except the state moves from `'exiting'` to `'exited'`.

Props

`children`

A `function` child can be used instead of a React element. This function is called with the current transition status ('entering', 'entered', 'exiting', 'exited', 'unmounted'), which can used to apply context specific props to a component.

```
<Transition timeout={150}>
  {(status) => (
    <MyComponent className={`fade fade-${status}`} />
  )}
</Transition>
```

type: `Function | element`
required

`in`

Show the component; triggers the enter or exit states

type: `boolean`
default: `false`

`mountOnEnter`

By default the child component is mounted immediately along with the parent `Transition` component. If you want to "lazy mount" the component on the first `in={true}` you can set `mountOnEnter` . After the first enter transition the component will stay mounted, even on "exited", unless you also specify `unmountOnExit` .

type: `boolean`
default: `false`

`unmountOnExit`

By default the child component stays mounted after it reaches the `'exited'` state. Set `unmountOnExit` if you'd prefer to unmount the component after it finishes exiting.

type: `boolean`
default: `false`

`appear`

Normally a component is not transitioned if it shown when the `<Transition>` component mounts. If you want to transition on the first mount set `appear` to `true` , and the component will transition in as soon as the `<Transition>` mounts.

> Note: there are no specific "appear" states. `apprear` only an additional `enter` transition.

type: `boolean`
default: `false`

`enter`

Enable or disable enter transitions.

type: `boolean`
default: `true`

`exit`

Enable or disable exit transitions.

type: `boolean`
default: `true`

`timeout`

The duration for the transition, in milliseconds.

You may specify a single timeout for all transitions like: `timeout={500}` , or individually like:

```
timeout={{
 enter: 300,
 exit: 500,
}}
```

type: `number | { enter?: number, exit?: number }`

`addEndListener`

Add a custom transition end trigger. Called with the transitioning DOM node and a `done` callback. Allows for more fine grained transition end logic. **Note:** Timeouts are still used as a fallback.

```
addEndListener={(node, done) => {
  // use the css transitionend event to mark the finish of a transition
  node.addEventListener('transitionend', done, false);
}}
```

type: `Function`

`onEnter`

Callback fired before the "entering" status is applied. An extra parameter `isAppearing` is supplied to indicate if the enter stage is occuring on the initial mount

type: `Function(node: HtmlElement, isAppearing: bool) -> void`
default: `function noop() {}`

`onEntering`

Callback fired after the "entering" status is applied. An extra parameter `isAppearing` is supplied to indicate if the enter stage is occuring on the initial mount

type: `Function(node: HtmlElement, isAppearing: bool)`
default: `function noop() {}`

`onEntered`

Callback fired after the "enter" status is applied. An extra parameter `isAppearing` is supplied to indicate if the enter stage is occuring on the initial mount

type: `Function(node: HtmlElement, isAppearing: bool) -> void`
default: `function noop() {}`

`onExit`

Callback fired before the "exiting" status is applied.

type: `Function(node: HtmlElement) -> void`
default: `function noop() {}`

`onExiting`

Callback fired after the "exiting" status is applied.

type: `Function(node: HtmlElement) -> void`
default: `function noop() {}`

`onExited`

Callback fired after the "exited" status is applied.

type: `Function(node: HtmlElement) -> void`
default: `function noop() {}`

TransitionGroup

The `<TransitionGroup>` component manages a set of `<Transition>` components in a list. Like with the `<Transition>` component, `<TransitionGroup>`, is a state machine for managing the mounting and unmounting of components over time.

Consider the example below using the `Fade` CSS transition from before. As items are removed or added to the TodoList the `in` prop is toggled automatically by the `<TransitionGroup>`. You can use *any* `<Transition>` component in a `<TransitionGroup>`, not just css.

```
import TransitionGroup from 'react-transition-group/TransitionGroup';

class TodoList extends React.Component {
  constructor(props) {
    super(props)
    this.state = {items: ['hello', 'world', 'click', 'me']}
  }
  handleAdd() {
    const newItems = this.state.items.concat([
      prompt('Enter some text')
    ]);
    this.setState({ items: newItems });
  }
  handleRemove(i) {
    let newItems = this.state.items.slice();
    newItems.splice(i, 1);
    this.setState({items: newItems});
  }
  render() {
    return (
      <div>
        <button onClick={() => this.handleAdd()}>Add Item</button>
        <TransitionGroup>
          {this.state.items.map((item, i) => (
            <FadeTransition key={item}>
              <div>
                {item}{' '}
                <button onClick={() => this.handleRemove(i)}>
                  remove
                </button>
              </div>
            </FadeTransition>
          ))}
        </TransitionGroup>
      </div>
    );
  }
}
```

Note that `<TransitionGroup>` does not define any animation behavior! Exactly *how* a list item animates is up to the individual `<Transition>` components. This means you can mix and match animations across different list items.

Props

`component`

`<TransitionGroup>` renders a `<div>` by default. You can change this behavior by providing a `component` prop.

type: `any`
default: `'div'`

`children`

A set of `<Transition>` components, that are toggled `in` and out as they leave. the `<TransitionGroup>` will inject specific transition props, so remember to spread them throguh if you are wrapping the `<Transition>` as with our `<Fade>` example.

type: `any`

`appear`

A convenience prop that enables or disabled appear animations for all children. Note that specifiying this will override any defaults set on individual children Transitions.

type: `boolean`

`enter`

A convenience prop that enables or disabled enter animations for all children. Note that specifiying this will override any defaults set on individual children Transitions.

type: `boolean`

`exit`

A convenience prop that enables or disabled exit animations for all children. Note that specifiying this will override any defaults set on individual children Transitions.

type: `boolean`

`childFactory`

You may need to apply reactive updates to a child as it is exiting. This is generally done by using `cloneElement` however in the case of an exiting child the element has already been removed and not accessible to the consumer.

If you do need to update a child as it leaves you can provide a `childFactory` to wrap every child, even the ones that are leaving.

type: `Function(child: ReactElement) -> ReactElement`
default: `child => child`

CSSTransition

A `Transition` component using CSS transitions and animations. It's inspired by the excellent ng-animate libary.

`CSSTransition` applies a pair of class names during the `appear`, `enter`, and `exit` stages of the transition. The first class is applied and then a second "active" class in order to activate the css animation.

When the `in` prop is toggled to `true` the Component will get the `example-enter` CSS class and the `example-enter-active` CSS class added in the next tick. This is a convention based on the `classNames` prop.

```
import CSSTransition from 'react-transition-group/CSSTransition';

const Fade = ({ children, ...props }) => (
 <CSSTransition
   {...props}
   timeout={500}
   classNames="fade"
 >
  {children}
 </CSSTransition>
);

class FadeInAndOut extends React.Component {
  constructor(...args) {
    super(...args);
    this.state= { show: false }

    setInterval(() => {
      this.setState({ show: !this.state.show })
    }, 5000)
  }
  render() {
    return (
      <Fade in={this.state.show}>
        <div>Hello world</div>
      </Fade>
    )
  }
}
```

And the coorresponding CSS for the `<Fade>` component:

```
.fade-enter {
  opacity: 0.01;
}

.fade-enter.fade-enter-active {
  opacity: 1;
  transition: opacity 500ms ease-in;
}

.fade-exit {
  opacity: 1;
}

.fade-exit.fade-exit-active {
  opacity: 0.01;
  transition: opacity 300ms ease-in;
}
```

Props

Accepts all props from <Transition> unless otherwise noted.

`classNames`

The animation classNames applied to the component as it enters or exits. A single name can be provided and it will be suffixed for each stage: e.g.

`classNames="fade"` applies `fade-enter`, `fade-enter-active`, `fade-exit`, `fade-exit-active`, `fade-appear`, and `fade-appear-active`. Each individual classNames can also be specified independently like:

```
classNames={{
 appear: 'my-appear',
 appearActive: 'my-active-appear',
 enter: 'my-enter',
 enterActive: 'my-active-enter',
 exit: 'my-exit',
 exitActive: 'my-active-exit',
}}
```

type: `{ appear?: string, appearActive?: string, enter?: string, enterActive?: string, exit?: string, exitActive?: string, }`

`onEnter`

A `<Transition>` callback fired immediately after the 'enter' or 'appear' class is applied.

type: `Function(node: HtmlElement, isAppearing: bool)`

`onEntering`

A `<Transition>` callback fired immediately after the 'enter-active' or 'appear-active' class is applied.

type: `Function(node: HtmlElement, isAppearing: bool)`

`onEntered`

A `<Transition>` callback fired immediately after the 'enter' or 'appear' classes are **removed** from the DOM node.

type: `Function(node: HtmlElement, isAppearing: bool)`

`onExit`

A `<Transition>` callback fired immediately after the 'exit' class is applied.

type: `Function(node: HtmlElement)`

`onExiting`

A `<Transition>` callback fired immediately after the 'exit-active' is class is applied.

type: `Function(node: HtmlElement`

`onExited`

A `<Transition>` callback fired immediately after the 'exit' classes are **removed** from the DOM node.

type: `Function(node: HtmlElement)`