Atitit 通用接口的设计与实现attilax 总结

[1. 现存的情况与预计目标 1](#_Toc16791)

[1.1. 实现零配置零注解 1](#_Toc29488)

[1.2. 4gl dsl sql sp接口 优先于3gl接口 1](#_Toc27977)

[1.3. 异常模型 代替返回值模型 1](#_Toc15808)

[1.4. 接口返回数据类型 指定 2](#_Toc3521)

[1.5. 返回结果 多种序列化格式 2](#_Toc8320)

[1.6. 异常传递模型 代替返回值模式 2](#_Toc19248)

[2. 通用接口原理与概念 2](#_Toc8392)

[2.1. 参照 Autoit Autohotkey pinvoke jna的模式 2](#_Toc25525)

[2.2. Ajax+ Jsbridge 3](#_Toc16834)

[2.3. Web rest版数据库接口， Rest+jdbc+mybatis+dbutil 3](#_Toc7500)

[3. 三大接口 3](#_Toc11017)

[3.1. 通用版全功能接口http param模式 范例 （ 可以运用于任何场合） 3](#_Toc23665)

[3.2. 通用版数据操作sql接口 （快速开发接口，适用与内部人员操作模块） 3](#_Toc17741)

[3.3. 通用版数据sp存储过程调用接口 （快速开发接口，适用于任何场合） 4](#_Toc11436)

[4. 后记 5](#_Toc1320)

[4.1. 核心代码 5](#_Toc12954)

[4.2. 未来的展望 dsl 模式 6](#_Toc7555)

[4.3. 未来的展望 dsl第二发展，支持表达式方法链 7](#_Toc31543)

# 现存的情况与预计目标

## 实现零配置零注解

增加接口通用度。，每增加一个子接口，无需做任何的配置和注解

## 4gl dsl sql sp接口 优先于3gl接口

其次对于数据库操作，3gl接口比较繁琐 ,增加了4gl dsl sql sp接口

## 异常模型 代替返回值模型

其次，接口的返回值模型稍有繁琐。。可以使用异常模型代替

## 接口返回数据类型 指定

返回类型类型就是ｓｔｒ　ｉｎｔ等，支持复杂格式map list,以及对象。

## 返回结果 多种序列化格式

返回序列化格式,即是结果使用什么样的序列化返回结果。。支持json ，预计还要支持xml yaml

## 异常传递模型 代替返回值模式

异常拥有比返回值更好的处理模式。

异常序列化为指定的序列化格式返回，传递给调用端。。

# 通用接口原理与概念

直接指明要调用的类与方法名。后台通过反射的模式调用。。

类似 pinvoke ， jna 模式

## 参照 Autoit Autohotkey pinvoke jna的模式

DllCall ( "dll", "返回值类型", "函数名称" [, 类型1, 参数1[, 类型n, 参数n]] )  
  
AutoIt中的参数类型与Win32 API中的参数类型不完全相同，这点要注意。

Autohotkey

Result := DllCall("[DllFile/]Function" [, Type1, Arg1, Type2, Arg2, "Cdecl ReturnType"])

## Ajax+ Jsbridge

## Web rest版数据库接口， Rest+jdbc+mybatis+dbutil

# 三大接口

## 通用版全功能接口http param模式 范例 （ 可以运用于任何场合）

[http://localhost:8080/](http://localhost:8080/AjaxJsbridge_HttpparamMode_servlet?iocFac=com.attilax.rest.JavaNewCreatorFac&createmode=new&class=com.attilax.rest.Class4test&method=m1&p1=123)[AjaxJsbridge\_HttpparamMode\_servlet?m=com.attilax.rest.Class4test.m1&p1=123](http://localhost:8080/AjaxJsbridge_HttpparamMode_servlet?iocFac=com.attilax.rest.JavaNewCreatorFac&createmode=new&class=com.attilax.rest.Class4test&method=m1&p1=123)

createmode参数： 类的创建模式 默认为动态new创建模式

。静态类的方法调用 为static

动态类，默认值，或者使用new

m：或者method ，指明要调用的方法,全类名加方法名，比如com.attilax.rest.Class4test.m1

retType:返回数据类型int str map list obj等

retFmt：返回数据序列化格式，一般为json，也可以为none,xml，默认为json

P1\_type ：第一个参数类型 有str int 等，默认为str

P1：第一个参数

P2\_type：第二个参数类型

P2:第二个参数

iocFac：ioc工厂：支持spring guice new 工厂模式，默认为com.attilax.rest.JavaNewCreatorFac

morennew 工厂模式为 com.attilax.rest.JavaNewCreatorFac

特点：：

全功能接口。

开发效率不是最高

## 通用版数据操作sql接口 （快速开发接口，适用与内部人员操作模块）

[http://localhost:8080/AjaxJsbridge\_HttpparamMode\_servlet?m=com.attilax.db.DbServiceV4qb9.executeQuery&p1=select+\*+from+ecs\_users+limit+10](http://localhost:8080/AjaxJsbridge_HttpparamMode_servlet?iocFac=com.attilax.rest.JavaNewCreatorFac&createmode=new&class=com.attilax.rest.Class4test&method=m1&p1=123)&iocFac=com.attilax.ioc.Ioc4other

注意：此数据接口是为快速开发而设置的，直接使用sql dsl存取数据，方便快捷。适用于后端管理，以及内部管理系统模块，适用于用户特定以及内部用户的模块。不适用于面向广大不特定用户的模块。。

面向广大不特定用户的模块需要隐藏sql，传递sql语句id即可，具体的sql语句应该存储在存储过程，代码或者配置文件里面 。。需要使用存储过程接口或其他接口即可

特点：：

开发效率貌似最高

非全功能接口，只针对数据操作接口。其次，不适用于面向公众人员使用的模块。

## 通用版数据sp存储过程调用接口 （快速开发接口，适用于任何场合）

调用存储过程mysql，即是使用特定sql语句，call 调用存储过程即可，比如call `sp\_查询用户`('mer')。。

[http://localhost:8080/AjaxJsbridge\_HttpparamMode\_servlet?m=com.attilax.db.DbServiceV4qb9.executeQuery&p1=call+%60sp\_%E6%9F%A5%E8%AF%A2%E7%94%A8%E6%88%B7%60%28%27mer%27%29](http://localhost:8080/AjaxJsbridge_HttpparamMode_servlet?iocFac=com.attilax.rest.JavaNewCreatorFac&createmode=new&class=com.attilax.rest.Class4test&method=m1&p1=123)&iocFac=com.attilax.ioc.Ioc4other

即是p1参数为 call `sp\_查询用户`('mer')。。 ,注意url参数的urlencode编码

特点：：

开发效率较高

非全功能接口，只针对数据操作接口。

# 后记

## 核心代码

|  |
| --- |
| @Override  **public** **void** **service**(ServletRequest req, ServletResponse response) {  **try** {  HttpServletRequest **req2** = (HttpServletRequest) req;  Global.*req*.set(req2);  String **iocx** =get\_iocFac( req.getParameter("iocFac"));  String **class\_meth\_full** = (getMeth(req));  String **cls** =refx.*getClassName*(class\_meth\_full);  String **meth\_name**=refx.*getMethodName*(class\_meth\_full);    Class **c** = Class.*forName*(cls);      List<Object> **params** = Lists.*newArrayList*();  **for** (**int** **i** = 1; i < 7; i++) {  String **pname** = "p" + i;  **if** (req.getParameter(pname) == **null**)  **continue**;  String **p** = req.getParameter(pname);  String **ptype** = getPtype(req, pname);  Object **p\_obj**=getP(p,ptype);  params.add(p\_obj);    }    Object[] **p\_a** = params.toArray();  String **createmode** =req.getParameter("createmode");  Object **rzt**;  **if**(isStaticMode(createmode))  {  rzt = MethodUtils.*invokeStaticMethod*(c, meth\_name, p\_a);    }  **else**//def is new  {  Object **cls\_obj** = com.attilax.reflect.MethodUtils.*invokeStaticMethod*(iocx, "getBean", c);  rzt = MethodUtils.*invokeMethod*(cls\_obj, meth\_name, p\_a);  }        System.***out***.println(rzt);  // Object r = new DslParser().parseV3(req2.getParameter("dsl"));  **if** (req.getParameter("retFmt") != **null**) {  **if** (req.getParameter("retFmt").equals("json"))  response.getWriter().println(AtiJson.*toJson*(rzt)); // �벻Ҫ�޸Ļ�ɾ��  } **else**  response.getWriter().println(rzt.toString()); // �벻Ҫ�޸Ļ�ɾ��  } **catch** (Exception **e**) {  ***logger***.error("--err", e);  ***logger***.error("--err2", AtiJson.*toJson*(e));  filex.*save\_safe*(~~core~~.*getTrace*(e), "c:\\0rechglog\\" + filex.*getUUidName*() + ".log");  ExUtil.*throwExV2*(e);  }  } |

## 未来的展望 dsl 模式

直接支持java表达式，<http://xxxxxx/api?dsl=new> com.xxx.userservice().reg()

## 未来的展望 dsl第二发展，支持表达式方法链