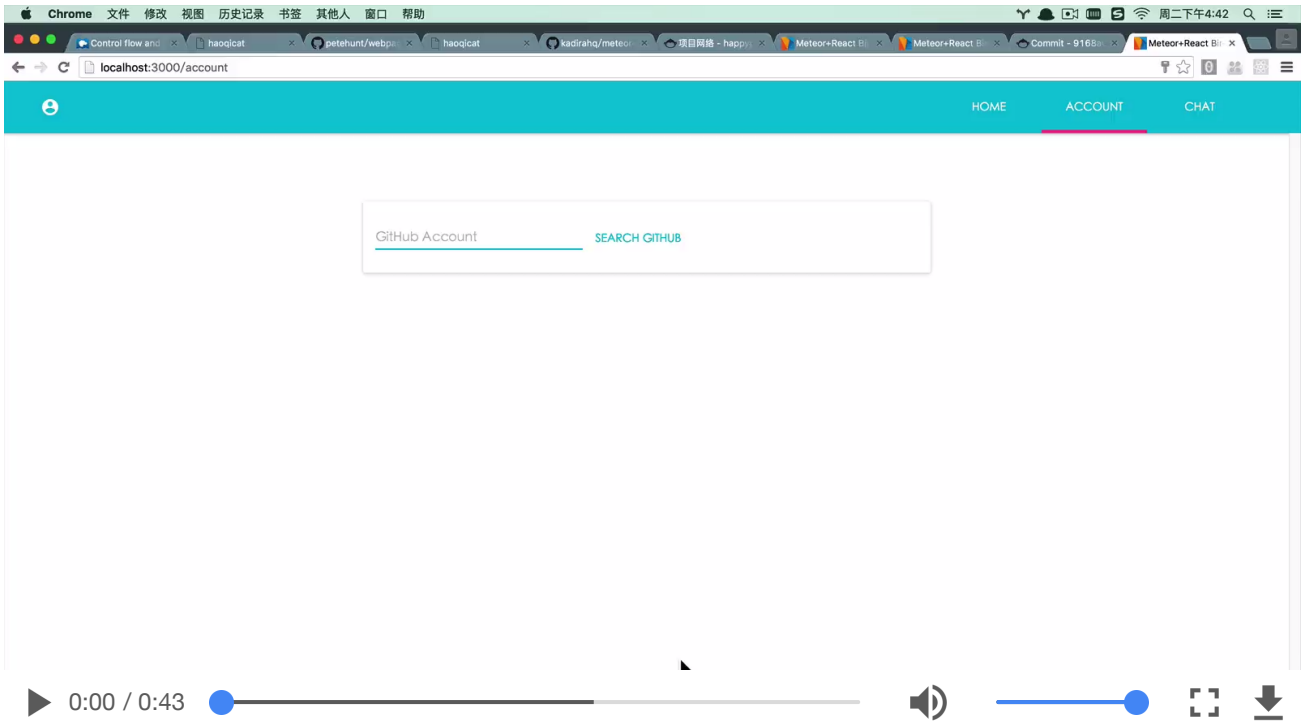

修改导航栏的样式

内联样式中使用媒体查询

响应式导航栏

样式重构，全局样式

第五章：用户认证系统

用户注册功能

三条线路上的数据读写

前面几节课程之中，咱们把注册登录的功能都做了，就是在 `/signup` 页面的注册表单中，输入用户名 `happypeter` 和密码，点击注册按钮，可以注册成功，登录进来。本节要实现的功能是让账户的信息更丰富一些，不是自己手动填写，而是从 GitHub 上获取数据。因此会在 `/account` 页面添加一个表单，用来输入用户在 GitHub 上的账号，提

交表单之后，从 GitHub 获取用户数据，并把获取的数据保存到本地数据库。要实现这些功能，要涉及到 Meteor 知识点还真是不少，但不用担心，我们分三个小步骤，一步一步的来讲解，第一步就是从我本地的客户端向 GitHub 的 API 服务器发 AJAX 请求，并从那边获取到数据；第二步把从 GitHub 获取的数据保存到我服务器上的 MongoDB 数据库之中；第三步就是在我的服务器和客户端之间搭建起数据的实时订阅通道，其实呢这主要是通过在服务器端进行数据的发布，在客户端进行数据的订阅来达成的。

第一步向 GitHub 发送 AJAX 请求

在向 GitHub 发请求之前，我们先修改一下 `Account.jsx` 文件，添加一个获取 GitHub 账号的静态表单进来：

查看更改：添加获取 GitHub 账号的表单

接下来向 GitHub 发请求。我们要解决的问题是向 GitHub 的 API 发 AJAX 请求，拿到我们的自己的头像。从 Meteor 这边发 AJAX 请求可以用 http 这个包，安装一下：

```
meteor add http
```

这个包装好了。接下来就是要实现获取 GitHub 账号的静态表单的交互功能。当提交表单之后，会触发表单的 `handleSubmit` 事件处理器，我们先实现这个方法。

```
handleSubmit(e) {
  e.preventDefault();
  const username = this.refs.username.getValue();
  const url = `https://api.github.com/users/${username}`;

  HTTP.call('get', url, (error, res) => {
    if(error) {
      console.log(error);
    } else {
      this.setState({user: JSON.parse(res.content)})
    }
  });
}
```

其中，代码 `e.preventDefault();` 是为了在表单提交的时候，不至于刷新页面。接下来一句，是从输入框中获得输入的 GitHub 用户名。然后，就是用获得的用户名来完成 GitHub 请求用户信息的 API 接口。最后调用 http 包提供的接口 HTTP.call 向 GitHub 发 AJAX 请求了。若有错误，则在浏览器 console 中打印存错误。若没有错误，则通过 JSON.parse 接口把返回数据中 `content` 字段对应的字符串转化成 JSON 数据，然后赋值给 state 变量 `user`，一般呢，我们会给 state 变量设置一个初始值，初始化 `user` 变量如下：

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

查看更改：发送 AJAX 请求获取 GitHub 账号信息

显示 GitHub 账号信息

既然已经获取了 GitHub 账号信息，那就把这些数据在 `/account` 页面显示出来吧。

先创建一个用来显示 GitHub 账号信息的组件 UserInfo，显示用户在 GitHub 上的头像、粉丝数、关注的人数和公开的仓库数，组件存放位置是 `imports/ui/user/UserInfo.jsx` 文件。组件很简单，没有新知识点。

在使用 UserInfo 组件之前，我们先安装了一个 node 包 lodash，安装命令：

```
npm install --save lodash
```

因为我们要使用 lodash 提供的 isEmpty 方法判断 state 变量 `user` 的值是否为空，不为空的时候，才挂载 UserInfo 组件和 RaisedButton 组件。

顺便说一下，目前 Meteor 自带了 JavaScript 库 Underscore，不过将来 Meteor 会放弃使用这个包，文档参考这里，因此本节用 Lodash 库。

现在到浏览器中操作一下，在输入框中输入我的 GitHub 账号 `happypeter`，回车或点击旁边的按钮，我的 GitHub 账号信息就会在输入框下方显示出来，还有一个粉色的保存按钮。

查看更改：显示 GitHub 账号信息

至此，第一步工作就做完了，信息拿到并显示出来了，下一步我们做保存，也就是第二步把数据保存到服务器的 MongoDB 数据库中。

第二步把 GitHub 账号信息存储到数据库

不管数据的保存步骤如何，它肯定是在**保存**按钮的 `handleClick` 事件处理器中实现的，所以我们就来定义这个 `handleClick` 方法。

```
handleClick(e) {
  e.preventDefault();
  Meteor.call('update/user', this.state.user, (error) => {
    if(error) {
      console.log(error);
      return;
    }
    this.context.router.push('/chat');
  });
}
```

这里用到了 `Meteor.call` 这个接口，这个知识点就有意思了，因为当前我们是在客户端的一个组件之内，Meteor 的一个规范是凡是放到 `client` 文件夹下的代码（或通过 `import` 导入的代码）都是在客户端浏览器中执行的，但现在我们要执行的操作是要在服务器端保存数据，所以在客户端呼叫一个在服务器端定义的方法，正是 `Meteor.call` 的用武之地，相应的参考文档查看这里。

上述代码中给 `Meteor.call` 接口传入的第一参数 `update/user` 就是在服务器端要执行的方法的名字；`this.state.user` 是向服务器端发送的数据；第三个参数是一个回调函数可以处理错误信息，若没有错误，则跳转到 `/chat` 页面。

下面，我们就集中精力创建这个要在服务器端执行的方法。参考 Meteor 提供的应用文件结构范例，像这种操作数据库的代码要放到 `import/api` 目录中，新建一个叫做

users.js 的文件，添加代码：

```
Meteor.methods({
  'update/user': function(user) {
    let info = {
      avatar_url: user.avatar_url,
      followers: user.followers,
      following: user.following,
      public_repos: user.public_repos
    };

    Meteor.users.update(this.userId, {$set: info});
  }
});
```

通过 Meteor.methods 接口定义 update/user 方法，就像定义一个函数一样。把从 Meteor.call 传过来的 GitHub 的信息，保留 avatar_url 、 followers 、 following 和 public_repos 四个字段的内容并定义成一个 info 对象，然后调用 Meteor 操作 MongoDB 数据库的接口 Meteor.users.update 修改 MongoDB 数据库中 users 集合（collection）里面的数据。如果 users 集合里有多个数据，可以通过 this.userId 当前用户的 id 号，定位需要更新的是哪个记录，当前用户，然后把所有的信息存储进去。

定义的 update/user 方法需要在客户端和服务器端都加载。若在客户端加载，就在 Account.jsx 文件中导入 users.js 文件，添加一行代码：

```
import '../api/users.js';
```

在服务器端加载，同样在 server/main.js 中导入 users.js 文件，添加一行代码：

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

查看更改：把 GitHub 账号信息存储到数据库

代码都写好之后，我们测试一下，到 `/account` 页面，在输入框中输入 `happypeter`，获取信息，点击保存按钮，页面会切换到 `/chat` 聊天室页面，但是服务器上到底数据有没有保存好呢，还是到命令行中运行：

```
meteor mongo
```

启动 MongoDB 的 shell 命令行，看看 `users` 集合之中都有哪些内容？输入命令：

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

会看到 `happypeter` 这个用户的头像以及各项信息都保存好了。这里分享一点儿调试的时候可能需要的技巧，如果 `users` 集合中有很多项数据，如何把它们一次性都删除呢？可以用下面的命令：

```
db.users.remove({})
```

至此，第二步保存数据到服务器，我们也做完了。

第三步发布和订阅用户信息

接下来，完成最后一步，打通服务器和客户端之间的实时订阅通道，主要通过 publish 和 subscribe 实现。那针对的我们这的实际问题就是，虽然数据已经保存了，但是返回到 `/account` 页面，获取的 GitHub 信息又没有了。在传统的一些架构之下，我们可以从客户端向服务器端发送请求，服务器端再返回给我们相应的数据就可以显示了。但 Meteor 是全新的 web 框架的架构，服务器和客户端数据是通过实时订阅来实现的。刚才我们提到了，这个实时订阅通道通过服务器端发布数据，客户端来订阅相应的字段来完成的，但是针对 Meteor.users 有点儿特殊，看一下它的文档，说是在默认情况下，当前用户数据记录中的 `username`、`emails` 和 `profile` 三个字段是默认发布给客户端的，客户端可以看到这三个字段的内容。当然，如果你想发布当前用户数据记录中其它字段的内容，还得自己手动添加代码实现，一会儿将看到。但是，不管怎么说，`/account` 页面上显示的就是当前用户的一些信息，所以到 `Account.jsx` 文件之中，用之前提到的方法，获取 Meteor 动态数据 Meteor.user()，代码如下：

```
export default createContainer(() => {
  return {
    currentUser: Meteor.user(),
  };
}, Radium(Account));
```

然后，用户信息显示部分需要做些修改。就是 `GitHubInfo` 变量，也是一个小组件，显示内容发生变化了。若向 GitHub 发送的 AJAX 请求刚刚返回数据的这一瞬间， `GitHubInfo` 组件显示的是刚返回的 GitHub 数据和一个粉红色的**保存**按钮；若有用户登录进来且当前用户的数据记录中有 `avatar_url` 字段，这意味着当前用户已经把获取的 GitHub 信息保存到数据库中了，那在这种条件下， `GitHubInfo` 组件显示的是当前用户在数据库中保存的 GitHub 信息。代码简写如下：

```
let currentUser = this.props.currentUser;
if(!isEmpty(this.state.user)) {
  ...
} else if(currentUser && currentUser.avatar_url) {
  GitHubInfo = (
    <UserInfo userInfo={this.props.currentUser} />
  );
}
```

但是，到浏览器里看一下，却看不到用户的信息。这是因为 `avatar_url` 这个字段虽然保存到服务器端数据库中了，但客户端是得不到的，所以我们需要手动把 `avatar_url` 等其他四个字段都发布出来。修改 `users.js` 文件：

```
if(Meteor.isServer) {
  Meteor.publish('userInfo', function(){
    return Meteor.users.find(
      {_id: this.userId },
      {fields: {avatar_url: 1, followers: 1, following: 1, public_repos
    );
  });
}
```

Meteor 的数据发布操作只在服务器端运行，所以使用了 Meteor.isServer 接口检查代码运行环境，若返回值为 true，说明是在服务器端运行。

通过 Meteor.publish 接口把 `users` 集合中，当前用户数据记录中除了 `username` 等默认发布的字段之外，把其它的 `avatar_url`、`followers`、`following` 和 `public_repos` 四个我们需要的字段都发布出来。`userInfo` 是所发布的数据集的名字。

服务器端发布数据之后，相应的客户端订阅数据之后，服务器端才会把数据发送到客户端，所以我们还得在 `Account.jsx` 文件中的 `createContainer` 容器中添加一行代码：

```
Meteor.subscribe('userInfo');
```

通过 Meteor.subscribe 接口完成对刚才发布的 `userInfo` 数据集合的订阅。代码修改完成之后，再到浏览器 `/account` 页面看一下，当前用户保存的 GitHub 账号信息已经显示出来了。

查看更改：第三步发布和订阅用户信息

这样，我们的第三步工作也完成了。不过，到运行 `meteor` 命令的终端窗口看一下，会看到一些警告信息，说 “你的 autopublish 还开着呢”，开着会有什么后果呢？服务器端所有数据会自动的被发送到所有的客户端，这显然不适用于刚才 `users` 集合中的一些数据字段，但其它集合中的数据是适用的，这显然不是特别安全。一般的建议是在开发过程就把 autopublish 功能关掉，在终端命令行中运行：

```
meteor remove autopublish
```

实际上，自动发布功能是由 autopublish 包提供的，所以删除这个包就可以了。

查看更改：取消自动发布功能

至此，本节课程就结束了，主要对数据的三种流向做了一些操作和控制。