在线实验，请到PC端体验

# 一、实验介绍

## 1.1 实验内容

本课程结和 Koa 和 React 开发 Web 应用，学习 Koa 和 React 基础知识。课程中将会学习如何创建 Koa 应用，Koa 常用中间件，Koa 路由，以及 React 应用开发等知识点。

koa 是由 Express 原班人马打造的，致力于成为一个更小、更富有表现力、更健壮的 Web 框架。使用 koa 编写 web 应用，通过组合不同的 generator，可以免除重复繁琐的回调函数嵌套，并极大地提升错误处理的效率。koa 不在内核方法中绑定任何中间件，它仅仅提供了一个轻量优雅的函数库，使得编写 Web 应用变得得心应手。

React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己写一套，用来架设 Instagram 的网站。做出来以后，发现这套东西很好用，就在2013年5月开源了。由于 React 的设计思想极其独特，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来 Web 开发的主流工具。

## 1.2 知识点

- Koa 基础
- 如何创建 Koa 应用
- Koa 常用中间件
- Koa 路由
- React 基础

## 1.3 实验环境

- Node.js 6.x

## 1.4 适合人群

本课程难度一般，属于初级课程，适合具有 Node.js 基础的用户学习 Koa 结和 React 开发。

# 二、实验步骤

## 2.1 创建 Koa 应用

新建 `koa-react` 目录，进入 `koa-react` 目录输入如下命令初始化项目：

```
$ npm init
```

在 `koa-react` 目录中创建 `app` 文件夹，用于存放应用代码。

首先安装 Koa：

```
$ npm install koa --save
```

进入 `app` 目录，创建 `app.js` 文件，输入如下代码：

```
// 引入 koa
import koa from 'koa';

// 创建 koa 应用
const app = koa();

// app.use 方法用于向 middleware 数组添加 Generator 函数
// 中间件和 Express 中的中间件类似，但必须都是 Generator 函数
app.use(function *() {
    this.body = 'Hello Koa';
});

app.listen(8080);
```

Koa 应用非常简单，一个 Koa 应用就是一个对象，包含了一个 middleware 数组，这个数组由一组 Generator 函数组成。这些函数负责对 HTTP 请求进行各种加工，比如生成缓存、指定代理、请求重定向等等。

这里使用了 ES6 的语法，我们需要引入 Babel 来支持更多的 ES6 语法，在 koa-react 目录下，安装 Babel：

```
$ npm install babel-core babel-cli babel-preset-es2015 --save-dev
```

然后修改 package.json 文件，其中的 scripts 项改为如下所示：

```
  "scripts": {
    "test": "babel-node --debug --presets es2015 app/app.js"
  },
```

这样我们就可以测试应用了，此次实验重点在 Koa 和 React 所以不详细介绍 Babel（后面还会用到其他的工具，比如 Webpack 等），运行命令测试：

```
$ npm test
```

OK，打开浏览器，访问 http://127.0.0.1:8080 ，可以看到显示的 Hello Koa 字样。说明这个简单的Koa服务运行成功了。

## 2.2 Koa 常用中间件

下面添加一些常用的 Koa 中间件，在 koa-react 目录下安装中间件：

```
$ npm install koa-bodyparser --save
$ npm install koa-static --save
$ npm install koa-session --save
```

koa-bodyparser 用于解析 request body，示例代码：

```
var koa = require('koa');
var bodyParser = require('koa-bodyparser');

var app = koa();
app.use(bodyParser());

app.use(function *() {
    // 使用了 bodyParser
    // 解析后的 body 会存储在 this.request.body 中
    this.body = this.request.body;
});
```

koa-session 用于管理应用的 session，示例代码：

```
var session = require('koa-session');
var koa = require('koa');
var app = koa();

// session 安全 key
app.keys = ['some secret hurr'];
app.use(session(app));

app.use(function *(){
    // 忽略 favicon
    if (this.path === '/favicon.ico') return;

    var n = this.session.views || 0;
    this.session.views = ++n;
    this.body = n + ' views';
});

app.listen(3000);
console.log('listening on port 3000');
```

koa-static 用于创建静态文件服务。

在 app.js 中引入中间件，修改 app.js 代码，如下所示：

```
import koa from 'koa';
import bodyParser from 'koa-bodyparser';
import session from 'koa-session';
import staticServer from 'koa-static';

const app = koa();

// 配置静态文件地址
app.use(staticServer(__dirname + '/public'));
// session
app.keys = ['some secret hurr'];
app.use(session(app));
// body parser
app.use(bodyParser());

app.use(function *() {
    this.body = 'Hello Koa';
});

app.listen(8080);
```

前端的代码就存放在静态文件文件夹里面，在 app 目录下创建 public 文件夹。

下面我们来处理**HTML**文件，需要引入一个模板引擎，我们使用 **Nunjucks**，安装：

```
$ npm install nunjucks --save
```

我们需要为 **nunjucks** 写一个中间件，**Koa** 才能使用 **nunjucks**，在 app 目录下创建 middlewares/koa-nunjucks/index.js ，输入如下代码：

```
import nunjucks from 'nunjucks';

export default (path, opts) => {
    let env = nunjucks.configure(path, opts);

    return function *(next) {
        let self = this;

        this.render = (view, ctx) => {
            return new Promise((resolve, reject) => {
                nunjucks.render(view, ctx, (err, res) => {
                    if (err) {
                        return reject(err);
                    }

                    self.body = res;
                    resolve();
                });
            });
        };

        yield next;
    };
}
```

然后在 app.js 文件中，引入这个中间件：

```
import fs from 'fs';
import koa from 'koa';
import bodyParser from 'koa-bodyparser';
import session from 'koa-session';
import staticServer from 'koa-static';
import nunjucks from './middlewares/koa-nunjucks';

const app = koa();

// nunjucks 配置
app.use(nunjucks(__dirname + '/views', {
    noCache: process.env.NODE_ENV === 'production',
    watch: ! process.env.NODE_ENV === 'production'
}));

// 配置静态文件地址
app.use(staticServer(__dirname + '/public'));
// session
app.keys = ['some secret hurr'];
app.use(session(app));
// body parser
app.use(bodyParser());

app.listen(8080);
```

在 app 目录下创建 views 文件夹，模板文件就放在这个文件夹中了。

## 2.3 Koa 路由

因为后面会把应用的路由交给 React 管理，所以 Koa 的路由主要是服务端 API 的路由，使用 koa-router 管理 API 路由，添加路由中间件 koa-router：

```
$ npm install koa-router --save
```

在 app 目录下创建 api 文件夹和 routers 文件夹，在 routers 文件夹中创建 index.js ，并输入如下代码：

```
import koaRouter from 'koa-router';
import user from '../api/user';

export default (app) => {
    let router = koaRouter();

    app.use(router.routes());

    // 定义服务器端的 API
    router.get('/api/user', user);

    // 其他路由全部渲染到模板
    // 这样可以把这些路由交给 react 处理
    router.get('*', function *(next) {
        yield this.render('index.html', {});
        yield next;
    });
}
```

在 views 中新建 index.html 文件，输入如下代码：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>koa app</title>
    </head>
    <body>
        <div id="app-root"></div>
        <script src="/js/dest/app.js"></script>
    </body>
</html>
```

在 api 文件夹中创建 user.js 文件，并输入如下代码：

```
export default function *(next) {
    yield this.body = { user: 'shiyanlou' };
}
```

然后在 app.js 中引入定义的 **router**，需要改 app.js 代码如下所示：

```
import fs from 'fs';
import koa from 'koa';
import bodyParser from 'koa-bodyparser';
import session from 'koa-session';
import staticServer from 'koa-static';
import nunjucks from './middlewares/koa-nunjucks';
import routers from './routers';

const app = koa();

// nunjucks 配置
app.use(nunjucks(__dirname + '/views', {
    noCache: process.env.NODE_ENV === 'production',
    watch: ! process.env.NODE_ENV === 'production'
}));

// 配置静态文件地址
app.use(staticServer(__dirname + '/public'));
// session
app.keys = ['some secret hurr'];
app.use(session(app));
// body parser
app.use(bodyParser());

// 加载路由
routers(app);

app.listen(8080);
```

现在在 koa-react 目录下运行应用：

```
$ npm test
```

打开浏览器，访问 http://127.0.0.1:8080/api/user ，浏览器显示 { "user": "shiyanlou" } 字样，则表示路由创建成功了。下面用 react 来渲染前端模板。

## 2.4 React 应用

首先，在 koa-react 目录下安装需要的 npm 包：

```
$ npm install jquery --save
$ npm install react react-dom react-router --save
$ npm install babel-loader babel-preset-react --save-dev
$ npm install webpack gulp gulp-util --save-dev
$ npm install gulp -g
```

在 public/js 目录下新建 app.js ，输入如下代码：

```
import React from 'react';
import { render } from 'react-dom';
import { Router, Route, IndexRoute, browserHistory } from 'react-router';
import Home from './pages/home';
import User from './pages/user';

// 创建 App 根组件
class App extends React.Component {
    render() {
        return (
            <div className="container">
                  {this.props.children}
              </div>
        );
    }
}

// 创建 react 路由
const routes = (
    <Router history={browserHistory}>
         <Route path="/" component={App}>
             <IndexRoute component={Home} />
             <Route path="user" component={User}></Route>
         </Route>
     </Router>
);

// 渲染 react 应用
render(routes, document.getElementById('app-root'));
```

在 public/js 目录中创建 components 和 pages 文件夹。

components 文件夹用于存放组件，在 components 文件夹中创建 getUser.js 文件，并输入如下代码：

```
import $ from 'jquery';
import React from 'react';
import { Link } from 'react-router';

export default class GetUser extends React.Component {
    constructor(props) {
        super(props);
        this.state = { data: '' };
    }

    componentDidMount() {
        let self = this;

        // 使用 jQuery ajax 方法获取 API 数据
        this.r = $.get(this.props.source, (user) => {
            self.setState({
                data: user,
            });
        });
    }

    componentWillUnmount() {
        this.r.abort();
    }

    render() {
        return (<div>{this.state.data.user}</div>);
    }
}
```

在 js/pages 目录中新建 home.js 和 user.js 文件。

home.js 中代码如下所示：

```
import React from 'react';
import { Link } from 'react-router';

export default class Home extends React.Component {
    render() {
        return (<div>home page, <Link to={"/user"}>to user</Link></div>);
    }
}
```

user.js 中代码如下所示：

```
import React from 'react';
import GetUser from '../components/getUser';

export default class User extends React.Component {
    render() {
        return (<GetUser source="/api/user" />);
     }
}
```

然后我们来编写**webpack**的配置为，在 koa-react 目录下新建 webpack.config.js 文件，输入如下代码：

```
'use strict';

var path = require('path');
var webpack = require('webpack');

module.exports = {
    devtool: '#source-map',
    colors: true,
    progress: true,
    // 入口文件
    entry: {
        app: './app/public/js/app.js',
    },
    // 编译后输出文件
    output: {
        path: path.join(__dirname, 'app/public/js/dest'),
        filename: '[name].js',
    },
    module: {
        // 调用 babel-loader 编译 react 组件
        loaders: [{
            loader: 'babel-loader',
            include: [
                path.resolve(__dirname, 'app/public/js')
            ],
            test: /\.jsx?$/,
            query: {
                plugins: [],
                presets: ['es2015', 'react'],
            },
        }]
    },
    plugins: [
        // 压缩代码
        new webpack.optimize.UglifyJsPlugin()
    ],
};
```

在 koa-react 目录下新建**gulp**的配置文件 gulpfile.js ，输入如下代码：

```
'use strict';

var gulp         = require('gulp');
var util         = require('gulp-util');
var webpack      = require('webpack');
var webpackConfig = require('./webpack.config');

// run webpack
gulp.task('webpack', function() {
    webpack(webpackConfig, function(err, stats) {
        if (err) throw new util.PluginError('webpack', err);
        util.log('[webpack]', stats.toString({}));
    });
});

gulp.task('default', ['webpack']);
```

**OK**，现在编译**JavaScript**代码，在 koa-react 目录下运行**gulp**：

```
$ gulp
```

编译完成之后，运行项目：

```
$ npm test
```

然后打开浏览器，访问 http://127.0.0.1:8080 可以看到显示**“home page，to user”**，访问 http://127.0.0.1:8080/user 可以看到显示**“shiyanlou”**。

下面来添加一个**404**组件，在 js/pages 目录下新建 notFound.js 文件，并输入如下代码：

```
import React from 'react';
import { Link } from 'react-router';

export default class NotFound extends React.Component {
    render() {
        return (<div>404, page not found</div>);
    }
}
```

然后在 public/js/app.js 文件中引入这个组件：

```
import React from 'react';
import { render } from 'react-dom';
import { Router, Route, IndexRoute, browserHistory } from 'react-router';
import Home from './pages/home';
import User from './pages/user';
import NotFound from './pages/notFound';


class App extends React.Component {
    render() {
        return (
            <div className="container">
                  {this.props.children}
              </div>
        );
    }
}

const routes = (
    <Router history={browserHistory}>
         <Route path="/" component={App}>
             <IndexRoute component={Home} />
             <Route path="user" component={User}></Route>
             <Route path="*" component={NotFound}></Route>
         </Route>
     </Router>
);

render(routes, document.getElementById('app-root'));
```

现在重新编译JavaScript代码，然后运行项目，现在访问没有处理的路由地址，会看到404页面了。

Koa 和 React 的结合基本完成了，还可以用 Mongoose 连接数据库，使用 Redux 管理 React 组件状态。

## 三、总结

通过本课程学习了 Koa 结和 React 的开发 Web 应用的方式和简单流程。