React.js 小书

<-- 返回首页

高阶组件（Higher-Order Components）

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson28

（本文未审核）

有时候人们很喜欢造一些名字很吓人的名词，让人一听这个名词就觉得自己不可能学会，从而让人望而却步。但是其实这些名词背后所代表的东西其实很简单。

我不能说高阶组件就是这么一个东西。但是它是一个概念上很简单，但却非常常用、实用的东西，被大量 React.js 相关的第三方库频繁地使用。在前端的业务开发当中，你不掌握高阶组件其实也可以完成项目的开发，但是如果你能够灵活地使用高阶组件，可以让你代码更加优雅，复用性、灵活性更强。它是一个加分项，而且加的分还不少。

本章节可能有部分内容理解起来会有难度，如果你觉得无法完全理解本节内容。可以先简单理解高阶组件的概念和作用即可，其他内容选择性地跳过。

了解高阶组件对我们理解各种 React.js 第三方库的原理很有帮助。

什么是高阶组件

高阶组件就是一个函数，传给它一个组件，它返回一个新的组件。

```
const NewComponent = higherOrderComponent(OldComponent)
```

重要的事情再重复一次，高阶组件是一个函数（而不是组件），它接受一个组件作为参数，返回一个新的组件。这个新的组件会使用你传给它的组件作为子组件，我们看看一个很简单的高阶组件：

```
import React, { Component } from 'react'

export default (WrappedComponent) => {
  class NewComponent extends Component {
    // 可以做很多自定义逻辑
    render () {
      return <WrappedComponent />
    }
  }
```

```
  return NewComponent
}
```

现在看来好像什么用都没有，它就是简单的构建了一个新的组件类 `NewComponent`，然后把传进入去的 `WrappedComponent` 渲染出来。但是我们可以给 `NewCompoent` 做一些数据启动工作：

```
import React, { Component } from 'react'

export default (WrappedComponent, name) => {
  class NewComponent extends Component {
    constructor () {
      super()
      this.state = { data: null }
    }

    componentWillMount () {
      let data = localStorage.getItem(name)
      this.setState({ data })
    }

    render () {
      return <WrappedComponent data={this.state.data} />
    }
  }
  return NewComponent
}
```

现在 `NewComponent` 会根据第二个参数 `name` 在挂载阶段从 LocalStorage 加载数据，并且 `setState` 到自己的 `state.data` 中，而渲染的时候将 `state.data` 通过 `props.data` 传给 `WrappedComponent`。

这个高阶组件有什么用呢？假设上面的代码是在 `src/wrapWithLoadData.js` 文件中的，我们可以在别的地方这么用它：

```
import wrapWithLoadData from './wrapWithLoadData'

class InputWithUserName extends Component {
  render () {
    return <input value={this.props.data} />
  }
}

InputWithUserName = wrapWithLoadData(InputWithUserName, 'username')
export default InputWithUserName
```

假如 `InputWithUserName` 的功能需求是挂载的时候从 LocalStorage 里面加载 `username` 字段作为 `<input />` 的 `value` 值，现在有了 `wrapWithLoadData`，我们可以很容易地做到这件事情。

只需要定义一个非常简单的 `InputWithUserName`，它会把 `props.data` 作为 `<input />` 的 `value` 值。然把这个组件和 `'username'` 传给 `wrapWithLoadData`，`wrapWithLoadData` 会返回一个新的组件，我们用这个新的组件覆盖原来的 `InputWithUserName`，然后再导出去模块。

别人用这个组件的时候实际是用了**被加工过**的组件：

```
import InputWithUserName from './InputWithUserName'

class Index extends Component {
  render () {
    return (
      <div>
        用户名：<InputWithUserName />
      </div>
    )
  }
}
```

根据 `wrapWithLoadData` 的代码我们可以知道，这个新的组件挂载的时候会先去 LocalStorage 加载数据，渲染的时候再通过 `props.data` 传给真正的 `InputWithUserName`。

如果现在我们需要另外一个文本输入框组件，它也需要 LocalStorage 加载 `'content'` 字段的数据。我们只需要定义一个新的 `TextareaWithContent`：

```
import wrapWithLoadData from './wrapWithLoadData'

class TextareaWithContent extends Component {
  render () {
    return <textarea value={this.props.data} />
  }
}

TextareaWithContent = wrapWithLoadData(TextareaWithContent, 'content')
export default TextareaWithContent
```

写起来非常轻松，我们根本不需要重复写从 LocalStorage 加载数据字段的逻辑，直接用 `wrapWithLoadData` 包装一下就可以了。

我们来回顾一下到底发生了什么事情，对于 `InputWithUserName` 和 `TextareaWithContent` 这两个组件来说，它们的需求有着这么一个相同的逻辑：“挂载阶段从 LocalStorage 中加载特定字段数据”。

如果按照之前的做法，我们需要给它们两个都加上 `componentWillMount` 生命周期，然后在里面调用 LocalStorage。要是有第三个组件也有这样的加载逻辑，我又得写一遍这样的逻辑。但有了 `wrapWithLoadData` 高阶组件，我们把这样的逻辑用一个组件

包裹了起来，并且通过给高阶组件传入 `name` 来达到不同字段的数据加载。充分复用了逻辑代码。

到这里，高阶组件的作用其实不言而喻，**其实就是为了组件之间的代码复用**。组件可能有着某些相同的逻辑，把这些逻辑抽离出来，放到高阶组件中进行复用。**高阶组件内部的包装组件和被包装组件之间通过 `props` 传递数据**。

高阶组件的灵活性

代码复用的方法、形式有很多种，你可以用类继承来做到代码复用，也可以分离模块的方式。但是高阶组件这种方式很有意思，也很灵活。学过设计模式的同学其实应该能反应过来，它其实就是设计模式里面的装饰者模式。它通过组合的方式达到很高的灵活程度。

假设现在我们需求变化了，现在要的是通过 Ajax 加载数据而不是从 LocalStorage 加载数据。我们只需要新建一个 `wrapWithAjaxData` 高阶组件：

```
import React, { Component } from 'react'

export default (WrappedComponent, name) => {
  class NewComponent extends Component {
    constructor () {
      super()
      this.state = { data: null }
    }

    componentWillMount () {
      ajax.get('/data/' + name, (data) => {
        this.setState({ data })
      })
    }

    render () {
      return <WrappedComponent data={this.state.data} />
    }
  }
  return NewComponent
}
```

其实就是改了一下 `wrapWithLoadData` 的 `componentWillMount` 中的逻辑，改成了从服务器加载数据。现在只需要把 `InputWithUserName` 稍微改一下：

```
import wrapWithAjaxData from './wrapWithAjaxData'

class InputWithUserName extends Component {
  render () {
    return <input value={this.props.data} />
  }
}
```

```
InputWithUserName = wrapWithAjaxData(InputWithUserName, 'username')
export default InputWithUserName
```

只要改一下包装的高阶组件就可以达到需要的效果。而且我们并没有改动 `InputWithUserName` 组件内部的任何逻辑，也没有改动 `Index` 的任何逻辑，只是改动了中间的高阶组件函数。

（以下内容为选读内容，有兴趣的同学可以继续往下读，否则也可以直接跳到文末的总结部分。）

多层高阶组件（选读）

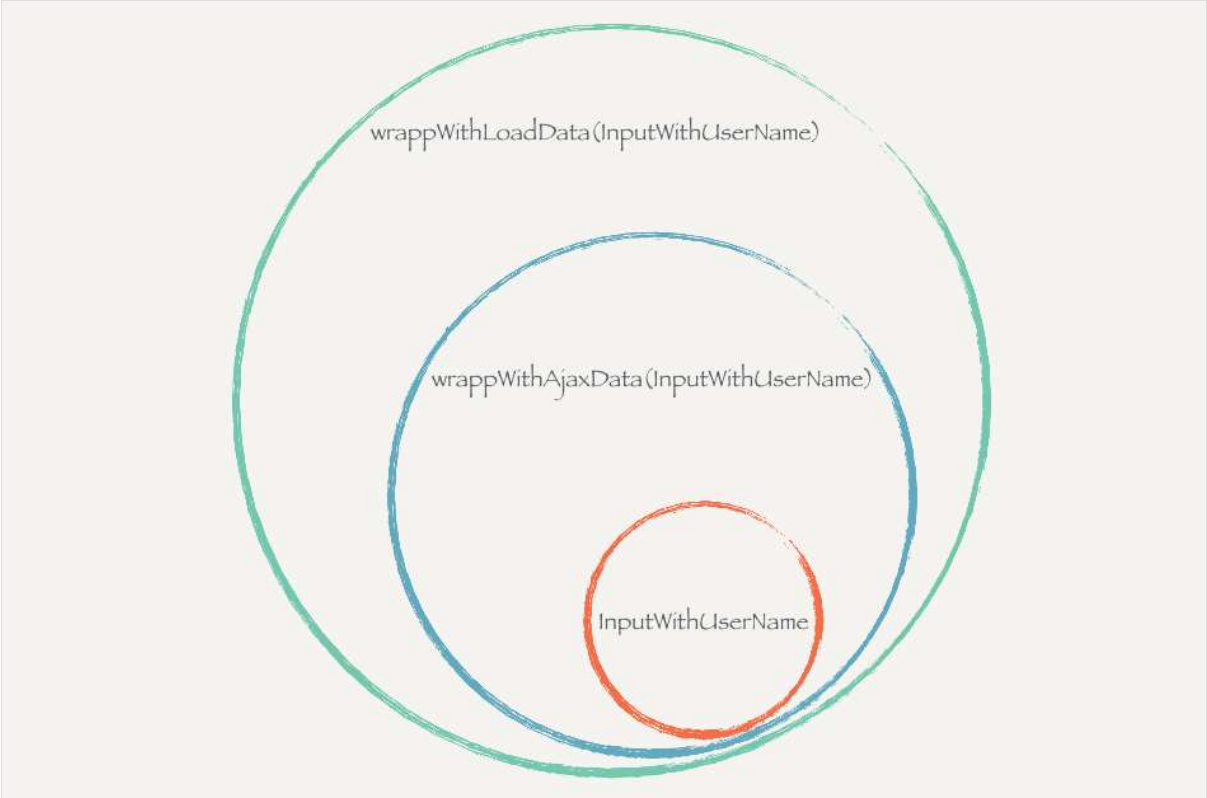
假如现在需求有变化了：我们需要先从 LocalStorage 中加载数据，再用这个数据去服务器取数据。我们改一下（或者新建一个）`wrapWithAjaxData` 高阶组件，修改其中的 `componentWillMount`：

```
...
    componentWillMount () {
      ajax.get('/data/' + this.props.data, (data) => {
        this.setState({ data })
      })
    }
...
```

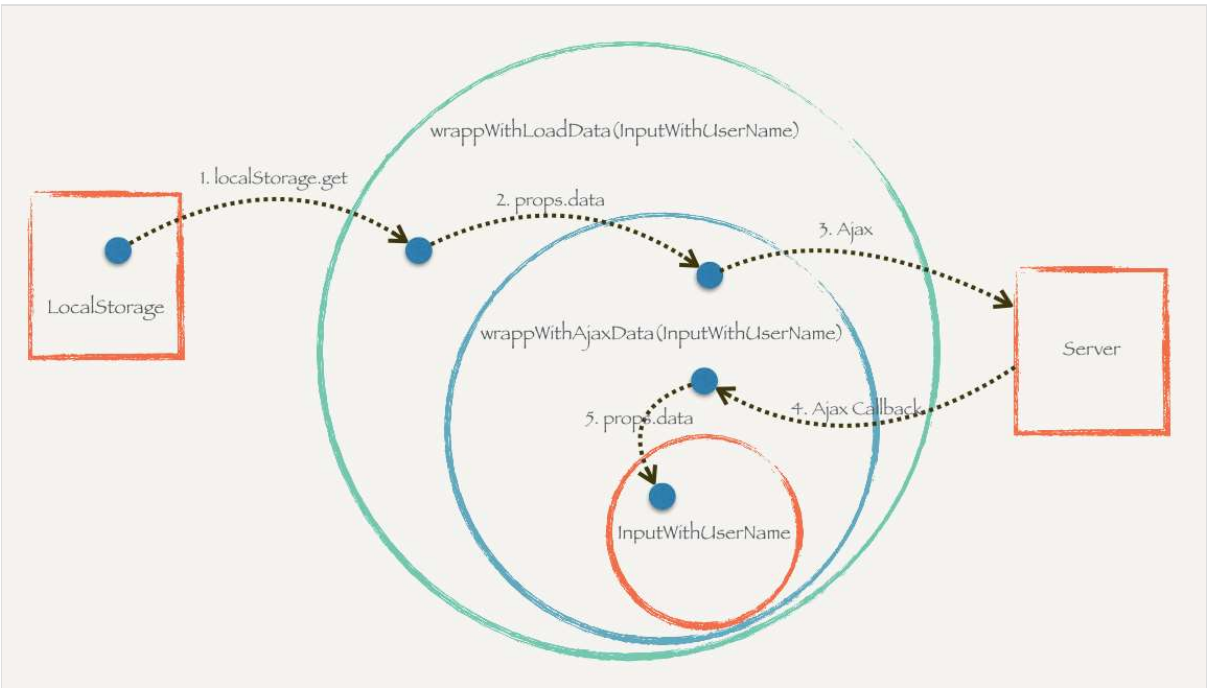
它会用传进来的 `props.data` 去服务器取数据。这时候修改 `InputWithUserName`：

```
import wrapWithLoadData from './wrapWithLoadData'
import wrapWithAjaxData from './wrapWithAjaxData'

class InputWithUserName extends Component {
  render () {
    return <input value={this.props.data} />
  }
}

InputWithUserName = wrapWithAjaxData(InputWithUserName)
InputWithUserName = wrapWithLoadData(InputWithUserName, 'username')
export default InputWithUserName
```

大家可以看到，我们给 `InputWithUserName` 应用了两种高阶组件：先用 `wrapWithAjaxData` 包裹 `InputWithUserName`，再用 `wrapWithLoadData` 包含上次包裹的结果。它们的关系就如下图的三个圆圈：



实际上最终得到的组件会先去 LocalStorage 取数据，然后通过 `props.data` 传给下一层组件，下一层用这个 `props.data` 通过 Ajax 去服务端取数据，然后再通过 `props.data` 把数据传给下一层，也就是 `InputWithUserName`。大家可以体会一下下图尖头代表的组件之间的数据流向：



用高阶组件改造评论功能（选读）

大家对这种在挂载阶段从 LocalStorage 加载数据的模式都很熟悉，在上一阶段的实战中，`CommentInput` 和 `CommentApp` 都用了这种方式加载、保存数据。实际上我们可以构建一个高阶组件把它们的相同的逻辑抽离出来，构建一个高阶组件 `wrapWithLoadData`：

```
export default (WrappedComponent, name) => {
  class LocalStorageActions extends Component {
    constructor () {
      super()
      this.state = { data: null }
    }

    componentWillMount () {
      let data = localStorage.getItem(name)
      try {
        // 尝试把它解析成 JSON 对象
        this.setState({ data: JSON.parse(data) })
      } catch (e) {
        // 如果出错了就当普通字符串读取
        this.setState({ data })
      }
    }

    saveData (data) {
      try {
        // 尝试把它解析成 JSON 字符串
        localStorage.setItem(name, JSON.stringify(data))
      } catch (e) {
        // 如果出错了就当普通字符串保存
        localStorage.setItem(name, `${data}`)
      }
    }

    render () {
      return (
        <WrappedComponent
          data={this.state.data}
          saveData={this.saveData.bind(this)}
          // 这里的意思是把其他的参数原封不动地传递给被包装的组件
          {...this.props} />
      )
    }
  }
  return LocalStorageActions
}
```

`CommentApp` 可以这样使用：

```
class CommentApp extends Component {
  static propTypes = {
    data: PropTypes.any,
    saveData: PropTypes.func.isRequired
  }

  constructor (props) {
    super(props)
    this.state = { comments: props.data }
```

```
  }

  handleSubmitComment (comment) {
    if (!comment) return
    if (!comment.username) return alert('请输入用户名')
    if (!comment.content) return alert('请输入评论内容')
    const comments = this.state.comments
    comments.push(comment)
    this.setState({ comments })
    this.props.saveData(comments)
  }

  handleDeleteComment (index) {
    const comments = this.state.comments
    comments.splice(index, 1)
    this.setState({ comments })
    this.props.saveData(comments)
  }

  render() {
    return (
      <div className='wrapper'>
        <CommentInput onSubmit={this.handleSubmitComment.bind(this)} />
        <CommentList
          comments={this.state.comments}
          onDeleteComment={this.handleDeleteComment.bind(this)} />
      </div>
    )
  }
}

CommentApp = wrapWithLoadData(CommentApp, 'comments')
export default CommentApp
```

同样地可以在 `CommentInput` 中使用 `wrapWithLoadData`，这里就不贴代码了。有兴趣的同学可以查看高阶组件重构的 CommentApp 版本。

总结

高阶组件就是一个函数，传给它一个组件，它返回一个新的组件。新的组件使用传入的组件作为子组件。

高阶组件的作用是用于代码复用，可以把组件之间可复用的代码、逻辑抽离到高阶组件当中。**新的组件和传入的组件通过 `props` 传递信息**。

高阶组件有助于提高我们代码的灵活性，逻辑的复用性。灵活和熟练地掌握高阶组件的用法需要经验的积累还有长时间的思考和练习，如果你觉得本章节的内容无法完全消化和掌握也没有关系，可以先简单了解高阶组件的定义、形式和作用即可。

课后练习

*React.js 加载、刷新数据 - 高阶组件

> 因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 React.js 小书的论坛 发帖，我会回答大家的疑问。

下一节：React.js 的 context

上一节：实战分析：评论功能（六）

4 条评论，4 人参与。

4

我有话说...

使用社交帐号登录 发布前先点击左边的按钮登录

最新评论

M • 6月21日 18:17
一直看不懂ant design 里的Modal 的写法是怎样的
顶 • 回复 • 分享»

围脖熊 • 5月3日 10:12
获益匪浅
顶 • 回复 • 分享»

XIII • 5月3日 09:45
为什么评论时页面会报错呢？
顶 • 回复 • 分享»

时间被海绵吃了 • 4月17日 19:21
赞，现在明白了 ant design 里面的 form 就用到了高阶组件
顶 • 回复 • 分享»

友言？