

代码开发的功能，现在用 Meteor 只需10行代码就能搞定”。本节将实现的用户登录功能真是可以体现 Meteor 开发的高效。

安装 accounts-password 包

Meteor 用自己的 Atmosphere 包来专门管理用户账号，这个包名为 **accounts-password**，它包含了一套完备的基于密码的用户认证系统，除了支持基本的通过用户名和密码来注册的功能之外，它还支持基于邮箱的注册流程并且包括邮箱地址验证功能，也提供邮箱找回密码功能。它使用 bcrypt 算法保存密码,避免密码泄露问题。之前的课程中安装的全是 npm 包，现在就学习一个新命令 `meteor add` 来安装 **accounts-password** 包，如下所示：

```
meteor add accounts-password
```

Atmosphere 包安装成功之后，在 `.meteor/packages` 文件中会记录一条所安装包的包名信息，同时也在 `.meteor/versions` 文件中记录一些所安装包的所有依赖包的信息。另外，你也可以通过直接修改 `.meteor/packages` 文件，来安装或删除 Atmosphere 包。

查看更改：[安装 accounts-password 包](#)

创建注册表单

Meteor 的 **accounts-password** 包已经保证了用户注册功能底层逻辑的实现，但是我们还需要一个用户注册表单，修改 `SignUp.jsx` 文件，添加注册表单的代码。

查看更改：[创建注册表单](#)

这样，一个简单而炫酷的注册表单就实现了。说简单是因为它只包含两个输入框：用户名和密码，还有一个注册按钮。说炫酷是因为点击输入框时出现的动态效果，多亏了 Material-UI 的 **TextField** 组件。

另外，这个注册表单是响应式的，所以用到了 Radium 来确保内联样式中的媒体查询语句正常工作。

真正实现用户注册功能

目前的注册表单只是一个静态的表单，不能提交数据。下面我们将实现表单的交互功能，继续修改 `SignUp.jsx` 文件，添加一些代码进来：

查看更改：实现用户注册功能

代码中，我们用到了 `onSubmit` 表单事件，当提交表单的时候会触发 `handleSubmit` 方法，从而获取用户输入的数据，创建一个新的用户账号。其中调用事件的 `preventDefault()` 方法是为了防止浏览器默认的提交表单的行为，要不然会刷新页面。

如何获取用户输入的用户名和密码信息呢？给每一个 `TextField` 组件添加一个 `ref` 属性，就能调用 `TextField` 组件定义的方法了。`TextField` 组件有一个 `getValue` 方法，可以获取 `input` 输入框中的内容，比如说用户名，

```
const userName = this.refs.userName.getValue();
```

得到了用户名和密码之后，调用由 `accounts-password` 提供的函数接口 `Accounts.createUser`，创建一个新的账号。

```
Accounts.createUser({username: userName, password: password}, (error)
  if (error) {
    console.log(error.reason);
    return;
  }
})
```

另外让代码更完善一些，可以给 `Accounts.createUser` 接口传递一个带有参数 `error` 的 `callback`，若创建失败则在浏览器 `console` 中打印错误信息并返回。到浏览器中测试一下，只输入用户名，然后点击注册按钮，页面没有反映，不过浏览器控制台中报错了：

```
Password may not be empty
```

若创建成功之后，则跳转到 `/account` 页面，渲染 `Account` 组件，又用到了 `context.router` 对象提供的 `push` 接口。因此，我们新建了一个

`imports/ui/Account.jsx` 文件，并到 `routes.jsx` 文件中增加了一个与 `Account` 组件相对应的路由，其路径为 `/account`。

这时候，在注册表单中填入用户名 `happypeter` 和密码（不为空），点击注册按钮，就会跳转到 `/account` 页面。那注册用户信息到底存储了吗？可以到浏览器 `console` 中看一下，输入：

```
Meteor.user()
```

回车，执行之后若返回类似下面这样的信息：

```
Object {_id: "dvDKEaFueP2A8DAji", username: "happypeter"}
```

说明用户信息已经存储到数据库中了。有意思的是，`Meteor` 的数据不仅仅存在服务器端（`MongoDB` 数据库），也存在客户端（`Minimongo` 数据库）。刚才在浏览器控制台中查看的信息来自客户端 `Minimongo` 数据库。那服务器端的数据保存好了吗？切换到命令行终端，保持应用运行的前提下，再新打开一个命令行终端窗口，进入到应用根目录下，执行命令：

```
meteor mongo
```

`meteor mongo` 会启动一个 `MongoDB shell` 程序，连接到 `127.0.0.1:3000/meteor` 数据库，在命令行中操作数据库，输入

```
show collections
```

列出 `meteor` 数据库中的所有集合（`collection`），其中有一个 `users` 集合，就是存储的用户信息。然后再输入：

```
db.users.find()
```

返回 `users` 集合中所有的用户数据，刚才注册的 `happypeter` 用户的信息也会出现在列表中。想了解更多基本的 MongoDB 数据库操作，可以参考文档 [MongoDB 快速入门](#)。

到此，我们的用户注册功能就实现了。

体现用户登录状态

接下来，我们想把导航栏的状态改变一下，当用户注册之后，导航栏上会有一个指向 `/account` 页面的 `Account` 标签。不过，我们不想再添加一个新的标签，而是想用 `Account` 标签替换 `Sign Up` 标签。如何实现呢？

以前写 Rails 的时候是这样，我相信在其他的语言当中也是非常类似的，就是我们会创建一个非常特殊的变量，名字还都是大同小异的，一般都叫做 `currentUser`。这个变量的特点就是如果有用户登录，`currentUser` 变量值不为空；没有用户登录，`currentUser` 的值为空或 `undefined`。因此，可以通过 `currentUser` 变量值决定显示 `Account` 标签还是 `Sign Up` 标签，简单用代码表示为：

```
<Tab label={ currentUser ? 'Account' : 'Sign Up' }  
      value={ currentUser ? '/account' : '/signup' } />
```

那 `currentUser` 变量值如何获得呢？Meteor 的默认行为是这样，当用户注册之后就自动处于登录状态了。那怎样知道用户处于登录状态呢？就使用一个新的 Meteor 接口 `Meteor.userId()`，当用户登录之后，这个接口可以返回当前登录用户的 `id`，我们可以到浏览器 `console` 中测试一下：

```
Meteor.userId()
```

返回值是一串类似下面这样的字符串，

```
"ru858EJbCnZzk8YAt"
```

若用户退出登录，这个接口返回值为 `null`。现在我们还没有添加用户退出登录功能，不过为了演示方便，我们可以在浏览器 `console` 中运行：

```
Meteor.logout()
```

用户就会退出登录了。然后再运行：

```
Meteor.userId()
```

可以看到返回值是 `null`。因此，`Meteor.userId()` 可以作为 `currentUser` 变量值的提供者，代码表示如下：

```
let currentUser = Meteor.userId();
```

这样通过 `currentUser` 变量值，`account` 标签和 `sign up` 标签就能够互相替换了。

另外还有一个问题，用户注册成功之后，`account` 标签不处于活跃状态，红色下划线不显示。修改 `componentWillReceiveProps` 生命周期方法，如下所示：

```
componentWillReceiveProps(nextProps) {
  setTimeout(() => {
    this.setState({
      tabIndex: this.getSelectedIndex()
    });
  }, 0)
}
```

延时设置 `tabIndex` 的状态值，让 `NavBar` 组件再渲染一次，这样就能使得 `account` 标签处于活跃状态，显示红色下划线。

FIXME: 不过，这种思路有点儿恶心。也可以用麻烦些的方法，定义两个导航栏组件，一个导航栏的标签是 `home`、`sign up` 和 `log in`，称为 `NavBar` 组件；另一个导航栏标签是 `home`、`account` 和 `chat`，称为 `LoginNavBar` 组件。然后在 `App.jsx` 文件中，通过 `currentUser` 判断到底挂载 `NavBar` 组件还是 `LoginNavBar` 组件。

```
let currentUser = Meteor.userId();
currentUser ? <LoginNavBar /> : <NavBar />
```

查看更改：体现用户登录状态

这样，用户注册功能的整个流程就算合理了。

欢迎添加 Peter 的微信：happypeter1983

冀ICP备15007992号-3

