## 简介：

wServer适用于java后台接口开发的简易后台框架。

## 配置：

<config>

<mysql> <!--配置mysql数据库的连接-->

<url></url>

<user></user>

<password></password>

</mysql>

<server>

<ip>127.0.0.1</ip> <!--服务器的ip地址-->

<port>8080</port> <!--服务器的端口号-->

</server>

<servlets> <!--配置servlet的映射关系-->

<servlet uri="/index1" class="servlet.AServlet"></servlet>

<servlet uri="/index2" class="servlet.BServlet"></servlet>

</servlets>

</config>

## 简易使用：

### 如何写servlet：

例子：在servlet包下面建立Aservlet类，继承Servlet类，并且重载

public void business(Request req, Response res) throws Exception {} 函数

通过req.get\_params().get(“参数名字”)来获取参数

通过res.sendData(String data)来返回值

### 如何启动服务器：

HttpServer.start(); 这样就启动了服务器

## 进阶使用：

### 如何使用依赖注入：

***依赖注入的配置：***

<objects>

<object id="hello" class="test.Hello"> <!--id用来唯一标识对象，class表示对象所属的类-->

<property name="person"> <!--name是对象中某个属性的名字-->

<ref id="person1"></ref> <!--hello这个对象依赖的外部对象-->

</property>

</object>

<object id="person1" class="test.Person" singleton=”true”> <!--singleton表示对象是单例，如果没有这个属性则这个对象默认是单例-->

<property name="myTelephone">

<ref id="telephone"></ref>

</property>

<property name="name">

<value type="java.lang.String">weiyuan</value> <!--hello这个对象中的非引用属性-->

</property>

</object>

<!--下面的两个对象体现出了循环引用-->

<object id="person2" class="test.Person">

<property name="myTelephone">

<ref id="telephone"></ref>

</property>

<property name="name">

<value type="java.lang.String">lixin</value>

</property>

</object>

<object id="telephone" class="test.Telephone">

<property name="contact">

<ref id="person2"></ref>

</property>

</object>

</objects>

***使用依赖注入的对象需要在为其中的引用对象设置set函数：***

比如id为person2的对象中有一个myTelephone的引用属性，则需要在对象中增加

set\_myTelephone(Telephone t){}这个函数

***通过wServer的IOC框架获取对象的方法是：***

IOC ioc = IOC.getInstance();

Object object = ioc.getObject("对象在配置文件中的id");

### 如何使用AOP方式编程（依赖注入的应用）：

***写HelloInterface这个接口：***

public interface HelloInterface {

public void sayHello();

}

***写Hello类实现HelloInterface接口：***

public class Hello implements HelloInterface {

@Override

public void sayHello() {

System.out.println("hello, this is test of aop");

}

}

***继承AOP类，重载invoke函数：***

public class AOP implements InvocationHandler {

private Object service;// 业务对象

public Object proxy;

public void set\_service(Object svc) {// 绑定代理对象

this.service = svc;

proxy=Proxy.newProxyInstance(service.getClass().getClassLoader(), service.getClass().getInterfaces(), this);// 绑定代理对象的接口

}

@Override

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

Object result = null;

try{

// 这里是可以修改的部分

result = method.invoke(service, args);// 执行方法调用

// 这里是可以修改的部分

} catch(Exception e){

e.printStackTrace();

}

return result;

}

}

***写配置文件，语法等同于上面的依赖注入配置文件写法：***

<objects><!-- 对象依赖配置 -->

<object id="AOP" class="aop.AOP">

<property name="service">

<ref id="hello"></ref> <!--被代理对象的引用-->

</property>

</object>

<object id="hello" class="servlet.Hello"></object> <!--被代理对象-->

</objects>

***通过代理完成Hello类中的sayHello()业务：***

IOC ioc = IOC.getInstance();

AOP proxy = (AOP)ioc.getObject("AOP");

((HelloInterface)(proxy.proxy)).sayHello();

## 如何使用数据库连接池：

### 普通查询：

所有的增删改查都通过Query类中的函数链式调用执行，最后一个函数一定是

fetchOne()，fetchAll()， fetchGroup()，executeUpdate()，才能使查询得以执行。

***Select***

new Query().select("\*").table("表名")

.where("属性").equal(属性值)

.andWhere(“属性”).equal(属性值)

.fetchAll();

其中，where子句中包含了以下一些函数：

andWhere(String attribute) 表示and

orWhere(String attribute) 表示or

equal(Object value) 表示=

bigger(Object value) 表示>

smaller(Object value) 表示<

in(ArrayList<Object> list) in

limit子句：

limit(int begin, int number)

orderby子句：

orderBy(String str)

Groupby子句：

fetchGroup(String Group)

取得结果集中的一个:

fetchOne()

取得结果集中的所有：

fetchAll()

***Insert***

new Query().insertInto("表名(属性1, 属性2, 属性3)").values(params).executeUpdate();

其中params是ArrayList<Object>，按顺序存储着属性1，2，3的值

***Update***

new Query().update("表名").set(attr\_value).executeUpdate();

其中attr\_value是一个HashMap，key是更新的属性名，value是属性值

***Delete***

new Query().deleteFrom("表名").where("属性名").equal(属性值).executeUpdate();

### 事务更新（不支持查询）：

Trasaction t = new Trasaction();// 产生事务对象

ArrayList<Object> params = new ArrayList<Object>();

params.add("a"); params.add("b"); params.add(1111);

new Query().insertInto("test(a,b,c)").values(params).addToTrasaction(t);

new Query().insertInto("test(a,b,c)").values(params).addToTrasaction(t);

new Query().insertInto("test(a,b,c)").values(params).addToTrasaction(t);

new Query().insertInto("test(a,b,c)").values(params).addToTrasaction(t);

t.trasactionExecute();// 执行事务