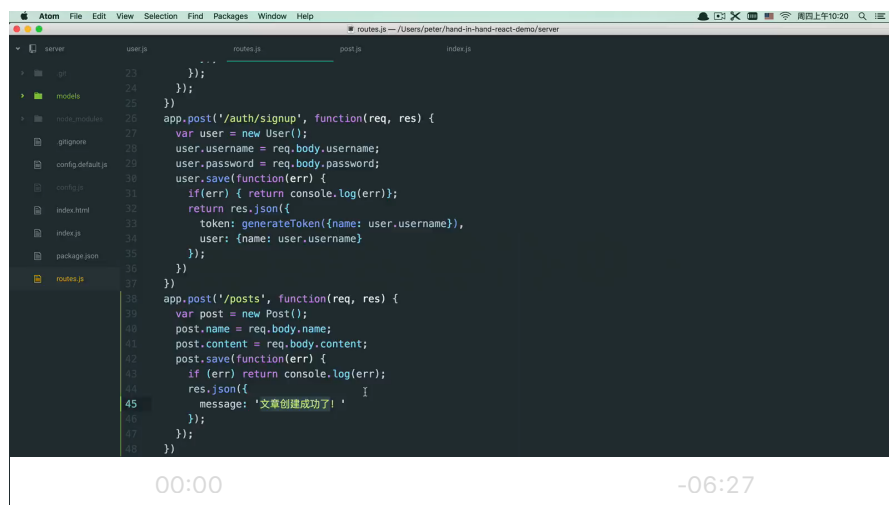


[首页](#) [目录](#)[退出](#)

```
23  });
24  });
25  })
26  app.post('/auth/signup', function(req, res) {
27    var user = new User();
28    user.username = req.body.username;
29    user.password = req.body.password;
30    user.save(function(err) {
31      if(err) { return console.log(err);
32      return res.json({
33        token: generateToken({name: user.username}),
34        user: {name: user.username}
35      });
36    });
37  })
38  app.post('/posts', function(req, res) {
39    var post = new Post();
40    post.name = req.body.name;
41    post.content = req.body.content;
42    post.save(function(err) {
43      if (err) return console.log(err);
44      res.json({
45        message: '文章创建成功了!'
46      });
47    });
48  })
49  })
```

第五章：Redux 大显神通

创建 Redux Store 传递给 React 组件

创建与用户认证相关的 Redux Reducer

提供 Redux Thunk 库

后端限定添加新文章接口的访问权限

验证请求

创建新文章接口是服务器端的受保护资源，不允许任意请求都能访问，只有提供认证码（JWT token）并且通过服务器端验证的请求才能访问这个接口。下面我们就编写一个验证请求的**中间件**，打开 `server/routes.js` 文件，添加代码：

```
var requireAuth = function(req, res, next) {
  var token = req.headers.authorization;

  if (token) {
    jwt.verify(token, secret, function(err, decoded) {
      if(err) {
```

```
    if(err.name === 'TokenExpiredError')
      return res.status(401).json({ error: '认证失败' })
    } else {
      return res.status(401).json({ error: '认证失败' })
    }
  } else {
    if(decoded.admin === true) {
      next();
    } else {
      res.status(401).json({ error: '认证失败' })
    }
  }
});
} else {
  return res.status(403).json({
    error: '请提供认证码!'
  });
}
}
```

客户端通过 HTTP 请求 Authorization 头标识把认证码发送给服务器端，服务器端通过代码

`req.headers.authorization` 获得认证码，然后判断认证码是否真的存在，若不存在，则响应 HTTP 403 状态码，返回错误信息；若存在，则通过

`jsonwebtoken` 模块提供的 `verify` 接口验证认证码是否有效，若认证码无效，则服务器端响应 HTTP 401 状态码，并返回错误信息；若认证码有效，则把解码后的认证码保存到 `decoded` 对象中，获取 `decoded` 中包含的用户信息，判断用户是否为 `admin`，若用户是管理员，则执行下一个中间件，若用户不是管理员，则返回错误信息。

认证码失效时间

当验证认证码是否有效的时候，会检验认证码的失效时间，若认证码过期，则报告错误

TokenExpiredError。在前面课程中讲解如何生成认证码的时候，给每个生成的认证码设置了一个过期间隔，如下：

```
var generateToken = function(user) {  
  return jwt.sign(user, secret, {  
    expiresIn: 3000  
  });  
}
```

`expiresIn` 选项指定了认证码自生成到失效的间隔时间（过期间隔），可用秒数表示，也就是说，本案例中生成的每个 JWT 认证码再过50分钟就失效了，当然为了课程演示方便，可以把这个过期间隔再设置的短些，比如说300秒。在实际应用中，则要根据业务场景的不同，合理的设置过期间隔。那么，以本案例中的认证码为例，解码之后的认证码类似这样：

```
{username: 'trump', admin: 'true', iat: 14791
```

其中，`iat` 指的是生成认证码的时间戳，`exp` 指的是认证码的失效时间戳，两者正好相差3000秒。

受保护的新建文章接口

然后，更改一下 `/posts` 路由接口，使用 `requireAuth` 中间件：

```
app.post('/posts', requireAuth, function(req,
```

执行完 `requireAuth` 中间件，请求验证通过之后，才执行 `function(req, res) {...}` 回调函数，这样就确保了 `/posts` 接口的安全性。

测试受保护的新建文章接口

首先，客户端以管理员的身份登录之后，从 `Session Storage` 中获取到管理员的认证码，然后通过 `curl` 命令测试：

```
curl -H "Content-Type: application/json" -H '
```

用真实有效的认证码代替 `xxx.xxx.xxx`，就能得到正确的返回结果了：

```
{"message": "文章创建成功了!"}
```

欢迎添加 Peter 的微信：happypeter1983

冀ICP备15007992号-3

