

React.js 小书

[<-- 返回首页](#)

PropTypes 和组件参数验证

- 作者: [胡子大哈](#)
- 原文链接: <http://huziketang.com/books/react/lesson24>
- 转载请注明出处, 保留原文链接和作者信息。

我们来到了到了一个非常尴尬的章节, 很多初学的朋友可能对这一章的知识点不屑一顾, 觉得用不用对程序功能也没什么影响。但其实这一章节的知识在你构建多人协作、大型的应用程序的时候也是非常重要的, 不可忽视。

都说 JavaScript 是一门灵活的语言 — 这就是像是说“你是个好人”一样, 凡事都有背后没有说出来的话。JavaScript 的灵活性体现在弱类型、高阶函数等语言特性上。而语言的弱类型一般来说确实让我们写代码很爽, 但是也很容易出 bug。

变量没有固定类型可以随意赋值, 在我们构建大型应用程序的时候并不是什么好的事情。你写下了 `let a = {}`, 如果这是个共享的状态并且在某个地方把 `a = 3`, 那么 `a.xxx` 就会让程序崩溃了。而这种非常隐晦但是低级的错误在强类型的语言例如 C/C++、Java 中是不可能发生的, 这些代码连编译都不可能通过, 也别妄图运行。

大型应用程序的构建其实更适合用强类型的语言来构建, 它有更多的规则, 可以帮助我们在编写代码阶段、编译阶段规避掉很多问题, 让我们的应用程序更加的安全。JavaScript 早就脱离了玩具语言的领域并且投入到大型的应用程序的生产活动中, 因为它的弱类型, 常常意味着不是很安全。所以近年来出现了类似 TypeScript 和 Flow 等技术, 来弥补 JavaScript 这方面的缺陷。

React.js 的组件其实是为了构建大型应用程序而生。但是因为 JavaScript 这样的特性, 你在编写了一个组件以后, 根本不知道别人会怎么使用你的组件, 往里传什么乱七八糟的参数, 例如评论组件:

```
class Comment extends Component {
  const { comment } = this.props
  render () {
    return (
      <div className='comment'>
        <div className='comment-user'>
          <span>{comment.username} </span>:
        </div>
        <p>{comment.content}</p>
      </div>
    )
  }
}
```

但是别人往里面传一个数字你拿他一点办法都没有：

```
<Comment comment={1} />
```

JavaScript 在这种情况下是不会报任何错误的，但是页面就是显示不正常，然后我们踏上了漫漫 debug 的路程。这里的例子还是过于简单，容易 debug，但是对于比较复杂的成因和情况是比较难处理的。

于是 React.js 就提供了一种机制，让你可以给组件的配置参数加上类型验证，就用上述的评论组件例子，你可以配置 `Comment` 只能接受对象类型的 `comment` 参数，你传个数字进来组件就强制报错。我们这里先安装一个 React 提供的第三方库 `prop-types`：

```
npm install --save prop-types
```

它可以帮助我们验证 `props` 的参数类型，例如：

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'

class Comment extends Component {
  static propTypes = {
    comment: PropTypes.object
  }

  render () {
    const { comment } = this.props
    return (
      <div className='comment'>
        <div className='comment-user'>
          <span>{comment.username} </span>:
        </div>
        <p>{comment.content}</p>
      </div>
    )
  }
}
```

注意我们在文件头部引入了 `PropTypes`，并且给 `Comment` 组件类添加了类属性 `propTypes`，里面的内容的意思就是你传入的 `comment` 类型必须为 `object`（对象）。

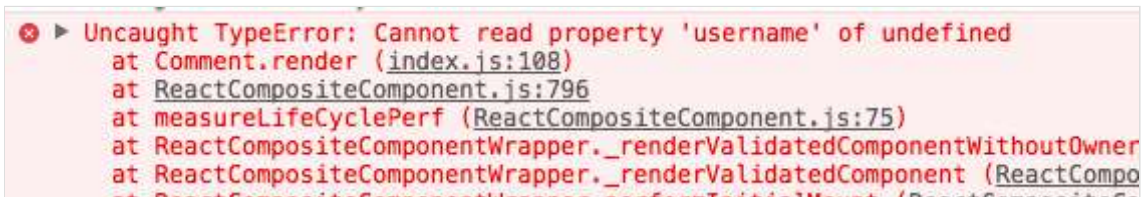
这时候如果再往里面传入数字，浏览器就会报错：

```
Warning: Failed prop type: Invalid prop `comment` of type `number` supplied to `Comment`, expected `object`.
in Comment (at index.js:376)
```

出错信息明确告诉我们：你给 `Comment` 组件传了一个数字类型的 `comment`，而它应该是 `object`。你就清晰知道问题出在哪里了。

虽然 `propTypes` 帮我们指定了参数类型，但是并没有说这个参数一定要传入，事实上，这些参数默认都是可选的。可选参数我们可以通过配置 `defaultProps`，让它在传入的时候有默认值。但是我们这里并没有配置 `defaultProps`，所以如果直接用

`<Comment />` 而不传入任何参数的话，`comment` 就会是 `undefined`，`comment.username` 会导致程序报错：



```
Uncaught TypeError: Cannot read property 'username' of undefined
    at Comment.render (index.js:108)
    at ReactCompositeComponent.js:796
    at measureLifeCyclePerf (ReactCompositeComponent.js:75)
    at ReactCompositeComponentWrapper._renderValidatedComponentWithoutOwner
    at ReactCompositeComponentWrapper._renderValidatedComponent (ReactCompo
    at ReactCompositeComponentWrapper.performInitialMount (ReactCompositeC
```

这个出错信息并不够友好。我们可以通过 `isRequired` 关键字来强制组件某个参数必须传入：

```
...
static propTypes = {
  comment: PropTypes.object.isRequired
}
...
```

那么会获得一个更加友好的出错信息，查错会更方便：



```
Warning: Failed prop type: The prop `comment` is marked as required in `Comment`, but its value is `undefined`.
    in Comment (at index.js:376)
```

React.js 提供的 `PropTypes` 提供了一系列的数据类型可以用来配置组件的参数：

```
PropTypes.array
PropTypes.bool
PropTypes.func
PropTypes.number
PropTypes.object
PropTypes.string
PropTypes.node
PropTypes.element
...
```

更多类型及其用法可以参看官方文档：[Typechecking With PropTypes - React](https://reactjs.org/docs/typechecking-with-proptypes.html)。

组件参数验证在构建大型的组件库的时候相当有用，可以帮助我们迅速定位这种类型错误，让我们组件开发更加规范。另外也起到了一个说明文档的作用，如果大家约定都写 `propTypes`，那你在别人写的组件的时候，只要看到组件的 `propTypes` 就清晰地知道这个组件到底能够接受什么参数，什么参数是可选的，什么参数是必选的。

总结

通过 `PropTypes` 给组件的参数做类型限制，可以在帮助我们迅速定位错误，这在构建大型应用程序的时候特别有用；另外，给组件加上 `propTypes`，也让组件的开发、使用更加规范清晰。

这里建议大家写组件的时候尽量都写 `propTypes`，有时候有点麻烦，但是是值得的。

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：实战分析：评论功能（四）

上一节：`dangerouslySetHTML` 和 `style` 属性

如果你觉得小书写得还不错，可以请胡子大哈喝杯茶 :)

赞赏

或者传播一下知识也是一个很好的选择