车联网接口自动化总结报告

一、背景

之所以要对车联网Web进行接口自动化，因为如今的系统复杂度不断上升，系统不仅有传统的新增删改等操作，还增加了大量的条件组合查询、导入查询、批量新增、批量更新批量导入以及大数据模块的导出，还有离线导出，这些功能遍布每个模块的每个页面，使得传统的手工测试方法急剧增加难度，而且由于功能繁琐使得测试效率大幅下降的同时，也增加了漏测或错测的风险。

而接口自动化测试是相对容易实现自动化持续集成，和UI自动化测试相比来说，运行不仅稳定，其运行效率也远比UI自动化要快，从而能减少人力回归测试的成本和时间，缩短测试周期，支持后端实现快速迭代更新发版的需求。

二、框架搭建

基于Jmeter+Ant+Git+Jenkins搭建的接口自动化测试框架，基于java语言，以数据库连接+csv表格获取测试数据来驱动测试，其中是以数据库连接为主，csv为辅。

Jmeter：是Apache组织开发的基于Java的压力测试工具。

Ant:是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具，大多用于Java环境中的软件开发。

Git:是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。

Jenkins:是一个开源软件项目，是基于Java开发的一种持续集成工具，用于监控持续重复的工作。

总体而言，就是利用Jmeter编写接口的脚本组成脚本库，这个过程是需要用到java语言和mysql数据库知识，然后把脚本库上传到Git管理，再利用Jenkins引用git的脚本库，并使用ant插件来构建、运行测试，最后发送测试报告。

三、解决问题

1. 完成车联网系统整体功能模块的98%的脚本开发，覆盖了成功测试，失败测试，异常测试以及字段校验测试。

2. 设计并实现通用的、标准化的接口脚本，阅读性好，维护性强。另外通过对jmeter的相关组件进行二次开发，使得脚本不受请求参数、数据库信息，Web地址，以及系统环境的约束，具备很好的灵活性，可不费多少人力，就可轻松的移植到其他环境使用。

3.编写了一套测试数据生成方法，一来解决了项目初始之时缺乏测试数据来驱动接口测试的难题，二来使得脚本不再依赖外部数据，只需引用自身生成的数据就可驱动测试，而且生成测试数据辨识度高，能与其他正式数据区分使用。

四、关键技术

1、脚本规范化；

主要是接口脚本名、单个脚本设计、接口公共区、接口验证区、Http请求名、Http请求参数、正则表达式提取、响应断言、beanshell使用等都制定标准。

2、对常用的请求参数进行封装类后进行调用；

主要是对常用的数据进行封装，诸如各种不同格式的时间数据、身份证号、电话、邮箱、地址、姓名、用户名、账户名、32位uuid、vin码、终端内部编号、车载终端型号、车辆型号、随机数、iccid等等，通过封装后调用可达到全局调用，且辨识度高；

3、开发PathUtil和 InitProperties辨识系统，读取配置文件；

接口脚本存在linux服务器环境和本地window环境，在linux是上存贮运行是为了持续部署、集成以及监控使用，而在本地window则是更方便于调试和维护，所以脚本设计里不能写死所要读取的配置信息，我们要确保脚本是可移植，保证无论存在哪个环境都能正常运行。所以PathUtil和 InitProperties就是这样而来，PathUtil是区别系统调用脚本所在系统的配置文本路径，InitProperties则是读取配置文本路径的信息。

4、开发DbUtil来取代Jmeter的JDBC Connection Configuration（数据库连接器）

接口的框架主要是以数据库的数据来驱动测试，所以会用到Jmeter的JDBC Connection Configuration，并由它来配合JDBC request对数据进行查增删改操作，但Jmeter的JDBC Connection Configuration将数据库信息写死，而且无法通过外部配置文件的读取来进行参数化，一旦车联网换个平台，所相对应的数据库信息在脚本里就失效，这时候所有的接口脚本都需要修改数据库信息来保证在新的平台可用，这样无疑是大大的增加了移植的人工成本。

所以我们另外开发了DbUtil来取代JDBC Connection Configuration，将数据库连接器写在脚本外部，脚本只需引用此jar包来读取数据库信息以此连接数据库，并通过Jmeter自带的BeanShell来编写查增更改语句来对数据进行操作；

5、开发FileUtils来对导出功能进行封装

Web的导出功能是很常见，基本遍布于每个页面，但Jmeter本身是不支持和浏览器一样的导出，接口只是返回的数据。所以我们需要开发FileUtils来读取接口响应的数据，通过代码进行解析，并写入文本，最后导出在指定目录。

6、引入Apache POI来对导入功能进行开发

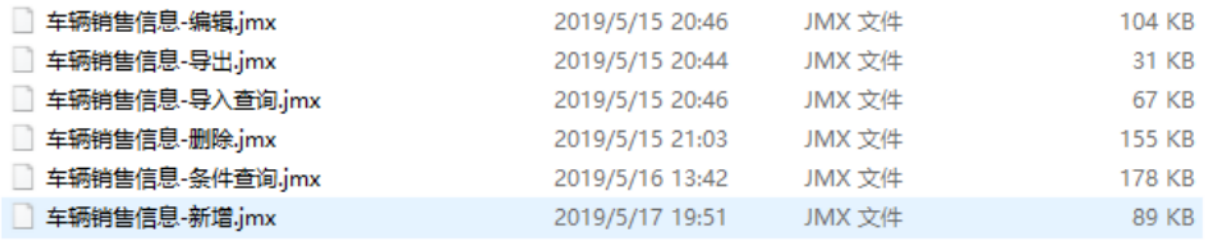
同样Web的导入功能也是很常见，一般进行导入操作，只需要配置好相关导入文件即可，但一旦这样做，就意味着文件里的数据将是写死的数据，缺乏灵活性，且不符合可复用可移植的这个前提，所以我们需要引入Apache POI包在脚本里对导入文件进行开发和生成，这样保持了脚本的灵活，而且不受外部因素影响。

五、脚本规范化

这里把脚本规范化拿出来单独总结，是因为脚本规范的重要性，如果不规范化，任由测试人员的意志去写脚本，那么后期的团队维护将会变得很困难，脚本不是一个人写完了就是完事了。我们要考虑的是长远的维护计划。脚本越规范，阅读性就越好，那么无论是谁来维护，自然就会变得很容易，先前的测试计划里已经有要求过，但是很遗憾有些脚本仍然不够规范，而且暴露了很多问题，这边我从整体出发，我重新整理一遍规范。而以下的规范可以

总结其要遵从的原则，包括有连续性，独立性，完整性，可重用性，可维护性以及逻辑分块；

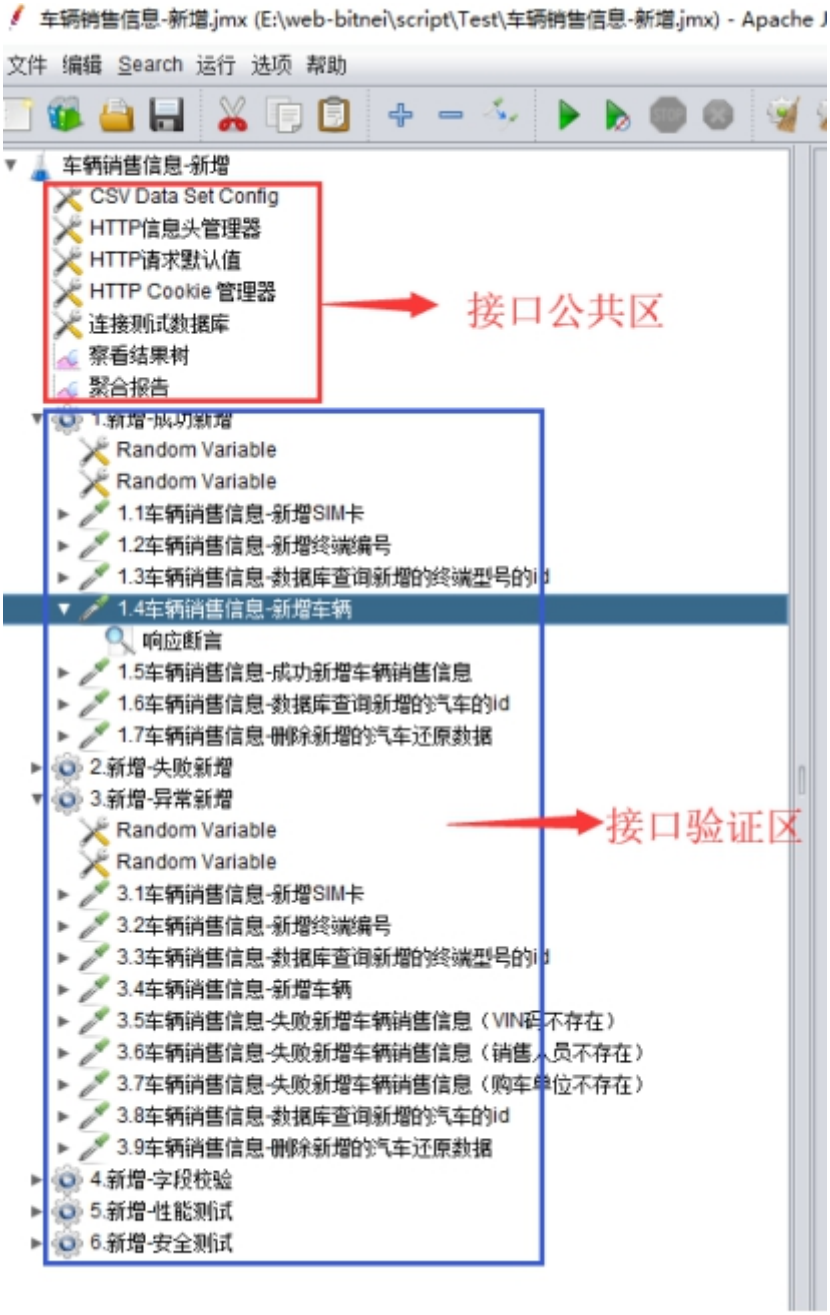
1、接口脚本名规范：【模块名-功能】，一个功能建立一个测试计划也就是jmx文件，这一个体现了可维护性；



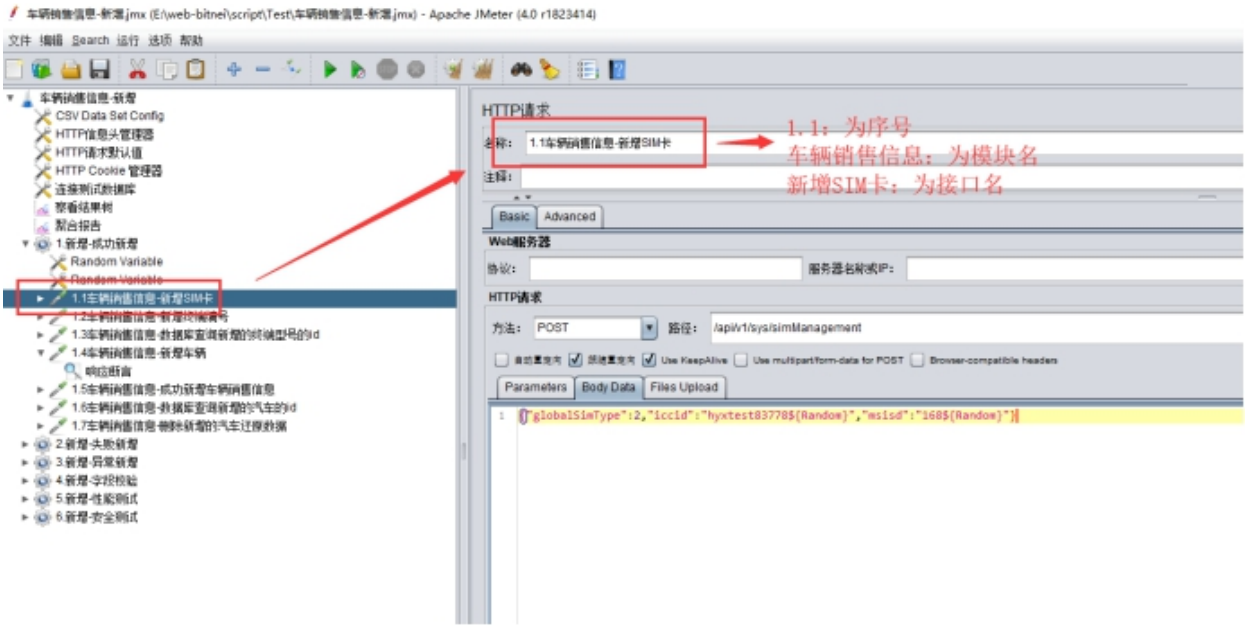
2、单个脚本设计规范：接口公共区+接口验证区，这一个体现了可维护性，独立性，独立性就是表示上1个用例的执行结果不会对下一个用例的执行产生影响；

接口公共区：为CSV数据文件配置、信息头管理器、HTTP请求默认值、HTTP Cookie管理器、数据库连接器，察看结果数和聚合报告，这公共区是整个接口自动化框架，可以全局通用。

接口验证区：则是针对每个模块的功能，按照设计用例进行划分，以【车辆销售信息-新增】这个测试计划jmt文件，它的接口功能验证区，会分成【1.新增-成功新增】、【2.新增-失败新增】【3.新增-异常新增】、【4.新增-字段校验】、【5.新增-性能测试】和【6.新增-安全测试】这6个线程组，而线程组里面就是Http请求、前置器、Jdbc request、数据提取、数据关联、后置器以及响应断言，组成这个接口验证区；



3、Http请求名规范:序号+模块名+接口名，这一个体现了可维护性。



4、引用参数的名称名规范，这一个体现了可维护性和可重用性，引用参数名称规范可以提供给其他用例复用或调用。

由于查询的id/name等在其他表都是由相似的名，所以我们这以表名_id，表名_name来定义引用的参数名，如需查询同一个表的多个id，name等，则表名_id，表名_name后面增加阿拉伯数字区分，比如表名_id1，表名_id2，表名_id3；

BeanShell PreProcessor

名称： beanshell查询供应商的id

注释：

Reset bsh.Interpreter before each call

Reset Interpreter: False

Parameters to be passed to BeanShell (=> String Parameters and String []bsh.args)

Parameters:

Script file (overrides script)

File Name:

Script (variables: ctx vars props prev sampler log)

Script:

```
import java.sql.*;
import com.bitnei.util.*;


        String sql=" SELECT id FROM sys_unit_type WHERE NAME='供应商' ";

        Connection conn=null;
        Statement stmt=null;
        ResultSet rs = null ;
        conn=DbUtil.getCon();
        stmt=conn.createStatement();
        rs = stmt.executeQuery(sql) ;

        while(rs.next()){
             String id=rs.getString("id");
             vars.put("sys_unit_type_id",id);
        }

         DbUtil.closeCon(conn);
```

BeanShell PostProcessor

名称： beanshell查询新增供应商的id

注释：

Reset bsh.Interpreter before each call

Reset Interpreter: False

Parameters to be passed to BeanShell (=> String Parameters and String []bsh.args)

Parameters:

Script file (overrides script)

File Name:

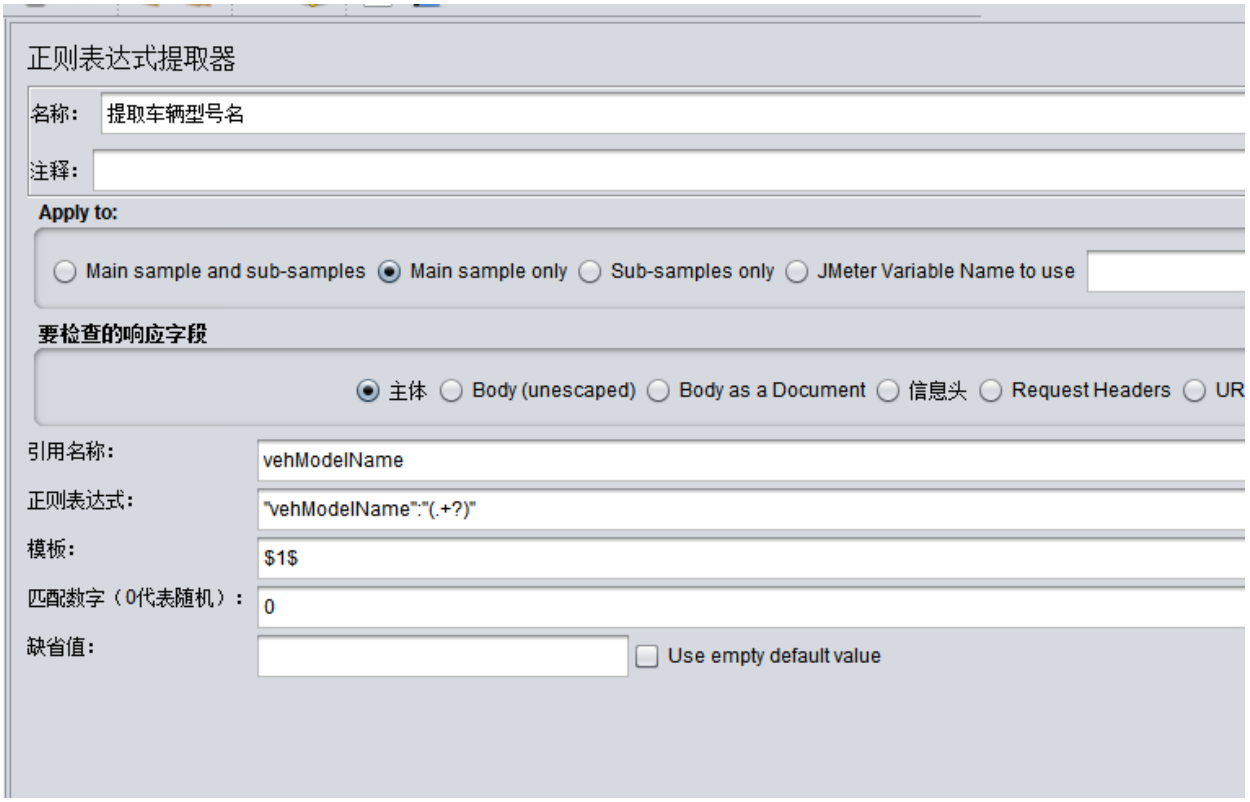
Script (variables: ctx vars props prev data log)

Script:

```
import java.sql.*;
import com.bitnei.util.*;


        String sql=" SELECT id FROM sys_unit where name='bitnei供应商${Random}' ORDER BY create_time DESC LIMIT 1 ";

        Connection conn=null;
        Statement stmt=null;
        ResultSet rs = null ;
        conn=DbUtil.getCon();
        stmt=conn.createStatement();
        rs = stmt.executeQuery(sql) ;

        while(rs.next()){
             String id=rs.getString("id");
             vars.put("sys_unit_id",id);
        }

         DbUtil.closeCon(conn);
```

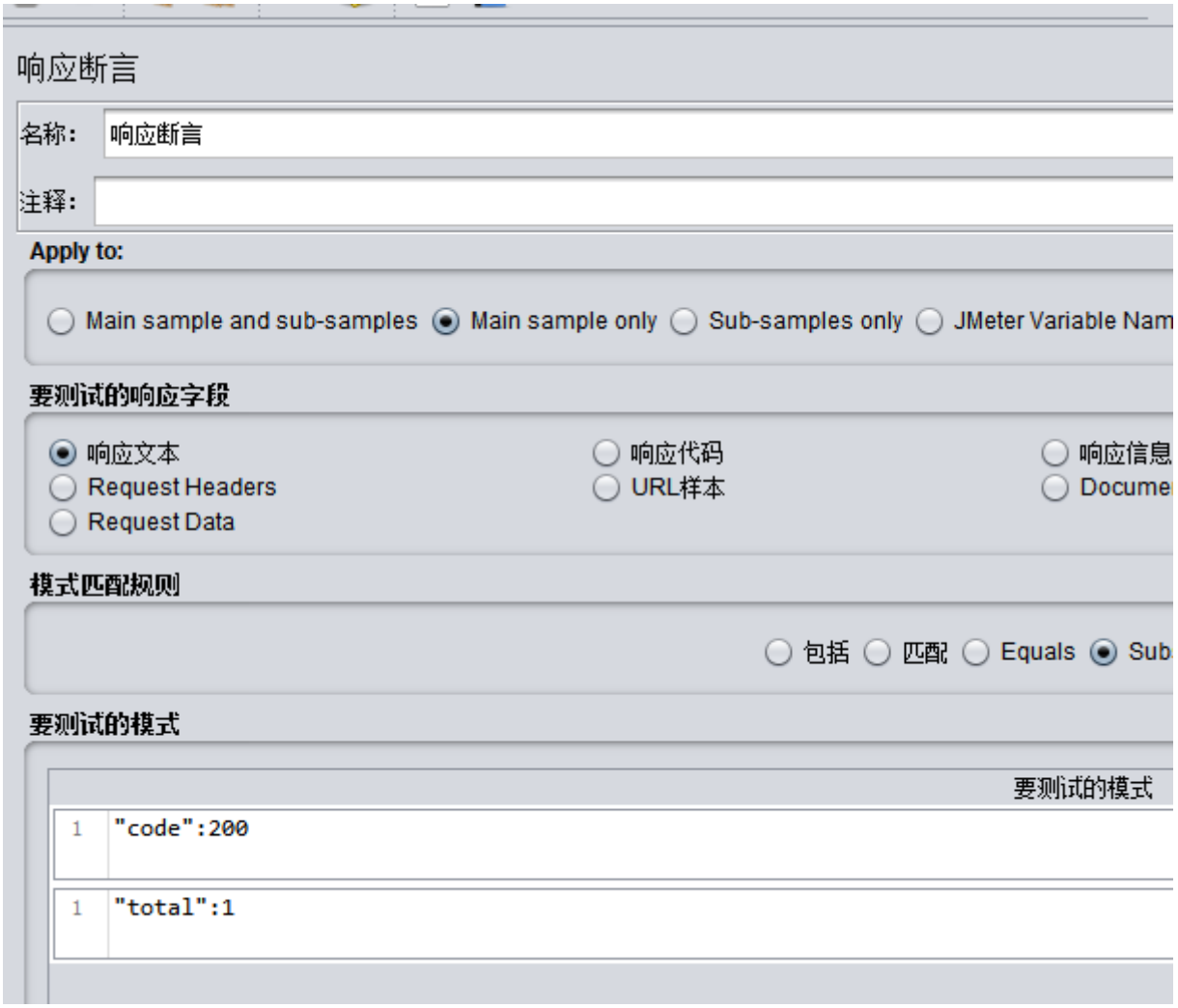
5、正则表达式提取规范，这一个体现了可维护性，可重用性，引用参数名称规范可以提供给其他用例调用。

正则表达式的提取器现在一般是从接口返回的响应数据中提取，引用名称就是响应数据的名称，如需提取同一个响应数据的同一字段的数据，则引用名称增加阿拉伯数字区分，vehModelName1,vehModelName2等等。

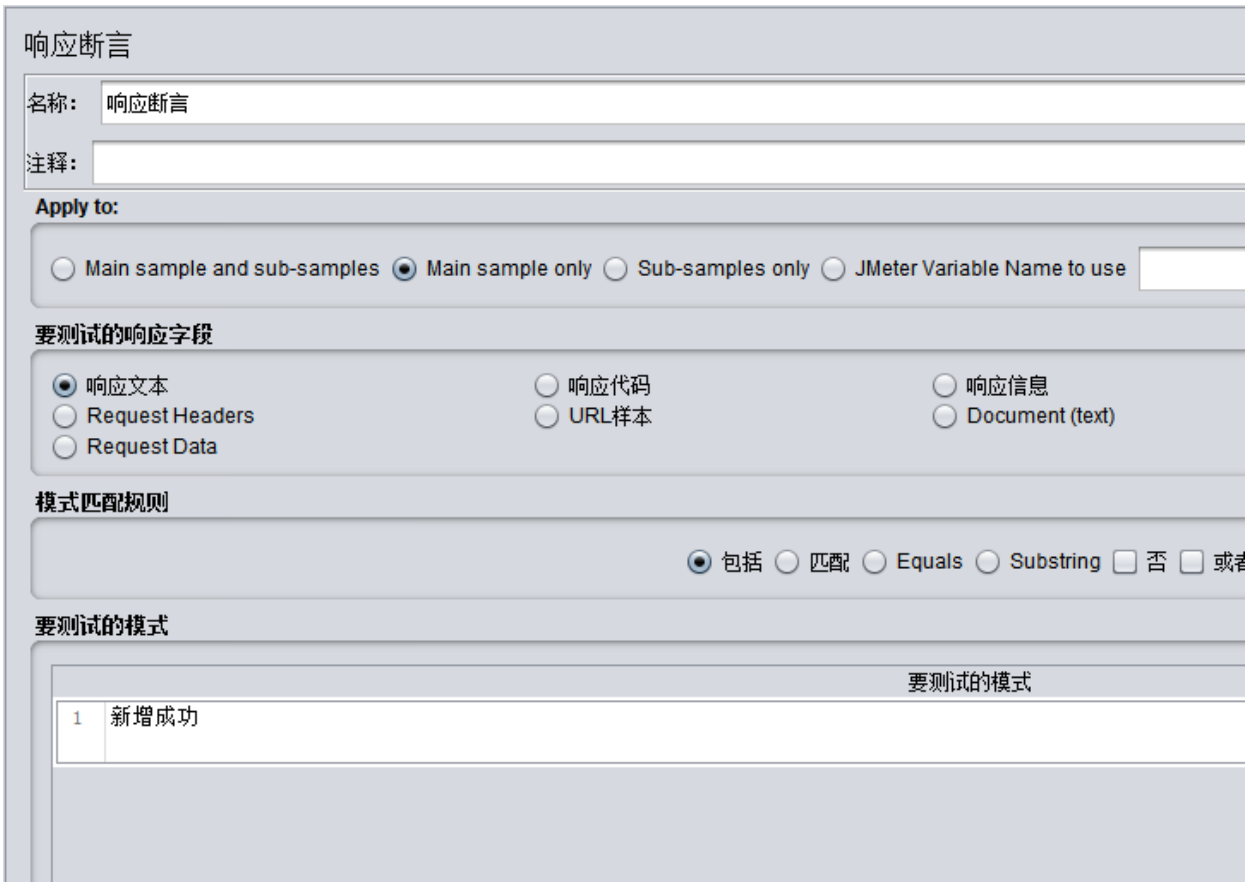


6、响应断言规范，这一个体现了可维护性，可重用性。

查询的响应断言规范，即状态code：200,以及对返回相关数据进行断言，可以是数量，也可以是返回的数据名等；



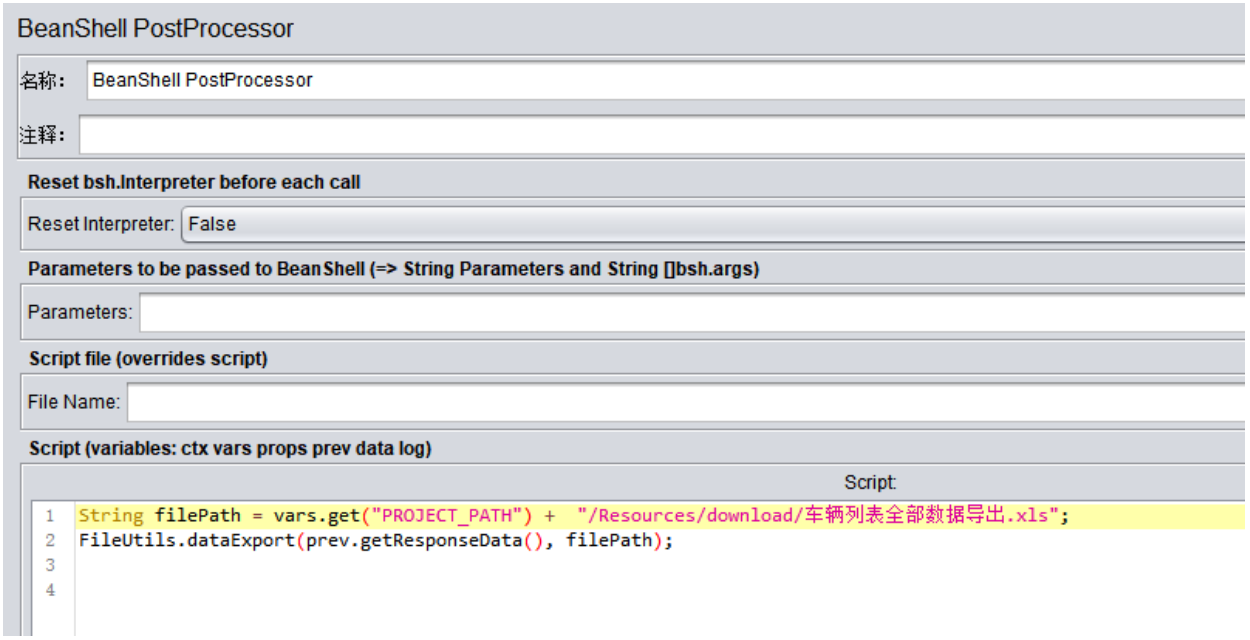
新增和更新的响应断言规范，即新增成功和更新成功即可；



删除的响应断言规范，删除成功，后面需要写上共删除了1条记录。因为删除失败的业务它接口返回也是删除成功，但后面是共删除了0条记录。

失败的新增/更新的响应断言，code的状态码都是-1，不需要写上其他提示信息，

7、导出和导入功能的beanshell规范；



BeanShell Sampler

名称： 1.1车载终端信息-生成导入文件

注释：

Reset bsh.Interpreter before each call

参数（-> String Parameters 和 String []bsh.args）

脚本文件

Script (see below for variables that are defined)

```
import org.apache.poi.ss.usermodel.Workbook;

        Workbook wb=new HSSFWorkbook();
        Sheet sheet0=wb.createSheet("Sheet0");

        Row row0=sheet0.createRow(0);
        row0.createCell(0).setCellValue("终端厂商自定义编号");
        row0.createCell(1).setCellValue("终端型号");
        row0.createCell(2).setCellValue("支持协议");
        row0.createCell(3).setCellValue("协议版本号");
        row0.createCell(4).setCellValue("ICCID");
        row0.createCell(5).setCellValue("IMEI");
        row0.createCell(6).setCellValue("固件版本号");
        row0.createCell(7).setCellValue("生产批次");
        row0.createCell(8).setCellValue("出厂日期");

        Row row1=sheet0.createRow(1);
        row1.createCell(0).setCellValue("${serialNumber}");
        row1.createCell(1).setCellValue("${termModelName}");
        row1.createCell(2).setCellValue("GBT32960标准版");
        row1.createCell(3).setCellValue("V3.15");
        row1.createCell(4).setCellValue("${iccid}");
        row1.createCell(5).setCellValue("${imei}");
        row1.createCell(6).setCellValue("V3.56");
        row1.createCell(7).setCellValue("201901");
        row1.createCell(8).setCellValue("2019-04-13");


        FileOutputStream fileout = null;
        String filePath = vars.get("PROJECT_PATH") +  "/Resources/upload/车载终端信息导入文件.xls";
       try {
            fileout = new FileOutputStream(filePath);
       } catch (FileNotFoundException e1) {
            e1.printStackTrace();
       }
        try {
            wb.write(fileout);
       } catch (IOException e) {
            e.printStackTrace();
       }
        try {
            fileout.close();
```

8、beanshell查删更改规范;

BeanShell Sampler

名称： 数据库查询（vin+车牌号+内部编号+ICCID+车辆型号+上牌城市+运营单位+运营区域+离线天数大于+起始时间）

注释：

Reset bsh.Interpreter before each call

参数（-> String Parameters 和 String []bsh.args）

脚本文件 浏览...

Script (see below for variables that are defined)

```
    String sql=
"SELECT veh.vin,veh.license_plate,veh.inter_no,veh.veh_model_id,veh.oper_unit_id,veh.oper_area_id,veh.oper_license_city_id,soc.start_time,tm.iccid FROM evsmc2.sys_vehicle as veh inner join evsmc2.
sys_soc_vehicle_log_${reportTime1}  as soc on soc.vid = veh.uuid inner join  evsmc2.sys_term_model_unit as tm on veh.term_id = tm.id where veh.vin is not null and veh.license_plate is not null and
 veh.inter_no is not null and veh.veh_model_id is not null and  veh.oper_unit_id is not null and veh.oper_area_id is not null and veh.oper_license_city_id is not null and soc.start_time is not nul
l and tm.iccid is not null and soc.status=0 limit 1";

        Connection conn=null;
        Statement stmt=null;
        ResultSet rs = null ;
        conn=DbUtil.getCon();
        stmt=conn.createStatement();
        rs = stmt.executeQuery(sql) ;
        System.out.println(sql);

        while(rs.next()){

            String pr1=rs.getString("vin");
            String pr2=rs.getString("license_plate");
            String pr3=rs.getString("inter_no");
            String pr4=rs.getString("veh_model_id");
            String pr5=rs.getString("oper_unit_id");
            String pr6=rs.getString("oper_area_id");
            String pr7=rs.getString("oper_license_city_id");
            String pr8=rs.getString("start_time");
            String pr9=rs.getString("iccid");

            vars.put("vin",pr1);
            vars.put("licensePlate",pr2);
            vars.put("interNo",pr3);
            vars.put("veh_model_id",pr4);
            vars.put("oper_unit_id",pr5);
            vars.put("oper_area_id",pr6);
            vars.put("oper_license_city_id",pr7);
            vars.put("start_time",pr8);
            vars.put("iccid",pr9);
        }

         DbUtil.closeCon(conn);
```

BeanShell Sampler

名称： 0.1删除实时车辆在线数据-sql查询最新创建的车辆数据

注释：

Reset bsh.Interpreter before each call

参数（-> String Parameters 和 String []bsh.args）

脚本文件

Script (see below for variables that are defined)

```
import java.sql.*;
import com.bitnei.util.*;


        String sql="SELECT id FROM sys_vehicle WHERE vin='JMETER12333129606' ";

        Connection conn=null;
        Statement stmt=null;
        ResultSet rs = null ;
        conn=DbUtil.getCon();
        stmt=conn.createStatement();
        rs = stmt.executeQuery(sql) ;

        while(rs.next()){
             String id=rs.getString("id");
             vars.put("sv.id",id);

        }

         DbUtil.closeCon(conn);
```

BeanShell Sampler

名称： 0.2删除实时无can车辆数据-sql插入最新的实时wucan数据

注释：

Reset bsh.Interpreter before each call

参数（-> String Parameters 和 String []bsh.args）

脚本文件

Script (see below for variables that are defined)

```
import java.sql.*;
import com.bitnei.util.*;


        String sql="UPDATE sys_vehicle_real_status SET online_status = '2',`update_time` = '${nowTimeWithHMS}' WHERE `vehicle_id` = '${sv.id}' ;";
        Connection conn=null;
        Statement stmt=null;
        conn=DbUtil.getCon();
        stmt=conn.createStatement();
        stmt.executeUpdate(sql) ;
        stmt.close();
        DbUtil.closeCon(conn);
```

BeanShell Sampler

名称： 1.3转发车辆黑名单-数据库新增刚刚被删除的车辆黑名单数据

注释：

Reset bsh.Interpreter before each call

参数（-> String Parameters 和 String []bsh.args）

脚本文件

Script (see below for variables that are defined)

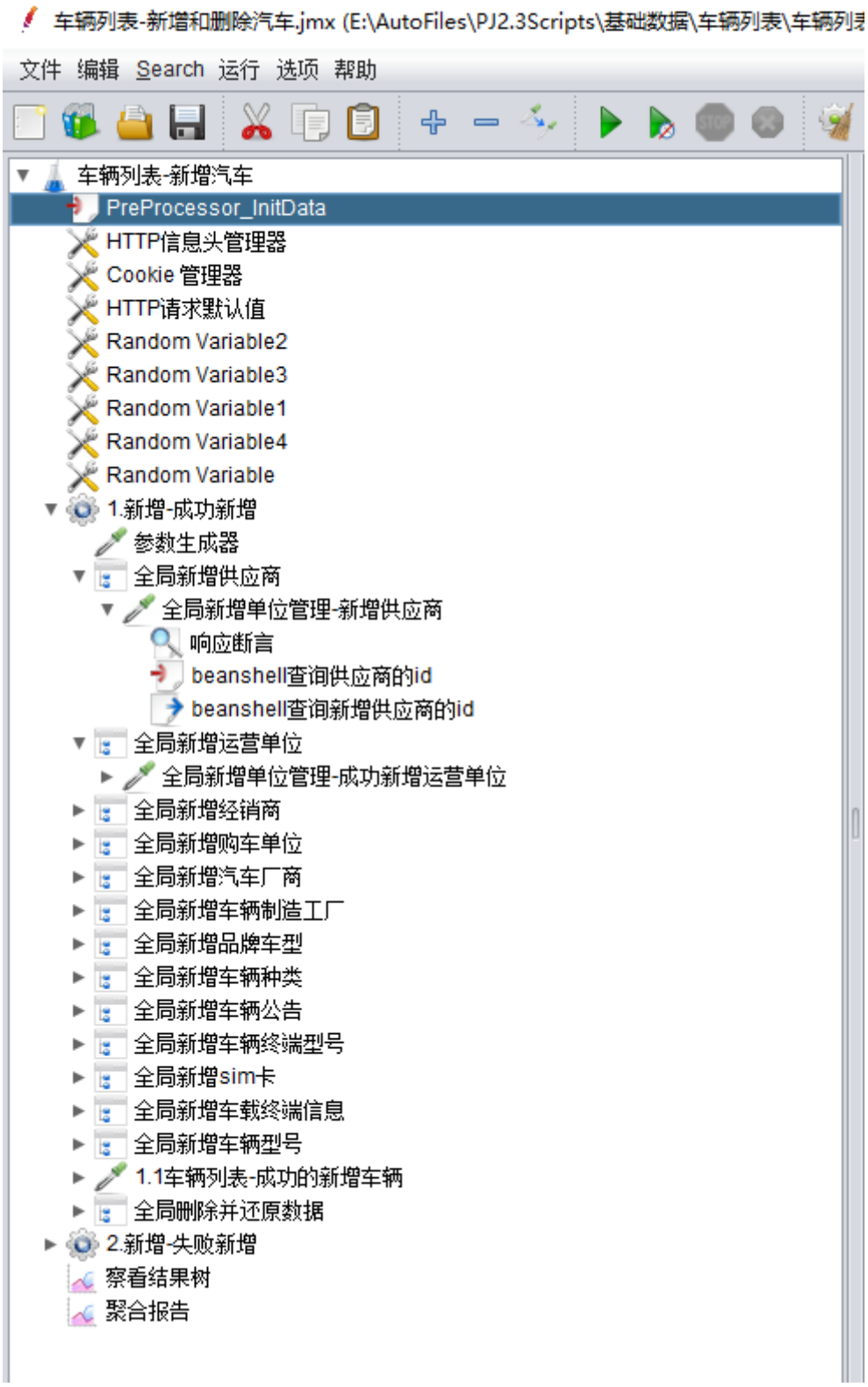
```
import java.sql.*;
import com.bitnei.util.*;

    String sql=
"INSERT INTO dc_forward_platform_vehicle_black_list( id,platform_id,vehicle_id,create_time,create_by) VALUES ( '${id}','${platform_id}', '${vehicle_id}','${create_time}','${create_by}')";
    Connection conn=null;
    Statement stmt=null;
    conn=DbUtil.getCon();
    stmt=conn.createStatement();
    stmt.executeUpdate(sql) ;
    stmt.close();
    DbUtil.closeCon(conn);
```

9.全局新增数据规范，这一个可以说是体现了可重用性，可维护性，逻辑分块。

引用到别的模块新增的数据，可以理解其为1个原子，我这里使用简单控制器来置放，并命名为全局XXXXX来单独区分，以便后期更好的维护。

主要是考虑如果上游接口数据变动导致出错，可以直接去上游修改，再返回需要引用的这个模块进行修改，这样能更好的提高维护效率。

比如车辆新增，需要引用供应商/运营单位/车辆制造工厂/车辆公告等这些上游模块的数据，一旦这些上游模块数据有变动导致报错，则应该先去上游模块修改接口，修改完毕后再回来车辆新增这里统一修改，由于这些引用的数据都是来自上游模块，只需复制粘贴即可。



-- END --

author 车联网一部测试组 黄永雄