link: null
title: 珠峰架构师成长计划
description: 1.构建基本服务器
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=63 sentences=373, words=2296

1.构建基本服务器

- 创建express模块,导出一个函数，执行函数可以返回一个app对象
- app对象里定义 get和 listen两个方法
- get方法用于往路由里添加一条路由规则
- 初始化router对象保存所有的路由
- listen方法用于启动一个HTTP服务器并指定处理函数

```
const express = require('../index');
const app = express();
app.get('/',function(req,res){
    res.end('hello');
});
app.listen(3000,function(){
    console.log('server started on port 3000');
});
```

2. 封装Router

- app从字面量变为Application类
- 丰富HTTP请求方法
- 封装Router
- 路径一样的路由整合为一组，引入Layer的概念
- 增加路由控制，支持next方法，并增加错误捕获功能
- 执行Router.handle的时候传入out参数

```
const express = require('../');
const app = express();

app.get('/',function(req,res,next){
    console.log(1);
    next('wrong');
},function(req,res,next){
    console.log(11);
    next();
}).get('/',function(req,res,next){
    console.log(2);
    next();
}).get('/',function(req,res,next){
    console.log(3);
    res.end('ok');
}).get('/',function(err,req,res,next){
    res.end('catch: '+err);
});
app.listen(3000);
```

```
function Route(path){
    this.path = path;
    this.methods = {};
    this.stack = [];
}

function Layer(path,handler){
    this.path = path;
    this.handler = handler;
    this.regexp = pathToRegexp(this.path,this.keys = []);
}

proto.route = function (path) {
    const route = new Route(path);
    const layer = new Layer(path, route.dispatch.bind(route));
    layer.route = route;
    this.stack.push(layer);
    return route;
}
methods.forEach(function (method) {
    proto[method] = function (path) {

        let route = this.route(path);

        route[method].apply(route, slice.call(arguments, 1));
        return this;
    }
});
lib/router/route.js
```js
function Route(path){
     +    this.path = path;
     +    this.methods = {};
     +    this.stack = [];
     +}
     +
     +methods.forEach(function(method){
     +    Route.prototype[method] = function(){
     +        const handlers = Array.from(arguments);
     +        for(let i=0;i= self.stack.length){
     +            return out(err);
     +        }
     +        let layer = self.stack[idx++];
     +        if(layer.method == req.method.toLowerCase()){
     +            layer.handle_request(req,res,next);
     +        }else{
     +            next();
     +        }
     +    }
     +    next();
     +}
```

lib/router/layer.js

```
function Layer(path,handler){
     +    this.path = path;
     +    this.handler = handler;
     +}
     +Layer.prototype.match = function(path){
     +    return this.path == path;
     +}
     +Layer.prototype.handle_request = function(req,res,next){
     +    this.handler(req,res,next);
     +}
     +Layer.prototype.handle_error = function(err,req,res,next){
     +    if(this.handler.length != 4){
     +        return next(err);
     +    }
     +    this.handle(err,req,res,next);
     +}
```

lib/application.js

```
+Application.prototype.lazyrouter = function(){
     +    if(!this._router){
     +        this._router = new Router();
               }
     +}
+methods.forEach(function(method){
     +    Application.prototype[method] = function(){
     +       this.lazyrouter();
     +       this._router[method].apply(this._router,slice.call(arguments));
     +       return this;
             }
     +});
     +
     +Application.prototype.listen = function(){
     +    const self = this;
     +    const server = http.createServer(function(req,res){
     +        function done(){
     +            res.end('Not Found');
     +        }
     +        self._router.handle(req,res,done);
     +    });
     +    server.listen.apply(server,arguments);
          }
```

lib/router/index.js
```

```
+Router.prototype.route = function(path){
+    const route = new Route(path);
+    const layer = new Layer(path,route.dispatch.bind(route));
+    layer.route = route;
+    this.stack.push(layer);
+    return route;
+}
+methods.forEach(function(method){
+    Router.prototype[method] = function(path){
+
+        let route = this.route(path);
+
+        route[method].apply(route,slice.call(arguments,1));
+        return this;
+    }
+});
+
+Router.prototype.handle = function(req,res,out){
+    let idx=0,self=this;
+    let {pathname} = url.parse(req.url,true);
+    function next(err){
+        if(idx >= self.stack.length){
+            return out(err);
+        }
+        let layer = self.stack[idx++];
+        if(layer.match(pathname) && layer.route&&layer.route._handles_method(req.method.toLowerCase())){
+            if(err){
+
+                layer.handle_error(err,req,res,next);
+            }else{
+                layer.handle_request(req,res,next);
+            }
+        }else{
+            next();
+        }
+    }
+    next();
+}
```

lib/router/route.js

```
+Route.prototype.dispatch = function(req,res,out){
+    let idx = 0,self=this;
+    function next(err){
+        if(err){
+            return out(err);
+        }
+        if(idx >= self.stack.length){
+            return out(err);
+        }
+        let layer = self.stack[idx++];
+        if(layer.method == req.method.toLowerCase()){
+            layer.handle_request(req,res,next);
+        }else{
+            next();
+        }
+    }
+    next();
+}
```

3.实现中间件

- application中添加use方法
- Router变函数
- 抽象出Router方便复用
- Router处理中间件

```
const express = require('../');
const app = express();

app.use(function(req,res,next){
    console.log('Ware1:',Date.now());
    next('wrong');
});
app.get('/',function(req,res,next){
    res.end('1');
});
const user = express.Router();
user.use(function(req,res,next){
    console.log('Ware2',Date.now());
    next();
});
user.use('/2',function(req,res,next){
    res.end('2');
});
app.use('/user',user);
app.use(function(err,req,res,next){
    res.end('catch '+err);
});
app.listen(3000,function(){
    console.log('server started at port 3000');
});
```

lib/application.js

```
+Application.prototype.use = function(handler){
    +    this.lazyrouter();
    +    let path = '/';
    +    let router = this._router;
    +    if(typeof handler != 'function'){
    +        path = handler;
    +        handler = arguments[1];
    +    }
    +    router.use(path,handler);
    +    return this;
    +}
    +
```

lib/router/index.js

```
 function Router(){
        -  this.stack = [];
    +  function router(req,res,next){
    +    router.handle(req,res,next);
    +  }
    +  Object.setPrototypeOf(router,proto);
    +  router.stack = [];
    +  return router;
         }
    +const proto = Object.create(null);

+proto.use = function(handler){
    +  let path = '/',router= this._router;
    +  if(typeof handler != 'function'){
    +      path = handler;
    +      handler = arguments[1];
    +  }
    +  let layer = new Layer(path,handler);
    +  layer.route = undefined;
    +  this.stack.push(layer);
    +  return this;
    +}

+proto.handle = function(req,res,out){
    +    let idx=0,self=this,removed='',slashAdded=false;
             let {pathname} = url.parse(req.url,true);
             function next(err){
    +        if(slashAdded){
    +            req.url = '';
    +            slashAdded = true;
    +        }
    +        if(removed.length>0){
    +            req.url = removed + req.url;
    +            removed = '';
    +        }
                 if(idx >= self.stack.length){
                     return out(err);
                 }
                 let layer = self.stack[idx++];
        -        if(layer.match(pathname) && layer.route&&layer.route._handles_method(req.method.toLowerCase())){
    +        if(layer.match(pathname)){
                     if(err){
        -
                         layer.handle_error(err,req,res,next);
                     }else{
        -               layer.handle_request(req,res,next);
    +               if(!layer.route){
    +                   let removed = layer.path;
    +                   req.url = req.url.slice(0,removed.length);
    +                   if(req.url == ''){
    +                       req.url = '/';
    +                       slashAdded = true;
    +                   }
    +                   layer.handle_request(req,res,next)
    +               }else if(layer.route._handles_method(req.method)){
    +                   layer.handle_request(req,res,next);
    +               }else{
    +                   next(err);
    +               }
                  }
                }else{
        -           next();
    +           next(err);
                  }
              }
              next();
```

lib/router/layer.js

```
if(this.path == path){
    +        return true;
    +    }
    +    if(!this.route){
    +        if(this.path == '/'){
    +            return true;
    +        }
    +        if(this.path = path.slice(0,this.path.length)){
    +            return true;
    +        }
    +    }
    +    return false;
```

4.req.params

- 可以获取 req.params
- 提供 app.param的能力
    - layer借助 path-to-regexp提取params
    - 在Router.handle里,process_params函数一次调用参数处理函数

```
const express = require('../');
const app = express();
app.param('uid',function(req,res,next,val,name){
    req.user = {id:1,name:'zfpx'};
    next();
})
app.param('uid',function(req,res,next,val,name){
    req.user.name = 'zfpx2';
    next();
})
app.get('/user/:uid',function(req,res){
    console.log(req.user);
    res.end('user');
});
app.listen(3000);
```

lib/router/layer.js

```
Layer.prototype.match = function(path){
              if(this.path == path){
                  return true;
              }
    +    if(this.route){
    +        this.params = {};
    +        let matches = this.regexp.exec(path);
    +        if(matches){
    +            for(let i=1;ilet key = this.keys[i-1];
    +                let prop = key.name;
    +                this.params[prop] = matches[i];
    +            }
    +            console.log(this.params);
    +            return true;
    +        }
    +    }
```

lib/router/index.js

```
+proto.param = function (name, handler) {
    +    if (!this.paramCallbacks[name]) {
    +        this.paramCallbacks[name] = []
    +    }
    +    this.paramCallbacks[name].push(handler);
    +}
    +proto.process_params = function (layer, req, res, done) {
    +  const paramCallbacks = this.paramCallbacks;
    +  const  keys = layer.keys;
    +  if(!keys || keys.length ==0){
    +      return done();
    +  }
    +  let keyIndex=0,name,callbacks,key,val;
    +  function param(){
    +      if(keyIndex >= keys.length){
    +          return done();
    +      }
    +      key = keys[keyIndex++];
    +      name = key.name;
    +      val = req.params[name];
    +      callbacks = paramCallbacks[name];
    +      if(!val || !callbacks){
    +          return param();
    +      }
    +      execCallback();
    +  }
    +  let callbackIndex = 0;
    +  function execCallback(){
    +      let cb = callbacks[callbackIndex++];
    +      if(!cb){
    +          return param();
    +      }
    +      cb(req,res,execCallback,val,name);
    +  }
    +  param();
    +}
```

5.模版引擎

- 如何开发或绑定一个渲染引擎
- 注册一个渲染引擎
- 指定模版路径
- 渲染模版引擎
- app.engine(ext,callback)
    - ext 文件扩展名
    - callback 模版引擎的主函数
        - 文件路径
        - 参数对象
        - 回调函数

```
const express = require('../');
const path = require('path');
const html = require('../lib/html');
const app = express();
const fs = require('fs');
app.engine('html',html);
app.set('views',path.resolve('views'));
app.set('view engine','html');
app.get('/',function(req,res,next){
    res.render('index',{title:'hello',user:{name:'zfpx'}});
});
app.listen(3000);
```

application.js

```
Application.prototype.set = function(key,val){
    if(arguments.length == 1){
        return this.settings[key];
    }
    this.settings[key] = val;
    return this;
}

Application.prototype.engine = function(ext,fn){
  let extension = ext[0]=='.'?ext:'.'+ext;
  this.engines[extension] = fn;
  return this;
}

Application.prototype.render = function(name,options,callback){
    console.log('app render');
    let engines = this.engines;
    let type = '.'+this.get('view engine');
    let render = engines[type];
    name = name.includes('.')?name:name+type;
    let file = path.join(this.get('views'),name);
    render(file,options,callback);
}

methods.forEach(function(method){
    Application.prototype[method] = function(){
        if(method == 'get'){
            if(arguments.length == 1){
                return this.set(arguments[0]);
            }
        }
       this.lazyrouter();
       this._router[method].apply(this._router,slice.call(arguments));
       return this;
    }
});

const server = http.createServer(function(req,res){
        function done(){
            res.end('Not Found');
        }
+       res.app = self;
        self._router.handle(req,res,done);
    });
```

middle/init.js

```
module.exports = function(req,res,next){
    res.render = function(filepath,options,callback){
        let self = this;
        let done = function(err,html){
            res.setHeader('Content-Type','text.html;charset=utf-8');
            res.end(html);
        }
        res.app.render(filepath,options,callback||done);
    }
    next();
}
```

```
const fs = require('fs');
function render(filepath,options,callback){
  fs.readFile(filepath,'utf8',function(err,content){
      if(err) return callback(err);
      let head = "let tpl = ``;\n with(obj){\n tpl +=`";
      content = content.replace(//g,function(){
          return "${"+arguments[1]+"}";
      });
      content = content.replace(//g,function(){
          return "`;\n"+arguments[1]+" tpl+=`";
      });
      let tail = "`\n}\nreturn tpl;";
      let html = head + content + tail;
      console.log(html);
      html = new Function('obj',html);
      html = html(options);
      return callback(null,html);
  })
}
module.exports = render;
```

lib/application.js

```
Application.prototype.set = function(key,val){
     +    if(arguments.length == 1){
     +        return this.settings[key];
     +    }
     +    this.settings[key] = val;
     +    return this;
     +}
     +
     +Application.prototype.engine = function(ext,fn){
     +  let extension = ext[0]=='.'?ext:'.'+ext;
     +  this.engines[extension] = fn;
     +  return this;
     +}
     +
     +Application.prototype.render = function(name,options,callback){
     +    console.log('app render');
     +    let engines = this.engines;
     +    let type = '.'+this.get('view engine');
     +    let render = engines[type];
     +    name = name.includes('.')?name:name+type;
     +    let file = path.join(this.get('views'),name);
     +    render(file,options,callback);
     +}
     +
           methods.forEach(function(method){
               Application.prototype[method] = function(){
     +        if(method == 'get'){
     +            if(arguments.length == 1){
     +                return this.set(arguments[0]);
     +            }
     +        }
                 this.lazyrouter();
                 this._router[method].apply(this._router,slice.call(arguments));
                 return this;
@@ -37,6 +70,7 @@ Application.prototype.listen = function(){
                   function done(){
                       res.end('Not Found');
                   }
     +        res.app = self;
                    self._router.handle(req,res,done);
               });
               server.listen.apply(server,arguments);
```

lib/html.js

```
+const fs = require('fs');
     +function render(filepath,options,callback){
     +  fs.readFile(filepath,'utf8',function(err,content){
     +      if(err) return callback(err);
     +      let head = "let tpl = ``;\n with(obj){\n tpl +=`";
     +      content = content.replace(//g,function(){
     +          return "${"+arguments[1]+"}";
     +      });
     +      content = content.replace(//g,function(){
     +          return "`;\n"+arguments[1]+" tpl+=`";
     +      });
     +      let tail = "`\n}\nreturn tpl;";
     +      let html = head + content + tail;
     +      console.log(html);
     +      html = new Function('obj',html);
     +      html = html(options);
     +      return callback(null,html);
     +  })
     +}
     +module.exports = render;
```

lib/middle/init.js

```
+module.exports = function(req,res,next){
     +    res.render = function(filepath,options,callback){
     +        let self = this;
     +        let done = function(err,html){
     +            res.setHeader('Content-Type','text.html;charset=utf-8');
     +            res.end(html);
     +        }
     +        res.app.render(filepath,options,callback||done);
     +    }
     +    next();
     +}
```