目录

1. Express 框架简介及初体验

1.1 Express 框架是什么

Express 是一个基于 Node 平台的 web 应用开发框架，它提供了一系列的强大特性，帮助你创建各种 Web 应用。我们可以使用 npm install express 命令进行下载。

1.2 Express 框架特性

1. 提供了方便简洁的路由定义方式
2. 对获取 HTTP 请求参数进行了简化处理
3. 对模板引擎支持程度高，方便渲染动态 HTML 页面
4. 提供了中间件机制有效控制 HTTP 请求
5. 拥有大量第三方中间件对功能进行扩展

1.3 原生 Node.js 与 Express 框架对比之路由

```
    app.on('request', (req, res) => {
        // 获取客户端的请求路径
        let { pathname } = url.parse(req.url);
        // 对请求路径进行判断 不同的路径地址响应不同的内容
        if (pathname == '/' || pathname == 'index')
{
            res.end('欢迎来到首页');
        } else if (pathname == '/list') {
            res.end('欢迎来到列表页页');
        } else if (pathname == '/about') {
```

```
// 当客户端以 get 方式访问/时
app.get('/', (req, res) => {
    // 对客户端做出响应
    res.send('Hello Express');
});

// 当客户端以 post 方式访问/add 路由时
```

1.4 原生 Node.js 与 Express 框架对比之获取请求参数

```
app.on('request', (req, res) => {
   // 获取 GET 参数
   let {query} = url.parse(req.url, true);
   // 获取 POST 参数
   let postData = '';
   req.on('data', (chunk) => {
       postData += chunk;
   });
   req.on('end', () => {
       console.log(querystring.parse(postData)
   }));
});
```

```
app.get('/', (req, res) => {
    // 获取 GET 参数
    console.log(req.query);
});

app.post('/', (req, res) => {
    // 获取 POST 参数
    console.log(req.body);
})
```

1.5 Express 初体验

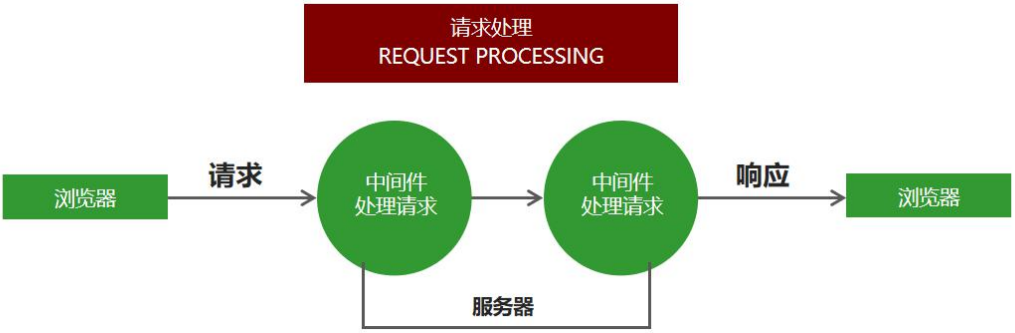
使用 Express 框架创建 web 服务器及其简单，调用 express 模块返回的函数即可。

```
// 引入 Express 框架
const express = require('express');
// 使用框架创建 web 服务器
const app = express();
// 当客户端以 get 方式访问/路由时
app.get('/', (req, res) => {
    // 对客户端做出响应  send 方法会根据内容的类型自动设置请求头
    res.send('Hello Express'); // <h2>Hello Express</h2> {say: 'hello'}
});
// 程序监听 3000 端口
app.listen(3000);
```

2. 中间件

2.1 什么是中间件

中间件就是一堆方法，可以接收客户端发来的请求、可以对请求做出响应，也可以将请求继续交给下一个中间件继续处理。



中间件主要由两部分构成，中间件方法以及请求处理函数。
中间件方法由 Express 提供，负责拦截请求，请求处理函数由开发人员提供，负责处理请求。

```
app.get('请求路径', '处理函数')   // 接收并处理 get 请求
app.post('请求路径', '处理函数')  // 接收并处理 post 请求
```

可以针对同一个请求设置多个中间件，对同一个请求进行多次处理。
默认情况下，请求从上到下依次匹配中间件，一旦匹配成功，终止匹配。
可以调用 next 方法将请求的控制权交给下一个中间件，直到遇到结束请求的中间件。

```
app.get('/request', (req, res, next) => {
    req.name = "张三";
    next();
});
app.get('/request', (req, res) => {
    res.send(req.name);
});
```

2.2 app.use 中间件用法

app.use 匹配所有的请求方式，可以直接传入请求处理函数，代表接收所有的请求。

```
app.use((req, res, next) => {
    console.log(req.url);
    next();
```

app.use 第一个参数也可以传入请求地址，代表不论什么请求方式，只要是这个请求地址就接收这个请求。

```
app.use('/admin', (req, res, next) => {
    console.log(req.url);
    next();
});
```

2.3 中间件应用

1. 路由保护，客户端在访问需要登录的页面时，可以先使用中间件判断用户登录状态，用户如果未登录，则拦截请求，直接响应，禁止用户进入需要登录的页面。
2. 网站维护公告，在所有路由的最上面定义接收所有请求的中间件，直接为客户端做出响应，网站正在维护中。
3. 自定义 404 页面

2.4 错误处理中间件

在程序执行的过程中，不可避免的会出现一些无法预料的错误，比如文件读取失败，数据库连接失败。
错误处理中间件是一个集中处理错误的地方。

```
app.use((err, req, res, next) => {
    res.status(500).send('服务器发生未知错误');
})
```

当程序出现错误时，调用 next()方法，并且将错误信息通过参数的形式传递给 next()方法，即可触发错误处理中间件。

```
app.get("/", (req, res, next) => {
    fs.readFile("/file-does-not-exist", (err, data) => {
        if (err) {
            next(err);
        }
    });
});
```

2.5 捕获错误

在 node.js 中，异步 API 的错误信息都是通过回调函数获取的，支持 Promise 对象的异步 API 发生错误可以通过 catch 方法捕获。
异步函数执行如果发生错误要如何捕获错误呢？

try catch 可以捕获异步函数以及其他同步代码在执行过程中发生的错误，但是不能其他类型的 API 发生的错误。

```
app.get("/", async (req, res, next) => {
    try {
        await User.find({name: '张三'})
    }catch(ex) {
        next(ex);
    }
});
```

3. Express 请求处理

3.1 构建模块化路由

```
// home.js
const home = express.Router();
home.get('/index', () => {
    res.send('欢迎来到博客展示页面');
```

```
// admin.js
const admin = express.Router();
admin.get('/index', () => {
    res.send('欢迎来到博客管理页面');
});
```

```
// app.js
const home = require('./route/home.js');
const admin = require('./route/admin.js');
app.use('/home', home);
app.use('/admin', admin);
```

3.2 GET 参数的获取

Express 框架中使用 req.query 即可获取 GET 参数，框架内部会将 GET 参数转换为对象并返回。

```
// 接收地址栏中问号后面的参数
// 例如: http://localhost:3000/?name=zhangsan&age=30
app.get('/', (req, res) => {
  console.log(req.query); // {"name": "zhangsan", "age": "30"}
});
```

3.3 POST 参数的获取

Express 中接收 post 请求参数需要借助第三方包 body-parser。

```
// 引入 body-parser 模块
const bodyParser = require('body-parser');
// 配置 body-parser 模块
app.use(bodyParser.urlencoded({ extended: false }));
// 接收请求
app.post('/add', (req, res) => {
    // 接收请求参数
    console.log(req.body);
})
```

3.4 Express 路由参数

```
app.get('/find/:id', (req, res) => {
    console.log(req.params); // {id: 123}
});
```

```
localhost:3000/find/123
```

3.5 静态资源的处理

通过 Express 内置的 express.static 可以方便地托管静态文件，例如 img、CSS、JavaScript 文件等。

```
app.use(express.static('public'));
```

现在，public 目录下面的文件就可以访问了。

http://localhost:3000/images/kitten.jpg

http://localhost:3000/css/style.css

http://localhost:3000/js/app.js

http://localhost:3000/images/bg.png

http://localhost:3000/hello.html

4. express-art-template 模板引擎

4.1 模板引擎

为了使art-template模板引擎能够更好的和Express框架配合，模板引擎官方在原art-template模板引擎的基础上封装了express-art-template。

使用 npm install art-template express-art-template 命令进行安装。

```
// 当渲染后缀为 art 的模板时 使用 express-art-template
app.engine('art', require('express-art-template'));
// 设置模板存放目录
app.set('views', path.join(__dirname, 'views'));
// 渲染模板时不写后缀 默认拼接 art 后缀
app.set('view engine', 'art');
app.get('/', (req, res) => {
    // 渲染模板
    res.render('index');
});
```

4.2 app.locals 对象

将变量设置到app.locals对象下面，这个数据在所有的模板中都可以获取到。

```
app.locals.users = [{
    name: '张三',
    age: 20
},{
    name: '李四',
    age: 20
}]
```

5. 实例

5.1 framework/框架入门.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

app.get('/' , (req, res) => {
    // send()
    // 1. send 方法内部会检测响应内容的类型
    // 2. send 方法会自动设置 http 状态码
    // 3. send 方法会帮我们自动设置响应的内容类型及编码
    res.send('Hello. Express');
})
app.get('/list', (req, res) => {
    res.send({name: '张三', age: 20})
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.2 framework/中间件.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

app.get('/request', (req, res, next) => {
    req.name = "张三";
    next();
})
app.get('/request', (req, res) => {
    res.send(req.name)
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.3 framework/app.use.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

// 接收所有请求的中间件
app.use((req, res, next) => {
    console.log('请求走了 app.use 中间件');
    next()
})
// 当客户端访问/request 请求的时候走当前中间件
app.use('/request', (req, res, next) => {
    console.log('请求走了 app.use / request 中间件')
    next()
})
app.get('/list', (req, res) => {
    res.send('/list')
})
app.get('/request', (req, res, next) => {
    req.name = "张三";
    next();
})
app.get('/request', (req, res) => {
    res.send(req.name)
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.4 framework/中间件应用.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

// 网站公告
// app.use((req, res, next) => {
//  res.send('当前网站正在维护...')
// })
app.use('/admin', (req, res, next) => {
    // 用户没有登录
    let isLogin = true;
    // 如果用户登录
    if (isLogin) {
        // 让请求继续向下执行
        next()
    } else {
        // 如果用户没有登录 直接对客户端做出响应
        res.send('您还没有登录 不能访问/admin 这个页面')
    }
})
app.get('/admin', (req, res) => {
    res.send('您已经登录 可以访问当前页面')
})
app.use((req, res, next) => {
    // 为客户端响应 404 状态码以及提示信息
    res.status(404).send('当前访问的页面是不存在的')
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.5 framework/错误处理中间件.js

```
// 引入 express 框架
const express = require('express');
const fs = require('fs');
// 创建网站服务器
const app = express();

app.get('/index', (req, res, next) => {
    // throw new Error('程序发生了未知错误')
    fs.readFile('./01.js', 'utf8', (err, result) => {
        if (err != null) {
            next(err)
        }else {
            res.send(result)
        }
    })
    // res.send('程序正常执行')
})
// 错误处理中间
app.use((err, req, res, next) => {
    res.status(500).send(err.message);
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.6 framework/异步函数错误的捕获.js

```
// 引入 express 框架
const express = require('express');
const fs = require('fs');
const promisify = require('util').promisify;
const readFile = promisify(fs.readFile);
// 创建网站服务器
const app = express();

app.get('/index', async (req, res, next) => {
    try {
        await readFile('./aaa.js')
    }catch (ex) {
        next(ex);
    }
})
// 错误处理中间
app.use((err, req, res, next) => {
    res.status(500).send(err.message);
})
// 监听端口
app.listen(3000);
console.log('网站服务器启动成功');
```

5.7 framework/构建模块化路由的基础代码.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();
// 创建路由对象
const home = express.Router();
// 为路由对象匹配请求路径
app.use('/home', home);
// 创建二级路由
home.get('/index', (req, res) => {
    res.send('欢迎来到博客首页页面')
})

// 端口监听
app.listen(3000);
```

5.8 framework/构建模块化路由.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

const home = require('./route/home');
const admin = require('./route/admin');
app.use('/home', home);
app.use('/admin', admin);
// 端口监听
app.listen(3000);
```

Framework/route/home.js

```
const express = require('express');

const home = express.Router();
home.get('/index', (req, res) => {
    res.send('欢迎来到博客首页页面')
});
module.exports = home;
```

Framework/route/admin.js

```
const express = require('express');

const admin = express.Router();
admin.get('/index', (req, res) => {
    res.send('欢迎来到博客管理页面')
});
module.exports = admin;
```

5.9 framework/如何获取 get 请求参数.js

```
// 引入 express 框架
const express = require('express');
// 创建网站服务器
const app = express();

app.get('/index', (req, res) => {
    // 获取 get 请求参数
    res.send(req.query)
})
// 端口监听
app.listen(3000);
```

5.10 framework/如何获取 post 请求参数.js

Framework/post.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <form action="http://localhost:3000/add" method="post">
        <input type="text" name="username">
        <input type="password" name="password">
        <input type="submit" name="">
    </form>
</body>
</html>
```

```js
// 引入 express 框架
const express = require('express');
const bodyParser = require('body-parser');
// 创建网站服务器
const app = express();
// 拦截所有请求
// extended: false 方法内部使用 querystring 模块处理请求参数的格式
// extended: true 方法内部使用第三方模块 qs 处理请求参数的格式
app.use(bodyParser.urlencoded({extended: false}))

app.post('/add', (req, res) => {
    // 接收 post 请求参数
    res.send(req.body)
})
// 端口监听
app.listen(3000);
```

5.11 framework/app.use 方法.js

```
// 引入 express 框架
const express = require('express');
const bodyParser = require('body-parser');
// 创建网站服务器
const app = express();
app.use(fn ({a: 2}))
function fn (obj) {
    return function (req, res, next) {
        if (obj.a == 1) {
            console.log(req.url)
        }else {
            console.log(req.method)
        }
        next()
    }
}
app.get('/', (req, res) => {
    // 接收 post 请求参数
    res.send('ok')
})
// 端口监听
app.listen(3000);
```

5.12 framework/路由参数.js

```
// 引入 express 框架
const express = require('express');
const bodyParser = require('body-parser');
// 创建网站服务器
const app = express();

app.get('/index/:id/:name/:age', (req, res) => {
    // 接收 post 请求参数
    res.send(req.params)
})
// 端口监听
app.listen(3000);
```

5.13 framework/静态资源访问功能.js

```
const express = require('express');
const path = require('path');
const app = express();

// 实现静态资源访问功能
app.use('/static',express.static(path.join(__dirname, 'public')))
// 端口监听
app.listen(3000);
```

5.14 framework/模板引擎.js

Framework/views/index.art

```
{{ msg }}

<ul>
    {{each users}}
    <li>
        {{$value.name}}
        {{$value.age}}
    </li>
    {{/each}}
</ul>
```

Framework/views/list.art

```
{{msg}}

<ul>
    {{each users}}
    <li>
        {{$value.name}}
        {{$value.age}}
    </li>
    {{/each}}
</ul>
```

```
const express = require('express');
const path = require('path');
const app = express();

// 1.告诉 express 框架使用什么模板引擎渲染什么后缀的模板文件
//  1.模板后缀
//  2.使用的模板疫情
app.engine('art', require('express-art-template'))
// 2.告诉 express 框架模板存放的位置是什么
app.set('views', path.join(__dirname, 'views'))
// 3.告诉 express 框架模板的默认后缀是什么
app.set('view engine', 'art');
app.get('/index', (req, res) => {
    // 1. 拼接模板路径
    // 2. 拼接模板后缀
    // 3. 哪一个模板和哪一个数据进行拼接
    // 4. 将拼接结果响应给了客户端
    res.render('index', {
        msg: 'message'
    })
});
app.get('/list', (req, res) => {
    res.render('list', {
        msg: 'list page'
    })
})
// 端口监听
app.listen(3000);
```

5.15 framework/app.locals 对象.js

```
const express = require('express');
const path = require('path');
const app = express();
// 模板配置
app.engine('art', require('express-art-template'))
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'art');

app.locals.users = [{
    name: 'zhangsan',
    age: 20
},{
    name: '李四',
    age: 30
}]
app.get('/index', (req, res) => {
    res.render('index', {
        msg: '首页'
    })
});
app.get('/list', (req, res) => {
    res.render('list', {
        msg: '列表页'
    });
})
// 端口监听
app.listen(3000);
```