



用户登录和登出

第六章：基本数据读写操作

三条线路上的数据读写

第七章：聊天室

提交聊天信息的表单

提交聊天信息的表单

后面几个小节的课程咱们开始制作实时聊天功能了，本节先完成第一小步，创建一个聊天表单，能够发送信息，但信息不能在页面中显示出来，不过信息已经保存在 MongoDB 数据库之中了。

显示聊天表单

打开聊天室的组件文件 `Chat.jsx`，添加能够使用 Meteor 随动数据 `Meteor.user()` 的代码:

```
export default createContainer(() => {  
  Meteor.subscribe('userInfo');  
  return {  
    currentUser: Meteor.user(),  
  };  
}, Chat);
```

都是前面已经介绍的内容了，咱们再复习一下，`createContainer` 容器保证我们当前获得的用户数据是随动的，是响应式的。另外，获得的用户数据默认情况下只有 `username`、`emails` 和 `profile` 这三项内容，但我们需要更多的用户字段，就需要订阅已经发布的用户数据集 `userInfo`。第一次见到这些代码的时候，可能感觉形式怪怪的，也不好理解，但是用几次之后，我们就会习惯成自然，跟说“早上好”、“吃了吗”这些日常用语一样简单了，就是一些惯用法。

然后，就是把 Chat 组件的显示内容替换为：

```
<MessageForm currentUser={this.props.currentUser}/>
```

把当前用户的信息传递给 MessageForm 组件的 `currentUser` 属性。当然，在使用 MessageForm 组件之前把它需要导入到 `Chat.jsx` 文件中才能使用。

```
import MessageForm from './messages/MessageForm.jsx';
```

下面，我们就定义 MessageForm 这个组件，组件位于 `imports/ui/messages/MessageForm.jsx` 文件中。代码很简单，什么技巧也没有，纯粹是以前介绍的 Material-UI 组件的拼凑内联样式的使用。

查看更改: [显示聊天表单](#)

受控组件 TextField

这会儿到浏览器 /chat 页面就可以看到聊天表单的显示效果了。但是在输入的内容这块儿，我们使用一个 React 的知识点 **controlled components**，受控组件。给组件设置一个 `value` 属性，组件就成了受控组件，由 `value` 属性值控制组件状态。之前我们在定义 `NavBar` 组件的时候，使用的 `Material-UI` 的 `Tabs` 组件就是一个受控组件。现在，我们就给表单的 `TextField` 组件再添加一个神奇的 `value` 属性，比如属性值为 `hello`：

```
<TextField value='hello' />
```

这会儿到聊天室页面，会发现表单输入框中永远显示一个 `hello` 字符串，输入任何内容都不会被显示出来，字符串 `hello` 也不会被删除。那你可能会说，这不是吃饱了撑得吗？原来输入框是让自由输入的，现在不让自由输入了。`TextField` 组件受控之后，除了被虐之外，还有什么好处呢？其实，`TextField` 组件的 `value` 属性值可以用一个 `state` 变量来控制：

```
<TextField value={this.state.inputValue} />
```

这样就可以通过 `inputValue` 状态值来随时改变 `TextField` 组件输入框中显示的字符，这其实是非常符合 `React` 组件的基本思想的，代码也显得更加干净利落。

每次使用一个 `state` 变量，就给它设置一个初始值：

```
constructor(props) {  
  super(props);  
  this.state = {  
    inputValue: ''  
  };  
}
```

现在导致的结果是，到聊天室页面的输入框中，不管我们输入什么字符，不显示任何内容，因为此时 `inputValue` 状态值为空字符，进而 `TextField` 组件的 `value` 属性值也为空字符，所以输入框不显示任何内容。解决方法是给 `TextFiled` 组件再添加一个 `onChange` 事件，当输入框内容有变化的时候，触发 `handleChange` 事件处理器，

```
<TextField value={this.state.inputValue}
  onChange={this.handleChange.bind(this)} />
```

接下来，定义 `handleChange` 方法：

```
handleChange(e) {
  this.setState({inputValue: this.refs.message.getValue()});
}
```

这的代码就有点儿搞笑了，通过 `this.refs.message.getValue()` 获得用户输入的文字，赋值给 `state` 变量 `inputValue`，然后在页面输入框中显示 `inputValue` 的状态值。这其实有点儿脱裤子放屁多费一道手续呀，不过没关系，代码虽然麻烦了一些，但整个思维流程更符合 `React` 规范，后面带来的方便会更多。再到页面中测试一下，现在无论我们在输入框中输入什么，都能显示出来了。

查看更改：[受控组件 TextField](#)

创建 messages collection

接下来，主角就要登场了，当点击**发送**按钮，提交聊天表单的时候，会触发表单的 `onSubmit` 的事件处理器 `handleSubmit`，那 `handleSubmit` 方法中到底要执行哪些操作呢？

```
handleSubmit(e) {
  e.preventDefault();
  const message = this.refs.message.getValue();
  const currentUser = this.props.currentUser;
  const username = currentUser.username;
  const avatar_url = currentUser.avatar_url;

  Meteor.call('insert/message', username, avatar_url, message, (error)
    if (error) {
      console.log(error);
      return;
    }
    this.setState({inputValue: ''});
  }
```

```
});  
}
```

首先拿到我们在输入框中输入的内容，然后获取当前用户的用户名和头像链接，最后把这三个数据作为参数传递给 `insert/message` 这个方法。前面我们介绍过了，`Meteor.call` 接口可以在客户端呼叫要在服务器端执行的某个方法，把聊天信息保存到 MongoDB 数据库。`Meteor.call` 接口的最后一个参数是一个回调函数，若有错误，在浏览器 console 中打印错误并返回；若没有错误，则把 `inputValue` 状态值设置为空字符，也就是清空表单的内容，这也体现出了受控组件的优点。按照我们前面的思路，`insert/message` 这个方法要定义在 `imports/api/` 目录下的一个文件中，所以新建了一个 `messages.js` 文件，添加以下代码：

```
import { Mongo } from 'meteor/mongo';  
import { Meteor } from 'meteor/meteor';  
  
export const Messages = new Mongo.Collection('messages');  
  
Meteor.methods({  
  'insert/message': function (username, avatar_url, message) {  
    let message = {  
      owner: username,  
      avatar_url: avatar_url,  
      content: message,  
      createdAt: new Date()  
    };  
  
    Messages.insert(message);  
  }  
});
```

定义了 `'insert/message'` 这个方法，和前面我们添加用户信息的做法非常类似，但不同的是 `users` 这个 collection 是比较特殊的，因为 `users` 相关的操作在整个 Meteor 中是一个独特的东西，默认自动创建 collection。现在，我们需要自己手动创建存储聊天信息的 `messages` collection，所以添加了下面这行代码：

```
Messages = new Mongo.Collection('messages');
```

详细文档，请参考 Meteor [collections](#) 章节的内容。

接下来构建一个 `info` 对象，然后调用 Meteor 的 MongoDB 数据接口

`Messages.insert` 插入这条数据记录。总体而言，代码都很好理解。`Meteor.call` 所呼叫的方法应该在客户端和服务端同时加载，参考文档查看 Meteor [定义和呼叫 Methods](#) 部分的内容。

因为上述代码需要在客户端和服务端两端都运行，因此要在文件 `MessageForm.jsx` 导入 `messages.js` 文件：

```
import { Messages } from '../api/messages.js';
```

在服务器端文件 `server/main.js` 中导入 `messages.js` 文件：

```
import 'imports/api/messages.js';
```

这样，`Messages` 对象既可以在客户端访问，也可以在服务器端访问了。注：官方的例子就是这样写的。

这也有一个前提，就是 Meteor 这个框架同时在客户端浏览器和服务端上都有 MongoDB，所以才能在客户端去执行这样的 MongoDB 的插入数据的操作，那这样相同的一段数据，在服务器端存一次，同时在客户端去存一次，这样有什么好处吗？除了麻烦之外呢，可以参考这篇文章[延迟补偿](#)，有了[延迟补偿](#)呢，可以让我们的界面看上去实时性更好，同时也就是用户体验更好。具体怎么实现的，我就不展开讲了，大家可以参考[延迟补偿](#)文档。

查看更改：[用 Meteor Methods 机制保存聊天信息](#)

到浏览器页面中发送信息试试，输入 `say you later`，点击[发送按钮](#)。然后在终端命令行中，执行 `meteor mongo` 打开 MongoDB 的 shell 命令行：

```
show collections
```

可以看到多了一个 `messages collection`，这个集合中到底有什么内容呢？查看一下：

```
db.messages.find()
```

会发现 happypeter 这个人的头像链接和发送的 say you later 信息都存储在 messages collection 中了。本节课程也就这些了。

删除 insecure 包

最后要解决一个安全隐患问题。默认情况下，每个新创建的 Meteor 项目都安装了一个 insecure 包。这个包提供的功能可以让我们在客户端直接修改数据库，这样做实在是不安全，所以要删除 insecure 包，在命令行终端中执行命令：

```
meteor remove insecure
```

insecure 包去掉之后，客户端就不能直接修改数据库了，安全隐患消除了。

查看更改：[删除 insecure 包](#)

欢迎添加 Peter 的微信：happypeter1983

冀ICP备15007992号-3

