Read the Docs Template Documentation

发布 *1.5*

Read the Docs

2022 年 05 月 06 日

Introduce

1 介绍 3
- 1.1 概要 3
- 1.2 创作背景 3

2 综述 5
- 2.1 特征 5
- 2.2 安装使用 5
- 2.3 Docker 6
- 2.4 加入贡献 6
- 2.5 支持 6
- 2.6 授权 6

3 目录结构 7

4 如何快速上手 9

5 请求 | 响应的对象结构 11

6 访问关系型数据库 13

7 数据库 dbutils.js 组件 15

8 数据库表 REST 服务化 17

9 调用远程 REST 服务 19

10 访问 Redis 缓存 21

11 JSON 序列化工具 23

12 使用日志工具 25

13 静态资源服务能力 27

14 配置文件介绍 29
- 14.1 配置示例 29
- 14.2 server 30
- 14.3 endpoints 30

14.4 filters . . . 31

15 动态热部署 **33**
15.1 热部署支持 . . . 33
15.2 动态绑定 . . . 34
15.3 动态打补丁 . . . 35

16 $.format 和 $.asMapList **37**

17 三层架构 (Controller/Service/Dao) **39**

18 访问非关系型数据库 **41**
18.1 访问 MongoDB . . . 41
18.2 访问 Neo4j . . . 42

19 如何使用过滤器 **43**

20 Servlet 和 Filter 的另一种写法 **45**

21 安全控制策略扩展 **47**
21.1 Http Basic Authentication . . . 47
21.2 开启 HTTPS . . . 48

22 生成 CRUD 代码 **51**

23 服务实例 **53**
23.1 boot() . . . 53
23.2 shutdown() . . . 53
23.3 afterBoot(fn) . . . 53
23.4 afterShutdown(fn) . . . 54
23.5 status() . . . 54
23.6 bind(entry) . . . 54
23.7 unbind(entry) . . . 54

24 数据访问 **55**
24.1 jdbc(back) . . . 55
24.2 db() . . . 55
24.3 sql() . . . 55
24.4 asMapList . . . 55
24.5 query(sql,params) . . . 56
24.6 format(mapList) . . . 56
24.7 redis(back) . . . 56
24.8 mongo(db,collection) . . . 56
24.9 asDoc(obj) . . . 56
24.10 neo4j(session) . . . 56

25 HTTP-Client **59**
25.1 get(url,headers,asJson) . . . 59
25.2 post(url,headers,data,asJson) . . . 59

26 其他工具函数 **61**
26.1 empty(str) . . . 61
26.2 at(index,string|array) . . . 61
26.3 each(object|array,fn[i,n]) . . . 61
26.4 servlet(name,fn[req,resp]) . . . 62
26.5 redirect(resp,url) . . . 62

26.6 filter(name,fn[req,resp]) . . . 62
26.7 toJson(obj) . . . 62
26.8 fromJson(jsonStr) . . . 62
26.9 setTimeout(fn,time) . . . 62
26.10 setInterval(fn,time) . . . 63

27 日志 **65**
27.1 logger(name) . . . 65
27.2 debug(str) . . . 65
27.3 info(str) . . . 65
27.4 warn(str) . . . 65
27.5 error(str) . . . 66

28 秘钥证书 **67**
28.1 genkey(cfg) . . . 67

29 代码生成 **69**
29.1 gencrud(tables,asCamel) . . . 69

30 作者 **71**

31 后记 **73**
31.1 痛点 . . . 73

32 素书 **75**
32.1 第一章原章 . . . 75
32.2 第二章正道 . . . 76
32.3 第三章求人之志 . . . 76
32.4 第四章本德宗道 . . . 76
32.5 第五章道义 . . . 77
32.6 第六章安礼 . . . 78

33 止学 **79**
33.1 智卷 . . . 79
33.2 用势卷 . . . 79
33.3 利卷 . . . 79
33.4 辩卷 . . . 80
33.5 誉卷 . . . 80
33.6 情卷 . . . 80
33.7 蹇卷 . . . 80
33.8 释怨 . . . 80
33.9 心卷 . . . 80
33.10 修身卷 . . . 81

Contents:

CHAPTER 1

介绍

1.1 概要

Tropic 是一套支持跑在 JVM 上的 Web 开发框架，允许开发人员以 JS 的语法来开发后台程序。这一点有点类似 NodeJs，但是本质上又不同于 NodeJs。

```
var index = {
service: function(req, resp) {
        try {
            resp.body.append("Hello!! Welcome to Tropic engine.");
            resp.msg.append("Version:1.0");
        } catch (e) {
            println(e);
        }
        return resp;
    }
}
```

这么一段代码，启动我们的 Tropic 后，访问 http://127.0.0.1:9999/ 。即可在浏览器里看到：

```
{"code":200,"msg":"Version:1.0","body":"Hello!! Welcome to Tropic engine."}
```

1.2 创作背景

众所周知，Java 是一种比较让人又爱又恨的编程语言。你说它是编译型语言，好像也对；你说它是解释性语言，好像也没毛病。JVM 的字节码技术的确做到了一次编译到处运行，这一点也是其最大的成功之处，备受推崇。尽管它具有面向对象的能力和泛型的加持，加上 lambda 表达式的助推，按说 Java 这编程语言已经没有啥槽点让人诟病了吧。可是，放眼业内，还有很多动态脚本解释性语言（当然也有些介于解释和编译之间的），比如 PHP，Ruby on rails，Python 之 Django，以及国人后起之秀 beego。这些技术都希望热部署，及时改及时生效，使用起来真的是爽出天际，当然性能打折扣也再算难免。其实吧，Java 也不是不可以，Java 领域的 JSP 技术本身就是可以随时改随时编译运行的。你看，技术总是有很多相通之处，但是 JSP 技术也有很多

蹩脚之处，比如不够灵活，只能针对页面动态编译，如果考虑逻辑分层处理，总不至于所有的代码都往一个 JSP 页面里塞吧。Spring 的横空出世，让无数程序员眼前一亮，爱不释手。但，Spring 本质上只是个无关痛痒的框架，这个框架也越来越繁冗，API 很多设计已经有了 API 绑架的影子，而且还有很多设计上的坑。

所以，我觉得距离我想要的还差很远。于是，不如自己写个超级迷你的小框架吧，要超级轻量迷你。于是，就有了 Tropic。总是，希望热部署，那就取名回归线好了，够热。

需要澄清的是，这框架本身并不预期解决所有用 Java 语言来开发程序的能力，但是提供了一种无需编译即可运行的能力，这一点不同 OSGI 技术，更加不同于 JSP 技术。需要补充说明的是运行 JS 的能力本身就是 JVM 自带的特性之一，本框架只是稍微往前走了那么一小步，使其更容易开发一些小微服务，动态灵活热部署，同时呢又可以完全拥抱强大的 Java 开源生态。整个框架的核心代码不超过 600 行，做到超小体积。并且，辅助核心工具库的 API 设计也向 Jquery 致敬，尽可能语义化，秉持少就是多的理念。

在这里，也希望有更多志同道合的朋友一起加入，将其打造得更优秀。

CHAPTER 2

综述

Jstropic 是什么意思？准确的说应该拆开来这个单词，js 代表的是 javascript，tropic 是回归线，热带的意思。Jstropic 是一套体积很小的框架，是面向数据库到浏览器的数据处理框架，秉持少就是多的思想来提供强大的能力。

2.1 特征

- 体积小
- 简单易用
- 无需编译
- 支持热部署
- 依托 Java 强大的开源生态
- 稳定可靠的 JVM 提供保障
- 可自由扩展

2.2 安装使用

环境准备

- JDK8/11
- Tropic

下载地址:

https://github.com/letui/Tropic/releases/download/Tropic-1.4/Tropic-1.4.zip

2.3 Docker

docker pull letui/tropic:1.3

简单模式启动镜像 docker run -d -p 8080:9999 letui/tropic:1.3

映射目录模式 docker run -d -p 8080:9999 -v /home/servlet:/servlet -v /home/static:/static letui/tropic:1.3

2.4 加入贡献

- Issue Tracker: http://github.com/letui/Tropic/issues
- Source Code: http://github.com/letui/Tropic
- Home Page: http://jstr.cc/

2.5 支持

如果你有任何问题，请一定让我们知道. 邮件地址: letui@qq.com

2.6 授权

Copyright © 2021-2022, jstr.cc

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER 3

目录结构

- /bin
- /lib
- /log
- /patch
- /servlet
- /filter
- /static
- app.js
- config.js
- start.bat
- start.sh
- stop.bat
- stop.sh

在/bin 目录下有四个文件:

- /bin/bind.js
- /bin/dbutils.js
- /bin/server.js
- /bin/index.js
- /bin/static.js
- /bin/crud.js

这四个文件中 server.js 是框架的核心，index.js 是框架的默认首页响应程序脚本，等同于 index.html/index.jsp 之类 dbutils.js 是对数据库服务化的一种模板化能力的提炼，将数据库访问 SQL 能力换成另一种语法。bind.js 是整个框架的一种动态绑定能力模块，在后面的章节中展开介绍。

/lib 目录存放我们所需要的 jar 包，这些 jar 包将在程序启动时加载至 JVM 中。/log 目录是我们存放程序运行时日志的目录，所有程序产生的日志将会写入到该目录下。/patch 目录是 bind.js 所提供的能力向服务器打补丁时所提交的代码所存放的目录。/servlet 目录是我们所有开发的小程序的存放目录。/filter 目录是我们所有开发的过滤器小程序的存放目录。/static 目录是我们存放静态资源的目录，例如 html,js,css, 图片等等。app.js 是程序的主启动程序，相当于 main 函数。config.js 是我们程序的集中配置文件，所有的配置信息将在这个配置文件中进行分类配置。start.bat 和 start.sh 分别是对应 Windows 和 Linux 系统的启动脚本。stop.bat 和.stop.sh 分别是对应 Windows 和 Linux 系统的停止脚本。

CHAPTER 4

如何快速上手

现在，我们开始编写我们的一个服务端小程序。首先我们要先了解下一份标准的服务端小程序应该是什么样子。我们重新回顾下，/bin/index.js 的样子：

```
var index = {
service: function(req, resp) {
        try {
            resp.body.append("Hello!! Welcome to Tropic engine.");
            resp.msg.append("Version:1.0");
        } catch (e) {
            println(e);
        }
        return resp;
    }
};
```

整体上来讲，我们定义了一个 Javascript 变量，这个变量的名字是 index，这个变量是个 Object 类型的变量，并且拥有一个命名为 service 的方法，这个方法有两个参数，分别为 req 何 resp（对应 request 和 response）。整个方法中的代码就是向 resp 的 body 中添加了一段字符串，以及向 msg 属性中添加了另一段字符串。最后，直接 return resp; 结束。

看上去很简单，对吧？

我们可以照着它来写另一个我们可以自由发挥的服务端小程序。我们在 servlet 目录下新建一个 test1.js，其内容如下

```
var test1 = {
service: function(req, resp) {
        resp.body.append("你看，这是我的第一个服务端小程序。");
        return resp;
    }
};
```

那么，如何才能让它运行起来呢？我们还要在 config.js 文件中进行配置这个小程序要服务的 Http 路径。我们编辑下 config.js 找到:

```
endpoints: [
    {path: "/", servlet: "./bin/index.js", name: "index"}]
```

我们把上面的配置稍微改动下，成为下面一行配置的样子，为我们的请求路径/test1 绑定服务端小程序。

```
{path: "/test1", servlet: "./servlet/test1.js", name: "test1"}
```

一切完成后，我们启动起来，就可以在访问 http://localhost:9999/test1，我们将在浏览器看到我们预期的效果了。

CHAPTER 5

请求 | 响应的对象结构

在上面的小节中，我们看到一个标准的服务端小程序是一个有 service 函数的 JS 变量，这个函数具有两个参数，分别为 req 和 resp。当然，实际使用中框架本身不会对这两个参数的命名进行严格控制约束。如果你愿意，你写成 service(request,response) 也不是不可以。但是，这两个变量所对应的就是请求和响应的处理对象。第一个参数负责携带请求相关数据和能力，第二个参数负责携带响应相关数据和能力。那么他们的结构又是什么样呢?

```
request={
    headers:
    method:
    uri:
    params:
    body:
}

response={
    headers:
    body:
    code: 200,
    msg:
}
```

以上就是请求和响应参数对象的字段信息，其中 headers 是个 Map，根据 key 取出的项是 List，params 也同样如此，但是 params 是标准的 JS 变量类型所定义出来的，这一点不同于 headers 是源自 Java 内生的类型系统。method 和 uri 都是字符串类型，分别是请求方法 (GET/POST....) body 属性，一般情况下都会处理成 JSON 对象格式，当请求方法为 PATCH 的时候，会作为原生字符串格式。

响应参数的 body 是 Java 中 StringBuffer 类型，所以一般可以直接使用 append 方法进行添加想要输出的内容，msg 同样是 StringBuffer 格式，其默认值都是 new StringBuffer(""); 属性 code 默认值是 200，这个属性可以完全交由开发人员进行重新赋值。

CHAPTER 6

访问关系型数据库

在我们了解了服务端小程序如何开发之后，我们接下来尝试快速访问数据库，进行数据库的数据查询处理。通常，我们如果访问关系型数据库，那 MySQL 来举例，大致分为以下几步。

- 注册 JDBC 驱动类
- 获取数据库连接
- 执行 SQL，或者是参数化的 SQL
- 返回结果，映射处理成 POJO 集合
- 关闭数据库连接

在这里，我们假设数据库中已经有了一张表名为 person 的数据表，表中定义了 id,name,age,address,birthday,pet_id 这几列。第一步我们要做的是把数据库对应的驱动 jar 包放在 lib 目录下，同时在 config.js 文件中进行相关的数据库连接参数配置。一份完整的配置信息应该如下：

```
db: {
    url: "jdbc:mysql://192.168.10.60:3306/test",
    user: "root",
    pass: "123qwe123",
    driver: "com.mysql.cj.jdbc.Drvier",
    poolSize:10,
}
```

在完成配置后，我们即可开始开发我们的数据库访问小程序了。接着请看示例代码：

```
importPackage(org.apache.commons.dbutils, org.apache.commons.dbutils.handlers, java.
↪sql, java.util, java.time.format)
var person = {
service: function (req, resp) {
    if (req.params) {
        if (req.params.get("id")) {
            resp.body= this.queryPerson(req.params.get("id"));
        }
    }
```

(下页继续)

(续上页)

```
    resp.msg.append("OK");
    return resp;
},
queryPerson: function (id) {
    try {
        var connection = $.jdbc();
        var run = $.sql();
        var result = run.query(connection, "select id,name,birthday from person where␣
↪id = ? ", $.asMapList, parseInt(id));
        $.jdbc(connection);
        return $.format(result);
    } catch (ex) {
        //println(ex);
    }
    return "NOT——FOUND";
}
};
```

在上面的代码中，我们按照小程序规范定义了一个含有 service 函数的变量，当然也可以称之为对象。但是，与之前不同的是，这个对象中还有另一个函数，这个名为 queryPerson 的函数只有一个参数，参数名为 id，仔细看函数的实现逻辑大致为：根据传入 id 查询一个 person 表中的数据行，将 id,name,birthday 三列进行返回。

- 1. 获取连接定义一个变量来接收 $.jdbc(); 的返回值
- 2. 准备一个 SQL 执行者 $.sql();
- 3. 执行一段 SQL 语句
- 4. 还回数据库连接 $.jdbc(connection);
- 5. 对数据进行格式化处理，并返回.

此处我们没有看到定义 Java 中的 POJO 类，直接将数据经过格式化后返回，那么我们如果现在启动后，会看到什么结果呢？我们在配置文件中加入路径绑定信息：

```
{path: "/persons", servlet: "./servlet/person.js", name: "person"}
```

启动后，访问 http://localhost:9999/persons?id=2。即可看到以下内容：

```
{"code":200,"msg":"OK","body":[{"id":2,"name":"test","birthday":"2022-03-26 10:34:48"}
↪]}
```

很明显，这里 body 的集合中有一条对应了数据库中的数据行的 JSON 对象数据。并且日期完成了格式化。此时，应该有很多朋友会疑问，这个 $ 到底是个什么东西呢？关键的 5 行代码里，出现了 5 次它的身影。其实呢，$ 就是我们这个框架的核心能力的体现，我们关键的能力都将集成在这个 $ 上，这个 $ 如果是会用 Jquery 的朋友看到应该会无比亲切吧，是的，准确的说这个 $ 就是向 Jquery 的一种致敬，将 less is more 的内涵发扬光大。需要注意的是，这里对数据库的访问上是集成了连接池的能力的，用完记得使用 $.jdbc(back_var); 将数据库连接还回池中。

CHAPTER 7

数据库 dbutils.js 组件

框架中的 $ 已经提供了简便的访问数据库的能力，那么 dbutils.js 又是什么鬼东西呢？不用着急，我们慢慢来看。dbutils 本身也是标准化的一个服务端小程序，它作为框架一个标准组件而附带，当然开发者是否使用，完全取决于配置。接下来，我们先来看一段代码：

```
{
"table":"person",
"select":"id,name,address,age,pet_id",
"filter":"id > 40 and id !=52",
"limit":"0,5",
"order":"id desc"
}
```

假设，我们设计了一套低代码化的结构性查询语言，按照上面的代码来进行解读，我们大概能够得出这样的意图。我们要查询的目标表是 person，我们要查询的字段是 id,name,address,age,pet_id，我们要过滤的条件是 id > 40 and id !=52，我们限制数据返回条件是 0,5 （熟悉 mysql 的朋友很容易就理解），我们要排序的字段设置是 id desc。读完以后，作为程序员的朋友应该很清楚这就是一条 SQL 语句的另一种表达方式了。没错，这就是用 JSON 的语法来重新定义 SQL 的能力。当然，这个能力肯定是受限制的，不能完全等价于 SQL 的全部能力。

既然如此，我们的意图是当接收到一个 HTTP 请求，其携带的 Body 体是以上数据结构的时候，我们如何才能够以一种以不变应万变的来提供数据库访问能力呢？这就是我们的 dbutils.js 要来解决的事情了。下面，我们来看下 dbutils.js 的代码：

```
importPackage(org.apache.commons.dbutils, org.apache.commons.dbutils.handlers, java.
↪sql, java.util, java.time.format)
var dbutils = {
service: function (req, resp) {
    try {
        if (req.body) {
            var connection = $.jdbc();
            var run = $.sql();
            var sql = "select ".concat(req.body.select).concat(" from ").concat(req.
↪body.table).concat(" where ").concat(req.body.filter);
```

(下页继续)

(续上页)

```
            if(req.body.order){
                sql=sql.concat(" order by ").concat(req.body.order);
            }
            if(req.body.limit){
                sql=sql.concat(" limit ").concat(req.body.limit);
            }
            var result = run.query(connection, sql, $.asMapList);
            resp.body = $.format(result);
            resp.msg = "OK";
            resp.code = 200;
            $.jdbc(connection);
        }else{
            resp.code=500;
            resp.msg="request body is not provided";
        }
        return resp;
    } catch (e) {
        println(e);
        resp.msg=e;
        return resp;
    }
}
};
```

一共有三十多行代码，此时我们仍然可以看到 $ 的身影，是的，此时的 dbutils.js 就是对之前的访问数据库的另一种高度抽象，添加了些参数校验逻辑，那之前的 5 个步骤一个也没少。我们只需要在 config.js 中对其进行配置就可以启用强大的数据库服务化能力了。Tropic 框架默认会将 dbutils.js 注册到/@db 路径上，@ 符号有助于和常规路径区分开。如果你作为一个开发者不需要这样的通用能力，完全可以取消其在 config.js 中的注册配置即可。

```
{path: "/@db", servlet: "./bin/dbutils.js", name: "dbutils"}
```

以上就是对 dbutils.js 的建议型配置，喜欢自定义路径的朋友可以根据自己的喜好调整即可，这里就不做测试结果的展示了。

CHAPTER 8

数据库表 REST 服务化

可能有这么一种场景，我们需要将某一张数据库表的数据暴露成 http-rest 服务，我们的预期要求是，简单，高效，快速，轻量，安全，定制化，热部署。看，这样的要求很高了，如何才能够实现呢？如何才能够优雅的实现呢？其实，仔细思考下就明白了，我们完全可以依托 dbutils 向这些高要求高目标前进，从而达到“低代码能力”。那么，接下来我们创建一个服务端小程序，其代码如下：

```
var db_person = {
    config:{
        table: "person",
        select: "id,name,address",
        filter: "id > 1 and id !=52"
    },
    service: function (req, resp) {
        load("./bin/dbutils.js");
        req.body = this.config;
        return dbutils.service(req, resp);
    }
};
```

我们来解读下这一份小程序，其 db_person 对象，拥有一个名为 config 的属性，这个属性也是个 Object 结构，并且刚好符合我上一小节当中对 Http 请求体的 JSON 格式要求。这里就不在过度解读这个 config 的语义。我们来到 service 函数内部，仔细看后，会发现只有三行代码。这里出现了一个 load 函数，这里需要重点说明，这个函数是 native 函数是引擎自带的，其作用就是帮我们加载另一个小程序的代码，加载后，下面的代码就可以直接使用被加载的代码中所有定义的能力。紧接着我们将 req.body 直接赋值为 this.config，然后返回 dbutils.service 调用结果。到此为止，我们发现，这个小程序中没有任何代码对 HTTP 请求进行处理，只是简单的将数据库以不透明的定制化将其服务出去。并且，其 service 函数中的代码是不需要做任何的改变的，某种意义上来讲，这部分代码就是固化的，是专门针对特定数据库访问做的场景化固化能力范式代码。如果这么理解的话，我们真正的对于数据库表服务化的要求就变成了对 config 属性的设置了。只要能够理解上一小节中对 SQL 的 JSON 话语义表达定义，那么我们开发数据库服务化就简直易如反掌。

当然，此时，我们仍然是可以对 service 函数添加自己的业务逻辑的，无非是写参数校验，响应结果换一种格式等等。

CHAPTER 9

调用远程 REST 服务

随着现在很多 API 设计 WEB 化，很多微服务暴露出来的接口已经完全是承载 JSON 数据格式的 HTTP 服务。那么，这就带来更多的调用 HTTP 接口的场景，做 Java 开发的朋友肯定很熟悉一些类似 Spring-RestTemplate，Apapche-HttpClient，Ok-Http 等等开源框架。这里不一一点评各框架的优劣，Tropic 框架本身也是希望集成进来这种能力，从而方便开发。

既然，要访问 Http-rest 服务，那么我们可以直接利用之前查询 dbutils.js 所提供的 Http-rest 服务。总结下，现在的场景就变成了：我们要提供个 Http 接口服务，这个接口服务的实际实现逻辑是当你请求它的时候，它去请求另一个 Http-rest 服务，将请求回来的数据经过处理（或者不处理）再次返回浏览器（或者其他 Http-Client）。这么听上去，有点类似 Http 代理的意思。是的，本质上我们写 WEB 程序，大多数是代理了数据库的能力。所以，此时我们就新创建个小程序文件命名为 httpproxy.js。

```
var httpproxy = {
    config: {
        table: "person",
        select: "id,name,address",
        filter: "id > 50 and id !=52"
    },
    service: function (req, resp) {
        resp = $.post("http://127.0.0.1:9999/@db", null, this.config, true);
        resp.body.push("Hello!!");
        return resp;
    }
};
```

同样，我们仍然需要为这个小程序配置下 http 路径，我们在 config.js 的 endpoints 里加入以下代码:

```
{path: "/pxy", servlet: "./servlet/httpproxy.js", name: "httpproxy"}
```

接下来，启动我们的程序。在浏览器里访问 http://127.0.0.1:9999/pxy，我们将看到以下结果：

```
{"code":200,"msg":"OK","body":[{"id":50,"name":"P990.6751635416179556","address":
↪"北京市海淀区中关村22号0.8318787196947994"},{"id":51,"name":"P990.6449720409186297",
↪"address":"北京市海淀区中关村22号0.7112042891897301"},"Hello!!"]}
```

以上数据是测试数据，不同使用者并不相同，但是这里我们仔细观察，发现响应结果里面，其中 body 这个集合中多出来一条字符串数据，内容为"Hello!!"。没错，这一条内容恰恰是我们的代理小程序加进去的。其关键的代码就是上文中的 resp.body.push("Hello!!"); 这是一段典型 JS 数组的操作，就不用再做过多解释。我们把关注点转移到神奇的 $ 上，我们又一次发现了这个 $ 对象，用过 Jquery 的朋友肯定很熟悉 Jquery 的 Ajax 请求有两个很常用的 $.get 和 $.post。没错，这里也是同样的 API 设计，但是稍微有些不同。我们来仔细看下这一行代码：

```
resp = $.post("http://127.0.0.1:9999/@db", null, this.config, true);
```

在这一行代码中，post 函数总共有四个参数，第一个参数不用解释也很清楚是个 Http 地址，第二个参数是 null，这个可能不太好理解，没关系我们接着看第三个参数，第三个参数是一个 JSON-Object，同时呢这个对象刚好符合了我们数据库服务化的格式要求，第四个参数是个 bool 类型的变量，我们这里传入了 true。那么让我们解开这里的 get 和 post 的庐山真面目吧。

```
get: function (url, headers, asJson) {
}

post: function (url, headers, data, asJson) {
}
```

看到上面的代码声明时，所有的疑惑都解开了。post 函数比 get 函数多了个数据对象的传入要求，但是这个 data 并不限定是严格 JSON 对象，其他类型也是可以的，因为无论如何它都要被塞进 Http 请求报文体当中。后面的 asJson 是说响应回来的结果要不要完成 JSON 反序列化，使其成为一个 JSON 对象。至此，我想各位朋友应该没有疑问了。我们可以这样简单的使用框架自带的能力来完成 Http 请求，并且是如此少的代码。

CHAPTER 10

访问 Redis 缓存

考虑到大家在实际开发中，很大部分场景是要使用 Redis 的，所以框架本身也将 Redis 的访问能力集成了进来，当然只是集成，其依赖的核心 jar 包是我们熟知的 Jedis。那么我们来看下，如何访问 Redis 吧。

```
var redis = $.redis();
redis.set("Hi","I'm Tropic");
redis.get("Hi");
$.redis(redis);
```

相信看了上面的代码，所有的朋友应该很容易理解了。这里的 $.redis(); 采用了和 $.jdbc(); 同样的设计，如果你调用函数没有传递任何参数那么就是获取一个 Jedis 对象，当你用完了以后，你需要把它还回去，使用方式就是同样的 $.redis(back); 这里传入刚才的返回值就可以了。细心的朋友可能会疑惑，那么我们的 Redis 要如何配置呢？是的，我们当然需要完成初始化配置。我们来看下 Redis 的连接配置长什么样子。

```
redis:{
  host:"192.168.10.173",
  port:6379,
  password: "123qwe123",
  maxIdle:10,
  maxTotal:20,
  maxTimeout:2000
}
```

```
redis:{
    clusters: [{host: "192.168.10.173", port: 7000}
        , {host: "192.168.10.173", port: 7001}
        , {host: "192.168.10.173", port: 7002}
        , {host: "192.168.10.173", port: 7003}
        , {host: "192.168.10.173", port: 7004}
        , {host: "192.168.10.173", port: 7005}],
    password: "123qwe123",
  maxIdle:10,
  maxTotal:20,
  maxTimeout:2000
}
```

以上就是在 config.js 中配置 redis 连接完整信息，支持比如密码啊，集群。我们看到两种不同的配置，第一种是单机版 Redis 配置；第二种是集群吧 Redis 配置。需要说明的是，如果两者同时存在，框架会默认采用集群配置而忽略单机配置。至于详细的 Redis 访问 API 我们就不在这里详细展示了，完全可以参考 Jedis 的 API。另外，这里也就不在做完整的 Redis 访问能力的服务端小程序的示例代码了。

CHAPTER 11

JSON 序列化工具

目前，业内已经普遍认可 JSON 语法来表达数据结构，我们很多朋友也会用很多像 fastxml-json，gson，fastjson 等等一些框架，这些框架功能很强大帮我们省去了很多 Json 序列化和反序列化的工作，我们经常会使用这些框架来完成 Entity 或者 VO 之类的序列化要求。但是，其实这都是在 Java 这种强类型编程语言上的一种妥协方案而已。然而，如果我们转移到 JS 领域，我们会发现一切都变了，我们发现声明式的 JSON 数据结构可以随时使用，熟练掌握 JS 的朋友肯定都知道默认 JSON.parse 和 JSON.stringify 这两个函数。不过，为了统一 API 风格，简便开发，框架的核心对象 $ 也提供了 JSON 序列化和反序列化的函数。其分别为 $.toJson 和 $.fromJson; 这两个函数名，用过谷歌 Gson 的应该再熟悉不过了。

那么，我们就来不厌其烦的演示下如何使用这两个方法吧。

```
var input="{\n" +
    "    \"table\":\"person\",\n" +
    "    \"select\":\"id,name,address,age,pet_id\",\n" +
    "    \"filter\":\"id > 40 and id !=52\",\n" +
    "    \"limit\":\"0,5\",\n" +
    "    \"order\":\"id desc\"\n" +
    "}";

var obj=$.fromJson(input);
println(obj.table);
obj.table="NPerson";
var after=$.toJson(obj);
println(after);
```

不出意外的话我们将在控制台看到两行输出：

```
person
{"table":"NPerson","select":"id,name,address,age,pet_id","filter":"id > 40 and id !=52
↪","limit":"0,5","order":"id desc"}
```

CHAPTER 12

使用日志工具

为了方便程序的开发和调试，很多时候需要用到日志框架，在 Tropic 里，默认是集成了 logback 作为日志工具框架的。那么，应该如何使用呢？配置文件又在哪里呢？之前讲到过整个框架的目录结构，在/log 目录下放了一个 logback.xml，这个文件中便是默认的日志配置相关信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false">
    <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->
    <property name="LOG_HOME" value="./log" />
    <!-- 控制台输出 -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <!--格式化输出：%d表示日期，%thread表示线程名，%-
↪5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg
↪%n</pattern>
        </encoder>
    </appender>
    <!-- 按照每天生成日志文件 -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!--日志文件输出的文件名-->
            <FileNamePattern>${LOG_HOME}/Tropic_.%d{yyyy-MM-dd}.log</FileNamePattern>
            <!--日志文件保留天数-->
            <MaxHistory>30</MaxHistory>
        </rollingPolicy>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <!--格式化输出：%d表示日期，%thread表示线程名，%-
↪5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg
↪%n</pattern>
        </encoder>
        <!--日志文件最大的大小-->
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy
↪">
```

(下页继续)

(续上页)

```
            <MaxFileSize>10MB</MaxFileSize>
        </triggeringPolicy>
    </appender>

    <!-- 日志输出级别 -->
    <root level="INFO">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="FILE" />
    </root>
</configuration>
```

当以上配置不能满足实际需要时，开发者完全可以根据自己的实际需要来进行调整。当然，建议开发者将日志的输出存放在/log 目录下。日志配置的相关介绍相信已经没什么需要交代的了，最多就是日志框架的替换选择。其实，Tropic 本身不对框架做任何强制性要求，因为 Tropic 本身顶层只需要遵循 slf4j-api 的日志规范即可任何一种 slf4j-api 的实现都可以无缝替换进来，只是简单的把/lib 目录里相关的日志 jar 包进行替换即可，这一点完全可以交给使用者自己定夺。

下面，我们来简单介绍下如何在小程序代码中使用日志功能。我们来看一段小程序代码：

```
var logtest={
    service:function(req,resp){
        var log=$.logger("loggtest");
        log.info("Hello,this is info-level's text");
        log.debug("Hello,this is debug-level's text");
        log.warn("Hello,this is warn-level's text");
        return resp;
    }
};
```

```
2022-01-13 14:12:11.971 [pool-1-thread-1] INFO  loggtest - Hello,this is info-level's␣
↪text
2022-01-13 14:12:11.973 [pool-1-thread-1] WARN  loggtest - Hello,this is warn-level's␣
↪text
```

不出意外的话，我们的服务端小程序在接收到 Http 请求后，后台的控制窗口里将出现上面的输入信息。可是为什么没有 DEBUG 级别的调试日志呢？那是因为我们默认配置的日志级别就是 INFO 级别，如果需要打开 DEBUG 的日志输出，那就需要自行设置成为 DEBUG 级别。

值得要说的是，在示例服务端小程序代码中，我们又一次看到了 $ 的身影，我们获取一个日志控制对象使用了 $.logger("loggtest"); 的方法。是的，Tropic 将获取日志的功能集成了进来，只需要调用 $.logger 方法就可以，括号中传入的是我们的日志记录器的名字。不过不用担心，即便多次调用传入相同名字，其内部代码并不会多次创建，而是会检测命名是否已经存在，如果已经有了那就直接返回，无需创建。这里与 JDBC 连接不同的是，$.logger 方法不用归还。

CHAPTER 13

静态资源服务能力

作为一个 Web 开发框架，很多开发者肯定会希望有静态资源的服务能力，比如 JS、CSS、HTML 和图片等等。Tropic 框架也支持服务静态资源，那么要如何打开这个功能呢？还记得之前的配置文件里 server 属性，在其子级加一个叫做 static_resource 的属性，其值是一个数组，数组中包含了认为是静态文件的后缀，当有这些后缀的 HTTP 请求就会当做是静态资源来处理，这些对应的文件都放在/static 目录下。如果还记得目录结构的话，在/bin 目录下有个 static.js，没错就是这个 js 脚本来处理静态资源的访问，它会根据请求路径映射在/static 目录下去寻找，当找不到时就会返回 404 错误。

```
server: {
    port: 9999,
    threads: 200,
    use_dynamic_bind: true,
    auth_bind_token: "Tropic",
    static_resource: [".js", ".css", ".html", ".png"]
}
```

开启了静态资源服务的配置，应该是上面这样。数组里是开放了的文件类型后缀，不在这个数组里配置的文件类型后缀是不被允许访问的。在/static 目录下有个默认的 index.html，其代码如下:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tropic</title>
</head>
<body>
<h1>Hello,I'm the static resource in folder "static"</h1>
</body>
</html>
```

在我们启动好后，访问地址:http://localhost:9999/index.html 就可以看到 html 标题内容了，此时你就拥有了一个静态资源 Http 服务器。如果你需要开发其他静态 HTML 或者 JS、CSS 之类，可以直接扔在/static 目录下，就可以直接在浏览器访问。

CHAPTER 14

配置文件介绍

任何一个框架终究免不了一些配置，Tropic 同样如此。由于 Tropic 本身由 Javascript 脚本语言所开发，秉持着配置即代码，代码即配置的理念而设计。所以，整个 Tropic 的配置文件实际上就是个全局 Javascript 变量，只不过是个比较大的 Json-Object 而已。整个配置中会有服务器监听端口，并行线程数，数据库驱动配置信息，Redis 连接信息，Http 工具默认超时时间等等。之前，提到过由于其本身是个全局 Javascript 变量，这就意味着任何一个服务端小程序的代码里都可以直接访问使用，其名字为:config。在此要说明的是，Tropic 并不限制该配置变量的其他额外配置信息加入。也就是说，任何一个开发者都可以根据自己的实际情况向 config 中添加自己需要的配置信息。在使用时，只需要根据 Javscript 的对象导航语法就可以完成设置或者读取。

14.1 配置示例

```
var config =
{
    db: {
        url: "jdbc:mysql://127.0.0.1:3306/test",
        user: "root",
        pass: "123qwe123",
        driver: "com.mysql.cj.jdbc.Drvier",
        poolSize:10,
    },
    redis:{
      host:"192.168.10.173",
      port:6379,
      maxIdle:10,
      maxTotal:20,
      maxTimeout:2000
    },
    http:{
        connect_timeout:3000,
        read_timeout:3000
    },
    server: {
```

(下页继续)

(续上页)

```
        port: 9999,
        threads: 200,
        use_dynamic_bind:true,
        auth_bind_token:"Tropic",
    },
    endpoints: [
        {path: "/", servlet: "./bin/index.js", name: "index"},
        {path: "/@db", servlet: "./bin/dbutils.js", name: "dbutils"},
        {path: "/db/person", servlet: "./servlet/db_person.js", name: "db_person"},
        {path: "/person", servlet: "./servlet/person.js", name: "person"},
        {path: "/pxy", servlet: "./servlet/httpproxy.js", name: "httpproxy"}
    ]
};
```

目前我们看到的这一份完整的配置信息，所有的一级属性字段都是不可以私自改变命名的。因为，这里所有展示的配置项都是 Tropic 框架本身所使用的。

- db 配置数据库连接相关
- redis 配置 redis 相关
- http 配置 Http 工具超时
- server 配置 Tropic 启动参数，详细属性后面会展开介绍
- endpoints 配置所有需要注册加载的服务端小程序

14.2 server

目前针对 server 的配置项只有四个，分别是 port,threads,use_dynamic_bind,auth_bind_token。前面两个相信很多人很容易就理解了，所以就展开介绍下后面两个，这两个配置具体来说是 Tropic 的一个高级功能，当使用了动态绑定能力时，Tropic-Server 允许你向一个特殊的路径发起控制请求。这个请求是用来实时启用或停用某一个服务端小程序的。这个功能是个很强大的功能，但同时也很危险，所以呢就需要引入一个安全令牌来保障。这就是另外一个 auth_bind_token 的作用。auth_bind_token 所配置的令牌内容将作为动态绑定控制请求的安全校验，也就是说发起动态绑定控制请求时，必须携带令牌，这个令牌的内容必须和配置的令牌保持一致才可以得以成功处理。

14.3 endpoints

这个配置项不难理解，这就是所有服务端小程序要注册的地方，大致上类似我们用 Spring 时的所有 Controller 的注册。这个配置项是个数组格式，其中每一项都要求是标准的 Json-Object 格式，拥有三个属性，分别是 path,servlet,name。

- path 服务端小程序要服务的路径
- servlet 服务端小程序源代码在服务器上存放的路径，一把建议使用相对路径./servlet/xxx.js，放在 Tropic 框架的 servlet 目录下，这里并不限制子级目录

的使用 * name 我们的任何一个服务端小程序都是个 Javascript 变量，Object 类型，务必持有一个 service(req,resp) 格式的函数，返回值是 resp。这个 name 要求就是所定义出的变量的名字，这一点就有点类似 Spring 单例的概念，等同于 beanName 的意义。

14.4 filters

过滤器和 servlet 的配置没有什么不同，在写法上也没有什么不同。这里启用单独配置，完全是为了分开独立管理，便于维护。

CHAPTER 15

动态热部署

这个章节里我们要展开什么内容呢？准确的说，这部分内容不同于之前的简单使用帮助说明。

- 假如你的程序已经部署上线了，那怎么样才能不停止服务器的情况下来进行升级个别服务端小程序呢？或者说，这种行为叫做打补丁。
- 打补丁的安全如何保障？
- 假如你想要临时停掉某个 Http-Path 的服务端小程序，这又该怎么办呢？
- 假如你想恢复停掉的那个 Http-Path 的服务端小程序，怎么办？
- 假如你不想用 Tropic 来开发 Web 程序，只想用它来一些控制台小程序，又该如何呢？

我们有这么多假如，那么就来看 Tropic 是如何做到的吧。

15.1 热部署支持

无需编译，热部署，总会有无数程序员讨厌编译的繁琐，对着这两大特性垂涎三尺。其实呢，JVM 作为一个中间代码解释器，本身就是支持热部署的，有很多字节码处理框架完全可以在运行时来改变某个 Java 类的结构和方法，这一点在面向切面技术领域已经屡试不爽了。由于 Tropic 本身不使用 Java 字节码技术，而是简单的使用 Java 的 Javascript 引擎能力，所以热部署这个能力就浑然天成了。

接下来演示下，热部署的效果。我们创建个服务端小程序 hot.js，其内容如下:

```
//path: /hot
var hot = {
    service: function (req, resp) {
      resp.body.append("我喜欢简单。");
      return resp;
    }
};
```

我们完成配置后，启动服务器，访问 http://localhost:9999/hot。将会在浏览器看到：

```
{"code":200,"msg":"","body":"我喜欢简单。"}
```

现在，我们将小程序的代码改以下内容:

```
//path: /hot
var hot = {
    service: function (req, resp) {
      resp.body.append("我喜欢热部署能力。");
      return resp;
    }
};
```

此时，只需要保存以上代码，在不重新启动服务器的同时，我们重新刷新浏览器，我们将在浏览器看到以下内容:

```
{"code":200,"msg":"","body":"我喜欢热部署能力。"}
```

不用为之惊叹，就是这么任性。此时此刻，是不是有点 PHP 的意思了？原来 Java 给我们留着这么大的一个惊喜，可是却鲜有人去挖掘。这么爽的特性，显然要比频繁重启好使多了。

15.2 动态绑定

动态绑定能力是指允许程序运行期间，动态的将服务端小程序绑定到某个预期的 Http-path 上来提供服务。我们这里拿之前的 hot.js 接着举例示范。当然，首先要记得在我们的 config.js 中打开这个黑科技。具体配置如下：

```
server:{
    use_dynamic_bind:true,
    auth_bind_token:"Tropic"
}
```

另外，我们还需要准备一个 Http-Client 测试工具，比如 Postman。一切准备就绪后，我们打开 Postman。假设，我们需要另一个/sohot 路径提供和/hot 同样的能力那么此时，我们准备好以下内容：

动态绑定功能的服务地址是 http://localhost:9999/@bind

我们要发送的报文内容是:

```
{
"path":"/sohot",
"servlet":"./servlet/hot.js",
"name":"hot"
}
```

准备好这些还不够，因为处于安全考虑，我们必须携带 token 才可以成功请求。token 是携带在 http 请求头里的，其名称为 js$auth_bind_token，我们在 Postman 设置 js$auth_bind_token 对应的值为:Tropic。最后，还有一点需要注意，否则是无法成功的。处于安全考虑，由于 POST 请求太过普通，所以这个动态绑定的功能使用了 PUT 请求作为准入限制，请一定记得设置 HTTP 请求方法为 PUT。一切都准备好后，我们用 Postman 发起请求，不出意外将返回以下内容:

当我们收到这样的返回结果时就代表我们已经绑定成功了，此时，我们访问浏览器地址 http://localhost:9999/sohot，将看到以下内容:

那么如何解绑定呢？

其实解绑定和绑定的动作很相似，地址都是/@bind 路径来提供服务，只是解绑定的时候我们需要使用 HTTP 的 DELETE 请求方法，请求头里依然要携带令牌，但是请求体里可以只携带一个 path 属性即可。

```
{
"path":"/sohot"
}
```

特别需要注意的是，所有动态绑定的小程序路径，在服务器重启后自动失效。

15.3 动态打补丁

动态绑定能力已经很强大了，对吧？但其实，更强大的是动态打补丁的能力。这个功能准确的描述来说，是指当你已经上线了一套服务端应用，此时你无法到到服务器上更换所有的源代码了，这时候就该动态打补丁的功能闪亮登场了。这个功能允许你上传一个服务端程序源代码，并且完成绑定到一个固定的 Http 路径上来提供服务。警告，这个功能已经和黑客所熟知的 WebShell 有些类似了，是个强大但危险的功能。

那我们接下来具体介绍如何使用这个强大的打补丁功能吧。假设你有以下源代码想要提交到服务器上提供 Http 服务。

```
//path: /door.jsp
var hot = {
    service: function (req, resp) {
      resp.body.append("我是个补丁。我更像个后门。");
      return resp;
    }
};
```

源代码准备好后，我们打开 Postman，键入之前的动态绑定地址 http://localhost:9999/@bind。将源代码黏贴在 body 输入区域中。此外，我们还有好多 Http -header 要进行设置，因为我们要高速服务端这个源代码存放的文件名，服务的路径等等。那么 Http-header 里要填写哪些内容呢？

- js$path 要服务的 Http 路径此处示例应该填写/door.jsp
- js$servlet 源代码在服务器上的文件名此处示例应该填写 hot.js
- js$name 源代码中声明的变量名称此处示例应该填写 hot
- js$auth_bind_token 安全令牌此处示例应该填写 Tropic

以上信息都设置好后，我们将 HTTP 请求方法调整为 PATCH，然后点击 Postman 发送按钮。一切万事大吉后，我们将收到：

```
{
    "code": 200,
    "msg": "patch for path: [/door.jsp]",
    "body": ""
}
```

此时，我们的补丁源码就会发送到服务器上，并且开始为 Http 路径为 /door.jsp 的地址提供服务。

特别需要注意的是，所有的补丁程序在服务器重启后将会全部失效，但是/patch 目录下将会保留所有的源代码文件。因为补丁终究是补丁，补丁的服务都应该是临时的当服务器需要重启的时候，应该已经达到了人工介入更新整体服务应用的时机。

CHAPTER 16

$.format 和 $.asMapList

这两个能力在之前的章节中的示例源代码里出现过，那么到底是什么意思呢？

这里，就展开解释下 Tropic 框架集成的查询关系数据库的依赖 jar 包 commons-dbutils。commons-dbutils 是 Apache 的一个开源数据库访问处理工具包，提供了简单易用的一些 API 封装，感兴趣的可以访问:https://commons.apache.org/proper/commons-dbutils/

其核心工具类主要是两个，一个是 QueryRunner，另一个是 ResultSetHandler，这两个一个负责执行 SQL，另一个负责将查询出的数据进行处理。在官方提供的 ResultSetHandler 里有一个 MapListHander 实现，作用是将查出的每一行数据处理成一个 Map，列名作 key，列值做 value，多行数据经过转换后放进一个 ArrayList 里。$.asMapList 就是一个语法糖，免去了写代码时 new MapListHandler() 的操作。

那 $.format 是干啥呢？为什么要用 $.format 呢？$.format 的目的是要做一个通用格式化的封装。目前呢主要是用来格式化 MapListHandler 返回的数据结构，因为数据库中难免会有些 Date 和 Datetime 类型的字段，这些类型是没有办法直接映射成 Js 变量类型的，在进行 toJson 的时候会有些问题，所以就需要对齐进行一个预处理。这个函数会默认将数据库 Date 列格式化成为 yyyy-MM-dd，将数据库 Datetime 列格式化为 yyyy-MM-dd HH:mm:ss 格式。

其具体实现很简单，双重循环，按照类型判断，找到日期、时间类型的字段，进行格式化处理。

```
function (maplist) {
    var list = [];
    for (var item in maplist) {
        var row = {};
        for (var key in maplist.get(item)) {
            var val = maplist.get(item).get(key);
            if (val != null && typeof val == "function") {
                if (val instanceof LocalDateTime) {
                    row[key] = val.format(DateTimeFormatter.ofPattern("yyyy-MM-dd␣
↪HH:mm:ss"));
                } else if (val instanceof LocalDate) {
                    row[key] = val.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
                }
            } else {
                row[key] = val;
            }
```

(下页继续)

(续上页)

```
        }
        list.push(row);
    }
    return list;
}
```

看了上面的实现代码，很容易就理解了。经过这一系列的转换之后呢，Java 的类型就被抹掉了取而代之的是一个填满了 Json-Object 的数组。在后面的数据使用时我们就可以使用对象导航的方式了。

最后，补充要说的是，commons-dbutils 的功能很强大，有很多 ResultSetHandler 的默认实现，也提供了 POJO 类到查询结果集的自动化 ORM 处理工具类，这里我们不妨来看下官网的示例代码。

```
QueryRunner run = new QueryRunner(dataSource);

// Use the BeanListHandler implementation to convert all
// ResultSet rows into a List of Person JavaBeans.
ResultSetHandler<List<Person>> h = new BeanListHandler<Person>(Person.class);

// Execute the SQL statement and return the results in a List of
// Person objects generated by the BeanListHandler.
List<Person> persons = run.query("SELECT * FROM Person", h);
```

上面的代码是映射成 POJO 类的集合，可是在 Tropic 框架的使用背景下，我们需要思考个问题：用 JS 也要强制按照 Java 实体类那样去写实体类吗？包括 Getter 和 Setter？这是个问题，没有答案，没有标准，只有适合不适合，我们完全可以根据自己的实际情况来做出开发规范。作为这个框架/引擎的开发者，我认为，我都用 JS 语法了，我还蹩脚的去搞 Getter/Setter 干啥呢？

CHAPTER 17

三层架构 (Controller/Service/Dao)

以往，我们用 Spring 开发 JavaWeb 应用，基本上清一色的 Controller->Service->Dao。还有一些代码生成的框架也是按照这个路子进行设计，使用时直接生成代码。

那么用 Tropic 开发，还需要吗？如果需要的话，又应该如何去实现呢？其实，即便用 Tropic 也完全可以沿用之前的分层架构去写代码。

- Controller

```
var person_ctrl={
  service:function (req,resp){
      println($.toJson(resp));
      load("./servlet/demo/person_service.js");
      if(req.params){
          var id=req.params.get("id");
          if(id==null){
              resp.code=500;
              resp.msg.append("id不可以为null");
          }else{
             var rst= person_service.queryOneById(id);
             resp.body=rst;
          }
          return resp;
      }else{
          resp.code=500;
          resp.msg.append("请携带id参数查询");
          return resp;
      }
  }
};
```

- Service

```
var person_service = {
    queryOneById: function (id) {
        load("./servlet/demo/person_dao.js");
```

(下页继续)

(续上页)

```
        var sql="select * from person where id = "+id;
        var rst=person_dao.query(sql);
        return rst;
    }
};
```

- DAO

```
var person_dao = {
    query: function (sql) {
        var conn=$.jdbc();
        var rn=$.sql();
        var obj=rn.query(conn,sql,$.asMapList);
        obj=$.format(obj);
        $.jdbc(conn);
        return obj;
    }
};
```

上面我们展示了三层架构的方式来写代码，当然这些示例代码都很简陋。不过，我们需要注意 load 方法，这个方法是将我们三个代码源文件串起来的函数，由于我们每个源文件都是声明式的对象变量，所以我们想使用就需要加载进来。

另外，必须要从应用的根级目录来进行加载./就是指当前的框架 home 目录。

CHAPTER 18

访问非关系型数据库

18.1 访问 MongoDB

MongoDB 是业内比较知名的 NoSQL 数据库，这里不做点评，只展示如何集成 MongoDB 并完成数据操作等等。老规矩，上代码:

```
var mongo_servlet = {
service: function (req, resp) {
    var db = $.mongo("local");
    var iter = db.listCollectionNames().iterator();
    var respCols = [];
    while (iter.hasNext()) {
        respCols.push(iter.next());
    }
    var cols = $.mongo("local", "test");
    cols.insertOne($.asDoc({name: "王逊", age: 29}));
    iter = cols.find($.asDoc({age: {$gt: 20}})).iterator();
    var rows = [];
    while (iter.hasNext()) {
        rows.push($.fromJson(iter.next().toJson()));
    }
    resp.body = {cols: respCols, rowsInTest: rows};

    return resp;

}
}
```

是的，我们为了方便观察，还是写一个 Servlet 更合适不过，在上面的代码中万能的 $ 再次出现了。这次是 $.mongo(); 这个函数允许使用者传两个参数，第一个是 databaseName 第二个是，位于第一个 databaseName 下的 CollectionName。上面代码的大意是，获取一个指定的 database，并遍历出其下的所有 Collection Name，获取一个名为 test 的 Collection，完成一次数据插入，并完成一次数据查询，其查询条件是 age > 20（这里用了 mongoDB 专用的查询语法），根据查询出的结果遍历并组装成响应结果。

这里必须点出三个 Java 类: * com.mongodb.client.MongoDatabase * com.mongodb.client.MongoCollection * org.bson.Document

准确的说，MongoDB 的交互是依靠 org.bson.Document 的，其查询的输入和输出都是这个 Document 来承载。也就是说，如果想对 database 进行操作，请查阅 MongoDatabase 的 API 即可，如果想对 Collection 进行操作，查阅 MongoCollection 的 API 即可。另外，值得注意的是，在查询出的 Document 进行遍历是使用一次 toJson，又使用了一次 fromJson。这里第一次 toJson 只是 Document 的内部格式化为 JSON 字符串的方法，但是如果我们要使用 Javascript 中 JsonObject 来操作就需要 $.fromJson 函数将其转化为 JS-Object。

那么，MongoDB 在配置文件中又该如何配置呢?

```
mongo:{
        uri:"mongodb://localhost:27017/?maxPoolSize=20&w=majority"
    }
```

加入以上代码在配置 config 中即可，至于这个 uri 的更多细节，还请移步至 mongodb 的官网。

18.2 访问 Neo4j

Neo4j 作为数据分析领域的专业图算法数据库的领导者，备受推崇。自然，加入访问 Neo4j 的支持也是必须的。

```
var neo4j_servlet = {
    service: function (req, resp) {
        var session = $.neo4j(true);
        var rst = session.run("MATCH (n:Tag) RETURN n LIMIT 25");
        var respArray = [];
        while (rst.hasNext()) {
            var row = rst.next().get("n");
            var obj = {
                name: row.get("name").asString()
                , level: row.get("value").asString()
            };
            respArray.push(obj);
        }
        resp.body = respArray;
        return resp;
    }
}
```

同样，还是作为 Servlet 小程序奉上，$.neo4j 这个函数允许你传入一个参数，在实际使用中如果传入 true 则返回 Neof4j 的 API 中提供的 Session，如果不传则返回 Driver。后面的代码则是 Neo4j 的 Cypher 语言。当我们得到一个结果集后就可以遍历按照数据结构进行组织处理。这个 rst.next().get(”n”) 当中的”n” 代表的是 Cypher 语句中 RETURN n 里的 n。后面的每一行数据 row.get(”xxx”) 则是对应的节点数据的属性名，类似一个 Map。那么，配置信息长什么样呢?

CHAPTER 19

如何使用过滤器

做过 Java Web 开发的朋友肯定都知道过滤器的存在，当我们想要对某些路径整体进行处理的时候会用到过滤器，比如检查用户是否登录，字符编码统一设置等等。Tropic 框架也同样支持过滤器，在框架中过滤器采用前缀匹配过滤，不支持正则或者后缀过滤。同样，过滤器作为一种服务端小程序，本质上和 servlet 没有区别所以在配置上也并没有什么不同，只不过过滤器应该配置在 config.filters 下，而 servlet 配置在 config.endpoints 下。与 servlet 配置相同的是都需要有 path,servlet,name 三个属性的配置。一个典型的 filters 配置应该如下:

```
filters: [
        {path: "/", servlet: "./filter/corefilter.js", name: "corefilter"}
]
```

看了上面的配置，会发现 filters 的配置的确和 servlet 没有什么不同，但值得注意的是进行 servlet 属性配置的时候，示例中用了./filter 目录而非 servlet 目录框架本身建议将 servlet 和 filter 分开放置。

那么除了配置相同以外，又该如何编写一个过滤器呢?

```
var corefilter={
    service:function(req,resp){
        $.logger().info(req.uri);
    }
}
```

上面的代码展示了一个过滤器的代码，这个过滤器会对每个请求的 path 进行打印。同样，看到了完整的 filter 代码，其开发上和 servlet 也没有什么不同，如果非要说不同，那么可能是没有 return resp; 这一行代码。其实在 servlet 中也不强制要求 return resp;。

CHAPTER 20

Servlet 和 Filter 的另一种写法

常规写法是 var xxx={service:function(req,resp){ }}; 顾名思义，就是要声明一个包含了 service 函数的对象。如果你不喜欢这种对象声明式的写法，那么我们来看看另一种写法：

```
$.servlet("hot",function(req,resp){
    resp.body.append("我喜欢热部署能力。");
    return resp;
});
```

这种写法等同于:

```
var hot = {
 service: function (req, resp) {
   resp.body.append("我喜欢热部署能力。");
   return resp;
 }
};
```

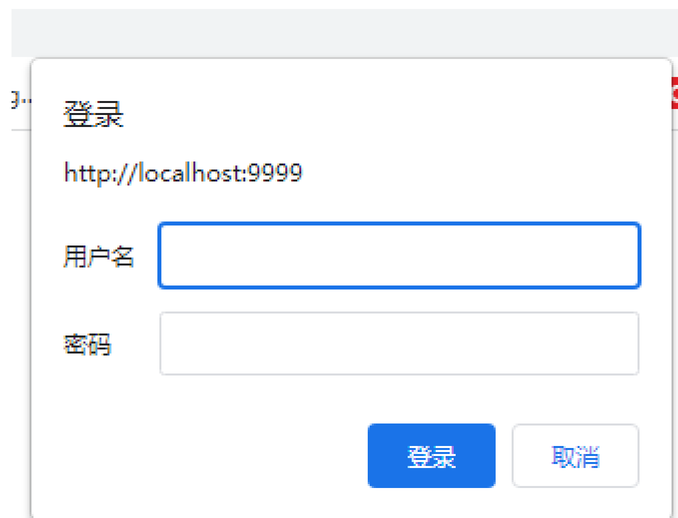
但是，显然第二种允许在对象上定义出更多的属性或者其他方法。第一种只适合比较简单的单一服务处理方法。另外，如果想用另一方式写过滤器，只需要 $.filter(name,function)。同样的方式即可。

CHAPTER 21

安全控制策略扩展

21.1 Http Basic Authentication

通常浏览器都支持 Basic Authentication，即在访问的时候会有个弹出窗口提示务必输入用户和密码。如下图:



Tropic 提供了 Basic Authentication 功能的配置化支持，只需要在 config.js 中的 server 下添加以下配置即可。

```
basic_auth_enable:true,
basic_auth_user:"admin",
basic_auth_pass:"admin",
```

无论如何，这种基本的认证控制都是很入门的，以上将认证的用户和密码都设置为了 admin，这就代表了所有的服务路径都将经过认证后才可以被访问。在开启了认证后，我们在 Postman 中测试接口时，就要对 Postman 的认证功能进行设置，详情见下图:

在进行了对应的设置后，就可以像以往一样进行测试 API 接口了。

21.2 开启 HTTPS

比较资深的开发者都清楚 HTTP 是在网上裸奔的，很容易遭受中间人攻击，于是有了 HTTPS。至于 HTTPS 的安全原理，这里就不展开□嗦。我们还是主要示范下如何将 Tropic 的 HTTPS 能力体现出来吧。

我们要先配置下秘钥库生成组件，也就是/bin 目录下的 keytool.js。另外，配置下允许静态资源服务，加入 html/js 的服务能力。在启动 Tropic 后，我们访问 http://127.0.0.1:9999/keytool.html，这是一个帮助我们生成秘钥库的工具页面。其内容应该如下：

此时，我们依次填入输入框中应该填写的内容，点击下方“生成”按钮。生成成功后，将会有 Alert 提示。特

别需要注意的是，秘钥库文件名是指在 Tropic 的主目录下的文件名，你所填写的值将是秘钥库文件的名字。在生成完毕后，我们需要停掉服务器。这时，打开 config.js，在 server 级下添加以下配置即可。

```
https_enable:true,
key_store_path:"./tropic.keystore",
key_store_pass:"tropic123456",
key_pass:"tropic123456",
```

一切配置完毕以后，我们重新启动服务，当再次在地址栏键入地址的时候，就需要完整的写 https://127.0.0.1:9999/，否则将会访问不到。启用了 Https 之后，所有的服务端小程序的响应都将会承载在 https 上。

CHAPTER 22

生成 CRUD 代码

Tropic 提供了生成 CRUD 代码的能力组件，如此一来，我们可以快速生成模板式的增删改查的代码，之后在生成后的代码基础上再做细致的业务开发，岂不是事半功倍?

- 此功能需要 JDK11

我们来到 app.js 文件内容中，默认如下：

```
load("nashorn:mozilla_compat.js");
load("./config.js");
load("./bin/server.js");
$.boot();
```

当然，按照之前章节里介绍到的配置，我们还需要事先配置好数据库的连接信息。接着，我们在将上面的代码改成以下：

```
load("nashorn:mozilla_compat.js");
load("./config.js");
load("./bin/server.js");
load("./bin/crud.js");
$.gencrud(["person"]);
```

此时，我们的代码中调用了 $.gencrud() 函数，并且传入了一个数组参数，这个数组中是你预期要实现生成代码的数据表名。完成，只需要 start.bat 或者 linux 系统下 start.sh。我们就可以在 servlet 目录下得到一个目录名为 person 的文件夹，在这个文件夹下将会产生 select.js,update.js,delete.js,save.js 四个文件。于此同时，还会在 Tropic 的根目录下生成一个 endpoints.js 的文件，这个文件中就是四个 servlet 小程序对应在 config.endpoints 的配置信息。我们只需要配置完成后，重新改回 app.js 原来的面貌，启动服务就可以正常使用了。

怎么样，如果你已经迫不及待了，不妨亲自试试吧。

CHAPTER 23

服务实例

23.1 boot()

无参数，启动服务实例

23.2 shutdown()

无参数，停止服务实例

23.3 afterBoot(fn)

- fn 服务实例启动后执行的函数

绑定一个函数，在服务实例启动后执行，使用方法：app.js 代码中加入。示范如下：

```
load("nashorn:mozilla_compat.js");
load("./config.js");
load("./bin/server.js");
//绑定启动后执行的函数
$.afterBoot(function(){
    $.logger().info("服务实例启动后，将会输出这行内容");
});
$.boot();
```

23.4 afterShutdown(fn)

- fn 服务实例启动后执行的函数

绑定一个函数，在服务实例停止后执行，使用方法：app.js 代码中加入。示范如下：

```
load("nashorn:mozilla_compat.js");
load("./config.js");
load("./bin/server.js");
//绑定启动后执行的函数
$.afterShutdown(function(){
    $.logger().info("服务实例停止后，将会输出这行内容");
});
$.boot();
```

23.5 status()

无参数，返回服务实例当前状态，以下三种

- RUNNING
- SHUTDOWN
- NOT_BOOT

23.6 bind(entry)

entry 格式和 config.js 中 endpoints 配置数组中的项格式相同，需要三个字段

- path 要提供服务的 http 路径
- servlet 提供服务的小程序文件路径
- name 提供服务的对象名

该函数一般不显式使用，配合/bin/bind.js 组件使用时由引擎自己使用

23.7 unbind(entry)

entry 格式和 config.js 中 endpoints 配置数组中的项格式相同，需要一个字段

- path 要停止提供服务的 http 路径

该函数一般不显式使用，配合/bin/bind.js 组件使用时由引擎自己使用

CHAPTER 24

数据访问

24.1 jdbc(back)

- back 提供该参数时，视为归还 JDBC Connection 实例，不提供则视为获取一个 JDBC Connection。

使用该函数默认使用了连接池技术，繁忙时，该函数将会阻塞等待，所以使用完毕请及时归还。

24.2 db()

无参数，不使用连接池创建一个 JDBC Connection 对象，一般不推荐使用。

24.3 sql()

无参数，返回一个 Apache DBUtils 组件的 QueryRunner 实例，具体使用方式可参加官方文档。

24.4 asMapList

非函数，默认提供一个 Apache DBUtils 组件的 MapListHandler 实例，负责将查询出的数据转换为一个填充了 HashMap 的 List，Map 中的 Key 为数据库列名值为数据库列的实际内容。

24.5 query(sql,params)

语法糖，支持执行 select * from x where a= ? and b= ? 的 SQL 语句，params 支持按照顺序填入，调用时将按照顺序替换 SQL 里的问号，其返回结果是经过格式化的数据集合。

24.6 format(mapList)

- mapList 为 QueryRunner 的查询结果集

该函数提供默认的日期或者时间类型字段的自动格式化。

24.7 redis(back)

- back 提供该参数时，视为归还 Jedis 实例，不提供则视为获取一个 Jedis。

使用该函数默认使用了连接池技术，繁忙时，该函数将会阻塞等待，所以使用完毕请及时归还。

24.8 mongo(db,collection)

- db 要访问的 db 名称，字符串格式
- collection 要访问的 collection 名称，字符串格式。该 collection 视为隶属于 db 参数值下。

返回一个 MongoDatabase 或者 MongoCollection 实例，只提供 db 参数时返回 MongoDatabase 实例，两者都提供时返回 MongoCollection 实例

24.9 asDoc(obj)

- obj 常规 Javascript 对象。

由于 MongoDB 要求查询条件务必是 Document 类型，所以该方法会将普通 Javascript 对象转换成为 MongoDB 所支持的 Document 类型，以便用于查询或其他数据访问交互需要。

24.10 neo4j(session)

- session 提供值 true 则表示获取 Session 不提供或者 false 则返回 Driver 实例

代码示例辅助说明:

```
public class DriverLifecycleExample implements AutoCloseable
{
private final Driver driver;

public DriverLifecycleExample( String uri, String user, String password )
{
    //默认驱动 Driver实例
    driver = GraphDatabase.driver( uri, AuthTokens.basic( user, password ) );
```

(下页继续)

(续上页)

```
}

@Override
public void close() throws Exception
{
    driver.close();
}
}

private Driver createDriver( String virtualUri, String user, String password,␣
↪ServerAddress... addresses )
{
    Config config = Config.builder()
            .withResolver( address -> new HashSet<>( Arrays.asList( addresses ) ) )
            .build();

    return GraphDatabase.driver( virtualUri, AuthTokens.basic( user, password ),␣
↪config );
}

private void addPerson( String name )
{
    String username = "neo4j";
    String password = "some password";

    try ( Driver driver = createDriver( "neo4j://x.example.com", username, password,␣
↪ServerAddress.of( "a.example.com", 7676 ),
            ServerAddress.of( "b.example.com", 8787 ), ServerAddress.of( "c.example.
↪com", 9898 ) ) )
    {
        //使用会话 Session
        try ( Session session = driver.session( builder().withDefaultAccessMode(␣
↪AccessMode.WRITE ).build() ) )
        {
            session.run( "CREATE (a:Person {name: $name})", parameters( "name", name␣
↪) );
        }
    }
}
```

该函数获取 neo4j 的 Driver 实例或者 Session 实例。上文代码中演示了二者的使用方式。

CHAPTER 25

HTTP-Client

25.1 get(url,headers,asJson)

- url 要请求的目标 url 地址，字符串格式
- headers 要携带的 http headers，对象格式，key 为 header-name，value 为 header-value。
- asJson 响应结果是否要 JSON 对象化处理，默认为 false，请根据实际情况自行设置。

使用 get 方式请求一个指定的 http 地址，并返回响应结果，该函数的超时配置在 config.js 中设置。

25.2 post(url,headers,data,asJson)

- url 要请求的目标 url 地址，字符串格式
- headers 要携带的 http headers，对象格式，key 为 header-name，value 为 header-value。
- data 要提交的 http-body 数据，支持对象格式
- asJson 响应结果是否要 JSON 对象化处理，默认为 false，请根据实际情况自行设置。

使用 post 方式请求一个指定的 http 地址，并返回响应结果，该函数的超时配置在 config.js 中设置。

CHAPTER 26

其他工具函数

26.1 empty(str)

- str 要检查的字符串

检查一个字符串是不是长度位 0，不对 null 值做处理。

26.2 at(index,string|array)

- index 下标
- string|array 目标字符串或者数组

返回指定下标的内容，如果第二个参数是字符串则返回字符串指定 index 下标的内容，如果是数组则返回指定位置的数组项。

26.3 each(object|array,fn[i,n])

- object|array 一个 Javascript 对象或者数组
- fn[i,n] 一个带有两个参数的函数，i 表示下标，n 表示当前项。

该函数为 for(var i in obj) 的语法糖，像 Jquery $.each 致敬。

26.4 servlet(name,fn[req,resp])

- name 小程序的名字
- fn[req,resp] 小程序的服务提供函数，定义两个形参，一个是 request 一个是 response。

语法糖，等价于定义一个包含了 service 函数的小程序对象。

26.5 redirect(resp,url)

- resp http 响应对象
- url 指定要重定向到的 url 地址，支持绝对路径和相对路径。

重定向函数，将请求指向另一个 http-url 地址

26.6 filter(name,fn[req,resp])

- name 小程序的名字
- fn[req,resp] 小程序的服务提供函数，定义两个形参，一个是 request 一个是 response。

语法糖，等价于定义一个包含了 service 函数的小程序对象。

26.7 toJson(obj)

- obj 指定要 JSON 格式化的对象，支持数组和对象

将传入参数格式化为 JSONString。

26.8 fromJson(jsonStr)

- jsonStr 指定要反序列化的 JSON 字符串，支持数组和对象

将传入参数反序列化为对象或数组。

26.9 setTimeout(fn,time)

- fn 无参函数
- time 延迟的时间，单位：毫秒

```
$.setTimeout(function(){
    println("Hello ,i'm in 'Timeout'");
},2000);
```

指定一个时间，延迟执行一个函数，和浏览器中 window.setTimeout 等价。

26.10 setInterval(fn,time)

- fn 无参函数
- time 间隔的时间，单位：毫秒

```
$.setInterval(function(){
    println("Hello ,i'm in 'Interval'");
},2000);
```

指定一个时间，以该时间为固定间隔执行一个函数，和浏览器中 window.setInterval 等价。

CHAPTER 27

日志

27.1 logger(name)

- name 日志对象的名字，不传入时默认为:default

获取一个日志对象，用于记录日志

27.2 debug(str)

语法糖，等价于 $.logger().debug(str);

27.3 info(str)

语法糖，等价于 $.logger().info(str);

27.4 warn(str)

语法糖，等价于 $.logger().warn(str);

27.5 error(str)

语法糖，等价于 $.logger().error(str);

CHAPTER 28

秘钥证书

28.1 genkey(cfg)

生成秘钥证书，建议配合组件/bin/keytool.js 使用。如果手动调用使用，请按照参数格式要求。cfg 为对象格式，包含以下属性字段

- alias 秘钥别名
- keypasswd 秘钥密码
- alg 加密算法
- keysize 长度
- expire 有效期
- keystorename 秘钥库文件名
- keystorepass 秘钥库密码
- cnname CN 名字与姓氏
- ouname OU 组织单位名称
- oname O 组织名称
- lname L 城市或区域名称
- stname ST 州或省份名称
- cname C 单位的两字母国家代码

CHAPTER 29

代码生成

29.1 gencrud(tables,asCamel)

- tables 表名数组，数组格式
- asCamel 列名是否 Camel 化，当列名中有 _ 时，此参数传 true 将自动去掉 _ 并将其后第一个字母大写。

生成 crud 模板化代码，示例:

```
load("nashorn:mozilla_compat.js");
load("./config.js");
load("./bin/server.js");
load("./bin/crud.js");
$.gencrud(["person","city"],true);
```

执行完毕后，即可在/servlet 目录下找到对应代码，并会产生一个 endpoints.js 的文件，其中包含了所有的默认配置。

CHAPTER 30

作者

余幼时即嗜学。家贫无从致书以观，每假借于藏书之家，手自笔录，计日以还。天大寒，砚冰坚，手指不可屈伸，弗之怠。录毕，走送之，不敢稍逾约。以是人多以书假余，余因得遍观群书。既加冠，益慕圣贤之道。又患无硕师名人与游，尝趋百里外，从乡之先达执经叩问。先达德隆望尊，门人弟子填室，未尝稍降辞色。余立侍左右，援疑质理，俯身倾耳以请；或遇其叱咄，色愈恭，礼愈至，不敢出一言以复；俟其欣悦，则又请焉。故余虽愚，卒获有所闻。

当余之从师也，负箧曳屣行深山巨谷中，穷冬烈风，大雪深数尺，足肤皲裂而不知。至舍，四肢僵劲不能动，媵人持汤沃灌，以衾拥覆，久而乃和。寓逆旅，主人日再食，无鲜肥滋味之享。同舍生皆被绮绣，戴朱缨宝饰之帽，腰白玉之环，左佩刀，右备容臭，烨然若神人；余则🄵袍敝衣处其间，略无慕艳意，以中有足乐者，不知口体之奉不若人也。盖余之勤且艰若此。

今虽耄老，未有所成，犹幸预君子之列，而承天子之宠光，缀公卿之后，日侍坐备顾问，四海亦谬称其氏名，况才之过于余者乎？

今诸生学于太学，县官日有廪稍之供，父母岁有裘葛之遗，无冻馁之患矣；坐大厦之下而诵《诗》《书》，无奔走之劳矣；有司业、博士为之师，未有问而不告，求而不得者也；凡所宜有之书，皆集于此，不必若余之手录，假诸人而后见也。其业有不精，德有不成者，非天质之卑，则心不若余之专耳，岂他人之过哉！

东阳马生君则，在太学已二年，流辈甚称其贤。余朝京师，生以乡人子谒余，撰长书以为贽，辞甚畅达，与之论辩，言和而色夷。自谓少时用心于学甚劳，是可谓善学者矣！其将归见其亲也，余故道为学之难以告之。谓余勉乡人以学者，余之志也；诋我夸际遇之盛而骄乡人者，岂知余者哉！

----宋濂《苦学》

- JackLee letui@qq.com

CHAPTER 31

后记

在这里，我将开诚布公的公开些我的想法和 Tropic 面临的困扰。

31.1 痛点

脱离了 Java 语言的 IDE 工具，我们在用 Tropic 开发程序时将面临没有那么强大语法提示，特别是涉及到 Java 第三方工具包的对象函数时。这是我深思的问题之一，难道我为了让框架更容易使用，我需要开发一套更强大的 IDE 工具吗？

CHAPTER 32

素书

- 黄石公

32.1 第一章原章

- 【原文】

夫道、德、仁、义、礼五者，一体也。道者，人之所蹈，使万物不知其所由。德者，人之所使，使万物各得其所欲。仁者，人之所亲，有慈慧恻隐之心，以遂其生成。义者，人之所宜，赏善罚恶，以立功立事。礼者，人之所履，夙兴夜寐，以成人伦之序。夫欲为人之本，不可无一焉。贤人君子，明于盛衰之道，通乎成败之数，审乎治乱之势，达乎去就之理。故潜居抱道，以待其时。若时至而行，则能极人臣之位；得机而动，则能成绝代之功。如其不遇，没身而已。是以其道足高，而名重于后代。

- 【译文】

道、德、仁、义、礼五者，本为一体，不可分离。

道，是一种自然规律，人人都在遵循著自然规律，自已却意识不到这一点，自然界万事万物亦是如此。德、即是获得，依德而行，可使一已的欲求得到满足，自然界万事万物也是如此。仁、是人所独具的仁慈、爱人的心理，人能关心、同情人，各种善良的愿望和行动就会产生。义、是人所认为符合某种道德观念的行为，人们根据义的原则奖善惩恶，以建立功业。礼、是规定社会行为的法则，规范仪式的总称。人人必须遵循礼的规范，兢兢业业，夙兴夜寐，按照君臣、父子、夫妻、兄弟等人伦关系所排列的顺序行事。这五个条目是做人的根本，缺一不可的。贤明能干的人物，品德高尚的君子，都能看清国家兴盛、衰弱、存亡的道理，通晓事业成败的规律，明白社会政治修明与纷乱的形势，懂得隐退仕进的原则。因此，当条件不适宜之时，都能默守正道，甘于隐伏，等待时机的到来。一旦时机到来而有所行动，常能建功立业位极人臣。如果所遇非时，也不过是淡泊以终而已。也就因此，像这样的人物常能树立极为崇高的典范，名重于后世呵！

32.2 第二章正道

- 【原文】

德足以怀远，信足以一异，义足以得众，才足以鉴古，明足以照下，此人之俊也；

行足以为仪表，智足以决嫌疑，信可以使守约，廉可以使分财，此人之豪也；

守职而不废，处义而不回，见嫌而不□免，见利而不□得，此人之杰也。

- 【译文】

品德高尚，则可使远方之人前来归顺。诚实不欺，可以统一不同的意见。道理充分可以得到部下群众的拥戴。才识杰出，可以借鉴历史。聪明睿智可以知众而容众。这样的人，可以称他为人中之俊。行为端正，可以为人表率。足智多谋，可以解决疑难问题。天无信，四时失序，人无信，行止不立。如果能忠诚守信，这是立身成名之本。君子寡言，言而有信，一言议定，再不肯改议、失约。是故讲究信用，可以守约而无悔。廉洁公正，且疏财仗义。这样的人，可以称他为人中之豪。见嫌而不苟免，克尽职守，而无所废弛；恪守信义，而不稍加改变；受到嫌疑，而能居义而不反顾；利字当头，懂得不悖理苟得。这样的人，可以称为人中之杰。

32.3 第三章求人之志

- 【原文】

绝嗜禁欲，所以除累。抑非损恶，所以让过。贬酒阙色，所以无污。

避嫌远疑，所以不误。博学切问，所以广知。高行微言，所以修身。

恭俭谦约，所以自守。深计远虑，所以不穷。亲仁友直，所以扶颠。

近恕笃行，所以接人。任材使能，所以济物。殚恶斥谗，所以止乱。

推古验今，所以不惑。先揆后度，所以应卒。设变致权，所以解结。

括囊顺会，所以无咎。橛橛梗梗，所以立功。孜孜淑淑，所以保终。

- 【译文】

杜绝不良的嗜好，禁止非分的欲望，这样可以免除各种牵累；抑制不合理的行为，减少邪恶的行径，这样可以避免过失；谢绝酒色侵扰，这样可以不受玷污；回避嫌疑，远离惑乱，这样可以不出错误。广泛地学习，仔细地提出各种问题，这样可以丰富自己的知识；行为高尚，辞锋不露，这样可以修养身心、陶冶性情；肃敬、节俭、谦逊、简约，这样可以守身不辱；深谋远虑，这样可以不至于困危；亲近仁义之士，结交正直之人，这样可以在逆境中得到帮助。为人尽量宽容，行为敦厚，这是待人处世之道。任才使能，使人人能尽其才，这是用人成事之要领；抑制邪恶，斥退谗佞之徒，这样可以防止动乱；推求往古，验证当今，这样可以不受迷惑；了解事态，心中有数，这样可以应付仓卒事变；采用灵活手法，施展权变之术，这样可以解开纠结；心中有数，闭口不言，凡事能顺从时机，这样可以远怨无咎；坚定不移，正直刚强，这样才能建功立业；勤勉惕励；心地善良，这样才能善始善终。

32.4 第四章本德宗道

- 【原文】

夫志心笃行之术。长莫长于博谋，安莫安于忍辱，先莫先于修德，乐莫乐于好善，神莫神于至诚，明莫明于体物，吉莫吉于知足，苦莫苦于多愿，悲莫悲于精散，病莫病于无常，短莫短于苟得，幽莫幽于贪鄙，孤莫孤于自恃，危莫危于任疑，败莫败于多私。

- 【译文】

欲始志向坚定，笃实力行：最好的方法，莫过于深思多谋；最安全的方式，莫过于安于忍辱；最优先的要务，莫过于进德修业；最快乐的态度，莫过于乐于好善；最神奇的效验，莫过于用心至诚；最高明的做法，莫过于明察秋毫；最吉祥的想法，莫过于安分知足；最痛苦的缺点，莫过于欲求太多；最悲哀的情形，莫过于心神离散；最麻烦的病态，莫过于反覆无常；最无聊的妄念，莫过于不劳而获；最愚昧的观念，莫过于贪婪卑鄙；最孤独的念头，莫过于目空一切；最危险的举措，莫过于任人而疑；最失败的行径；莫过于自私自利；

32.5 第五章道义

- 【原文】

以明示下者暗，有过不知者蔽，迷而不返者惑，以言取怨者祸，令与心乖者废，后令缪前者毁，怒而无威者犯，好众辱人者殃，戮辱所任者危，慢其所敬者凶，貌合心离者孤，亲谗远忠者亡，近色远贤者昏，女谒公行者乱，私人以官者浮，凌下取胜者侵，名不胜实者耗。略己而责人者不治，自厚而薄人者弃废。以过弃功者损，群下外异者沦，既用不任者疏，行赏吝色者沮，多许少与者怨，既迎而拒者乖。薄施厚望者不报，贵而忘贱者不久。念旧而弃新功者凶，用人不得正者殆，强用人者不畜，为人择官者乱，失其所强者弱，决策于不仁者险，阴计外泄者败，厚敛薄施者凋。战士贫，游士富者衰；货赂公行者昧；闻善忽略，记过不忘者暴；所任不可信，所信不可任者浊。牧人以德者集，绳人以刑者散。小功不赏，则大功不立；小怨不赦，则大怨必生。赏不服人，罚不甘心者叛。赏及无功，罚及无罪者酷。听谗而美，闻谏而仇者亡。能有其有者安，贪人之有者残。

- 【译文】

在部下面前显示高明，一定会遭到愚弄。有过错而不能自知，一定会受到蒙蔽。走入迷途而不知返回正道，一定是神志惑乱。因为语言招致怨恨，一定会有祸患。思想与政令矛盾，一定会坏事。政令前后不一，一定会失败。发怒却无人畏惧，一定会受到侵犯。喜欢当众侮辱别人，一定会有灾难。对手下的大将罚之过当，一定会有危险。怠慢应受尊重的人，一定会招致不幸。表面上关系密切，实际上心怀异志的，一定会陷于孤独。亲近谗慝，远离忠良，一定会灭亡。亲近女色，疏远贤人，必是昏🄵目盲。女子干涉大政，一定会有动乱。随便将官职到处乱送，政治就会出现乱相。欺凌下属而获得胜利的，自己也一定会受到下属的侵犯。所享受的名声超过自己的实际才能，即使耗尽精力也治理不好事务。对自己马虎，对别人求全责备的，无法处理事务。对自己宽厚，对别人刻薄的，一定被众人遗弃。因为小过失便取消别人的功劳的，一定会大失人心。部下纷纷有离异之心，必定沦亡。既然用了人却不给予信任，必定导致关系疏远。论功行赏时吝啬小气，形于颜色，必定使人感到沮丧。承诺多，兑现少，必招致怨恨。起初竭诚欢迎，末了又拒于门外，一定会恩断义绝。给予别人很少，却希望得到厚报的，一定会大失所望。富贵之后就忘却贫贱时候的情状，一定不会长久。念及别人旧恶，忘记其所立新功的，一定遭来大凶。任用邪恶之徒，一定会有危险。勉强用人，一定留不住人。用人无法摆脱人情纠结，政事必越理越乱。失去自己的优势，力量必然削弱。处理问题、制定决策时向不仁之人问计，必有危险。秘密的计划泄露出去，一定会失败。横征暴敛、薄施寡恩，一定会衰落。奋勇征战的将士生活贫穷，鼓舌摇唇的游士安享富贵，国势一定会衰落。贿赂政府官员的事到处可见，政治必定十分昏暗。知道别人的优点长处却不重视，对别人的缺点错误反而耿耿于怀的，则是作风粗暴。使用的人不堪信任，信任的人又不能胜任其职，这样的政治一定很混浊。依靠道德的力量来治理人民，人民就会团结；若一味地依靠刑法来维持统治，则人民将离散而去。小的功劳不奖赏，便不会建立大功劳；小的怨恨不宽赦，大的怨恨便会产生。奖赏不能服人，处罚不能让人甘心，必定引起叛乱；赏及无功之人，罚及无罪之人，就是所谓的残酷。听到谗佞之言就十分高兴，听到忠谏之言便心生怨恨，一定灭亡。藏富于民，以百姓的富有作为本身的富有，这样才会安定；欲壑难填，总是贪求别人所有的，必然残民以逞。

32.6 第六章安礼

- 【原文】

怨在不舍小过，患在不预定谋。福在积善，祸在积恶。饥在贱农，寒在堕织。安在得人，危在失事。富在迎来，贫在弃时。上无常操，下多疑心。轻上生罪，侮下无亲。近臣不重，远臣轻之。自疑不信人，自信不疑人。枉士无正友，曲上无直下。危国无贤人，乱政无善人。爱人深者求贤急，乐得贤者养人厚。国将霸者士皆归，邦将亡者贤先避。地薄者大物不产，水浅者大鱼不游，树秃者大禽不栖，林疏者大兽不居。山峭者崩，泽满者溢。弃玉取石者盲，羊质虎皮者柔。衣不举领者倒，走不视地者颠。柱弱者屋坏，辅弱者国倾。足寒伤心，人怨伤国。山将崩者下先隳，国将衰者人先弊。根枯枝朽，人困国残。与覆车同轨者倾，与亡国同事者灭。见已生者慎将生，恶其迹者须避之。畏危者安，畏亡者存。夫人之所行，有道则吉，无道则凶。吉者，百福所归；凶者，百祸所攻。非其神圣，自然所钟。务善策者无恶事，无远虑者有近忧。同志相得，同仁相忧，同恶相党，同爱相求，同美相妒，同智相谋，同贵相害，同利相忌，同声相应，同气相感，同类相依，同义相亲，同难相济，同道相成，同艺相规，同巧相胜：此乃数之所得，不可与理违。释己而教人者逆，正己而化人者顺。逆者难从，顺者易行，难从则乱，易行则理。如此理身、理国、理家，可也！

- 【译文】

怨恨产生于不肯赦免小的过失；祸患产生于事前未作仔细的谋画；幸福在于积善累德；灾难在于多行不义。轻视农业，必招致饥馑；惰于蚕桑，必挨冷受冻。得人必安，失士则危。招来远客即富，荒废农时则贫。上位者反覆无常，言行不一，部属必生猜疑之心，以求自保。对上官轻视怠慢，必定获罪；对下属侮辱傲慢，必定失去亲附。近幸左右之臣不受尊重，关系疏远之臣必不安其位。自己怀疑自己，则不会信任别人；自己相信自己，则不会怀疑别人。邪恶之士决无正直的朋友；邪僻的上司必没有公正刚直的部下。行将灭亡的国家，决不会有贤人辅政；陷于混乱的政治，决不会有善人参与。爱人深者，一定急于求贤才，乐得于贤才者，待人一定丰厚。国家即将称霸，人才都会聚集来归；邦国即将败亡，贤者先行隐避。土地贫瘠，大物不产；水浅之处，大鱼不游；秃树之上，大禽不栖；疏林之中，大兽不居。山势过于陡峭，则容易崩塌；沼泽蓄水过满，则会漫溢出来。弃玉抱石者目光如盲，羊质虎皮者虚于矫饰。拿衣服时不提领子，势必把衣服拿倒。走路不看地面的一定会跌倒。

房屋梁柱软弱，屋子会倒塌；才力不足的人掌政，国家会倾覆。脚下受寒，心肺受损；人心怀恨，国家受伤。大山将要崩塌，土质会先毁坏；国家将要衰亡，人民先受损害。树根干枯，枝条就会腐朽；人民困窘，国家将受伤害。与倾覆的车子走同一轨道的车，也会倾覆；与灭亡的国家做相同的事，也会灭亡。见到已发生的事情，应警惕还将发生类似的事情；预见险恶的人事，应事先回避。害怕危险，常能得安全；害怕灭亡，反而能生存。人的所作所为，符合行事之道则吉，不符合行事之道则凶。吉祥的人，各种各样的好处都到他那里；不吉祥的人，各种各样的恶运灾祸都向他袭来。这并不是什么奥妙的事，而是自然之理。务善策者无恶事，无远虑者有近忧。同志相得，同仁同忧，同恶相党，同爱同求，同美相妒，同智相谋，同贵相害，同利相忌。同声相应，同气相感，同类相似，同义相亲，同难相济。同道相成，同艺相窥，同巧相胜。以上这些都是自然而然的道理，凡人类有所举措，均应遵守这些规律，不可与理相抗。把自己放在一边，单纯去教育别人，别人就不接受他的大道理；如果严格要求自己，进而去感化别人，别人就会顺服。违反常理，部属则难以顺从；合乎常理，则办事容易。部属难以顺从，则容易产生动乱；办事容易，则能得到畅通的治理。

以上所述的各项事理，用在修身、持家、治国，均会获得丰硕的效果。

CHAPTER 33

止学

- 文中子

33.1 智卷

智极则愚也。圣人不患智寡，患德之有失焉。才高非智，智者弗显也。位尊实危，智者不就也。大智知止，小智惟谋，智有穷而道无尽哉。谋人者成于智，亦丧于智也。谋身者恃其智，亦舍其智。智有所缺，深存其敌，慎之少祸焉。智不及而谋大者毁，智无歇而谋远者逆。智者言智，愚者言愚，以愚饰智，以智止智，智也。

33.2 用势卷

势无常也，仁者勿恃。势伏凶也，智者不矜。势莫加君子，德休于小人。君子势不于力也，力尽而势亡焉。小人势不惠人也，趋之必祸焉。众成其势，一人堪毁。强者凌弱，人怨乃弃。势极无让者疑，位尊弗恭者忌。势或失之，名或谤之，少怨者再得也。势固灭之，人固死之，无骄者惠嗣焉。

33.3 利卷

惑人者无逾利也，利无求弗获，德无施不积。众逐利而富寡，贤让功而名高。利大伤身，利小惠人，择之宜慎也。天贵于时，人贵于明，动之有戒也。众见其利者，非利也。众见其害者，或利也。君子重义轻利，小人嗜利远信，利御小人而莫御君子矣。利无尽处，命有尽时，不怠可焉。利无独据，运有兴衰，存畏警焉。

33.4 辩卷

物朴乃存，器工招损。言拙意隐，辞尽锋出。识不逾人者，莫言断也。势不及人者，休言讳也。力不胜人者，勿言强也。王者不辩，辩则少威焉。智者讷言，讷则惑敌焉。勇者无语，语则怯行焉。

33.5 誉卷

忠臣不表其功，窃功者必奸也。君子堪隐，人恶，谤贤者固小人矣。誉存其伪，谄者以誉欺人。名不由己，明者言不自赞。贪巧之功，天不佑也。赏誉勿轻，轻则誉贱，贱则无功也。受誉知辞，辞则德显，显则释疑也。上下无争，誉之不废焉。人无誉堪存，誉非正当灭。求誉不得，或为福也。

33.6 情卷

情滥无行，欲多失矩。其色如一，神鬼莫测。上无度失威，下无忍莫立。上下知离，其位自安。君臣殊密，其臣反殃。小人之荣，情不可攀也。情存疏也，近不过己，智者无痴焉。情难追也，逝者不返，明者无悔焉。多情者多艰，寡情者少难。情之不敛，运无幸耳。

33.7 蹇卷

人困乃正，命顺乃奇。以正化奇，止为枢也。事变非智勿晓，事本非止勿存。天灾示警，逆之必亡；人祸告诫，省之固益。燥生百端，困出妄念，非止莫阻害之蔓焉。视己勿重者重，视人为轻者轻。患以心生，以蹇 (jian) 为乐，蹇不为蹇矣。穷不言富，贱不趋贵。忍辱为大，不怒为尊。蹇非敌也，敌乃乱焉。

33.8 释怨

世之不公，人怨难止。穷富为仇，弥祸不消。君子不念旧恶，旧恶害德也。小人存隙必报，必报自毁也。和而弗争，谋之首也。名不正而谤兴，正名者必自屈焉。惑不解而恨重，释惑者固自罪焉。私念不生，仇怨无结焉。宽不足以悦人，严堪补也。敬无助于劝善，诤堪教矣。

33.9 心卷

欲无止也，其心堪制。惑无尽也，其行乃解。不求于人，其尊弗伤。无嗜之病，其身靡失。自弃者人莫救也。苦乐无形，成于心焉。荣辱存异，贤者同焉。事之未济，志之非达，心无怨而忧患弗加矣。仁者好礼，不欺其心也。智者示愚，不显其心哉。

33.10 修身卷

服人者德也。德之不修，其才必曲，其人非善矣。纳言无失，不辍亡废。小处容疵，大节堪毁。敬人敬，德之厚也。诚非虚致，君子不行诡道。祸由己生，小人难于胜己。谤言无惧，强者不纵，堪验其德焉。不察其德，非识人也。识而勿用，非大德矣。