<%@ page contentType=*"text/html;charset=UTF-8"* %>

<%@ page import=*"com.javatodo.core.JavaTodo"* %>

<%@ page import=*"com.javatodo.config.C"* %>

<%

JavaTodo javaTodo=**new** JavaTodo();

request.setCharacterEncoding(C.default\_encoding);

response.setCharacterEncoding(C.default\_encoding);

response.setHeader("Content-type", "text/html;charset="+C.default\_encoding);

javaTodo.setRequestAndResponse(request, response,**this**);

out.clear();

out = pageContext.pushBody();

%>

**package** com.javatodo.core;

**import** java.lang.reflect.InvocationTargetException;

**import** java.lang.reflect.Method;

**import** java.lang.reflect.Modifier;

**import** java.util.Map;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** com.javatodo.config.C;

**import** com.javatodo.core.router.RC;

**import** com.javatodo.core.router.Router;

**public** **class** JavaTodo {

**static** **boolean** *is\_init* = **false**;

**public** JavaTodo() {

**if** (!JavaTodo.*is\_init*) {

C.*set\_router*();

JavaTodo.*is\_init* = **true**;

}

}

**public** **void** setRequestAndResponse(HttpServletRequest request, HttpServletResponse response, HttpServlet servlet) {

Router router = **new** Router(request);

Map<String, String> routerMap = router.parse();

String package\_name = RC.*getRC*(routerMap.get("m"));

String class\_name = RC.*getRC*(routerMap.get("m"), routerMap.get("c"));

String function\_name = RC.*getRC*(routerMap.get("m"), routerMap.get("c"), routerMap.get("a"));

**try** {

Class<?> javatodo\_class = Class.*forName*(package\_name + "." + class\_name);

Object javatodo\_object = javatodo\_class.newInstance();

Class<?>[] javatodo\_args\_class = { HttpServletRequest.**class**, HttpServletResponse.**class**, HttpServlet.**class** };

Object[] javatodo\_args = { request, response, servlet };

Method javatodo\_set\_parameter = javatodo\_class.getMethod("setRequestAndResponse", javatodo\_args\_class); javatodo\_set\_parameter.invoke(javatodo\_object, javatodo\_args);

Method javatodo\_init = javatodo\_class.getMethod("init");

Method javatodo\_before = javatodo\_class.getMethod("\_before");

Method javatodo\_after = javatodo\_class.getMethod("\_after");

Method javatodo\_method = javatodo\_class.getMethod(function\_name);

Boolean initRet = (Boolean) javatodo\_init.invoke(javatodo\_object);

**if** (initRet != **true**) {

**return**;

}

javatodo\_before.invoke(javatodo\_object);

**if** (Modifier.*isStatic*(javatodo\_method.getModifiers())) {

javatodo\_method.invoke(**null**);

} **else** {

javatodo\_method.invoke(javatodo\_object);

}

javatodo\_after.invoke(javatodo\_object);

} **catch** (ClassNotFoundException | InstantiationException | IllegalAccessException | NoSuchMethodException

| SecurityException | IllegalArgumentException | InvocationTargetException e) {

e.printStackTrace();

}

}

}

**package** com.javatodo.core.controller;

**import** java.io.ByteArrayOutputStream;

**import** java.io.IOException;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.util.Enumeration;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.ServletOutputStream;

**import** javax.servlet.WriteListener;

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpServletResponseWrapper;

**import** javax.servlet.http.HttpSession;

**import** com.alibaba.fastjson.JSON;

**import** com.javatodo.config.C;

**import** com.javatodo.core.router.RC;

**import** com.javatodo.core.router.Router;

**import** com.javatodo.core.tools.Captcha;

**import** com.javatodo.core.tools.Page;

**import** com.javatodo.core.tools.T;

**import** com.javatodo.core.view.FreeMakerView;

**import** com.javatodo.core.view.JspView;

**import** com.javatodo.core.view.VelocityView;

**import** com.javatodo.core.view.View;

**import** freemarker.template.TemplateException;

**public** **class** Controller {

**public** **boolean** IS\_POST = **false**;

**public** HttpServletRequest request;

**public** HttpServletResponse response;

**public** HttpServlet servlet;

**public** View view;

**public** Map<String, String> routerMap = **null**;

**public** String MODULE\_NAME = "";

**public** String CONTROLLER\_NAME = "";

**public** String ACTION\_NAME = "";

**public** String PACKAGE\_NAME = "";

**public** String CLASS\_NAME = "";

**public** String FUNCTION\_NAME = "";

**public** **boolean** IS\_AJAX = **false**;

**public** String templatePath = C.*default\_template\_path*;

**public** String ROOT = "";

**public** String PUBLIC = "";

**private** Map<String, Object> assignMap = **new** HashMap<>();

**private** String templateSuffix = ".html";

**private** String entrance = "";

**public** **void** setRequestAndResponse(HttpServletRequest request, HttpServletResponse response, HttpServlet servlet) {

**this**.setParam(request, response, servlet);

**this**.setRouter(request);

**this**.response.setHeader("X-Powered-By", "JavaToDo");

**if** (C.*is\_debug* && C.*log\_file\_path*.equals("")) {

C.*log\_file\_path* = servlet.getServletContext().getRealPath("/") + "WEB-INF/log/";

}

}

**private** **void** setParam(HttpServletRequest request, HttpServletResponse response, HttpServlet servlet) {

**this**.request = request;

**this**.response = response;

**this**.servlet = servlet;

**if** ("POST".equals(request.getMethod())) {

**this**.IS\_POST = **true**;

}

**if** (**this**.request.getHeader("X-Requested-With") == **null**) {

**this**.IS\_AJAX = **false**;

} **else** {

**if** (**this**.request.getHeader("X-Requested-With").toString().equals("XMLHttpRequest")) {

**this**.IS\_AJAX = **true**;

} **else** {

**this**.IS\_AJAX = **false**;

}

}

**this**.ROOT = request.getContextPath();

**this**.PUBLIC = ROOT + "/" + C.*default\_template\_public*;

// 设置模版

**if** ("velocity".equals(C.*template\_engines*)) {

view = **new** VelocityView();

**this**.templateSuffix = ".html";

}

**if** ("jsp".equals(C.*template\_engines*)) {

view = **new** JspView(request);

**this**.templateSuffix = ".jsp";

}

**if** ("freemaker".equals(C.*template\_engines*)) {

view = **new** FreeMakerView();

**this**.templateSuffix = ".html";

}

// 设置入口文件

String url = request.getRequestURI();

String[] urlArr = url.split("/");

**if** (url.contains(".jsp")) {

**this**.entrance = urlArr[urlArr.length - 1];

} **else** {

**this**.entrance = "index.jsp";

}

}

**private** **void** setRouter(HttpServletRequest request) {

// 设置路由参数

Router router = **new** Router(request);

**this**.routerMap = router.parse();

**this**.MODULE\_NAME = router.MODULE\_NAME;

**this**.CONTROLLER\_NAME = router.CONTROLLER\_NAME;

**this**.ACTION\_NAME = router.ACTION\_NAME;

**this**.PACKAGE\_NAME = router.PACKAGE\_NAME;

**this**.CLASS\_NAME = router.CLASS\_NAME;

**this**.FUNCTION\_NAME = router.FUNCTION\_NAME;

}

**public** Boolean init() {

**return** **true**;

}

**public** **void** \_before() **throws** Exception {

}

**public** **void** \_after() **throws** Exception {

}

**public** Object session(String name) {

**if** (name == **null**) {

Enumeration<String> sNames = request.getSession().getAttributeNames();

**while** (sNames.hasMoreElements()) {

request.getSession().removeAttribute(sNames.nextElement().toString());

}

**return** **true**;

} **else** {

HttpSession session = request.getSession();

Object value = session.getAttribute(name);

**return** value;

}

}

**public** **void** session(String name, Object value) {

**if** (value == **null**) {

HttpSession session = request.getSession();

session.removeAttribute(name);

} **else** {

HttpSession session = request.getSession();

session.setAttribute(name, value);

}

}

**public** String cookie(String name) {

String value = **null**;

Cookie[] cookies = request.getCookies();

**for** (Cookie cookie : cookies) {

**if** (cookie.getName().toString().equals(name)) {

value = cookie.getValue();

**break**;

}

}

**return** value;

}

**public** **void** cookie(String name, String value) {

**if** (value == **null**) {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(0);

response.addCookie(cookie);

} **else** {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(3600);

response.addCookie(cookie);

}

}

**public** **void** cookie(String name, String value, **int** expiry) {

**if** (value == **null**) {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(0);

response.addCookie(cookie);

} **else** {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(expiry);

response.addCookie(cookie);

}

}

**public** **void** cookie(String name, String value, String domain) {

**if** (value == **null**) {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(0);

response.addCookie(cookie);

} **else** {

Cookie cookie = **new** Cookie(name, value);

cookie.setDomain(domain);

cookie.setPath("/");

response.addCookie(cookie);

}

}

**public** **void** cookie(String name, String value, **int** expiry, String domain) {

**if** (value == **null**) {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(0);

response.addCookie(cookie);

} **else** {

Cookie cookie = **new** Cookie(name, value);

cookie.setMaxAge(expiry);

cookie.setDomain(domain);

cookie.setPath("/");

response.addCookie(cookie);

}

}

**public** Map<String, String> I() {

Map<String, String> map = **this**.routerMap;

Map<String, String> retMap = **new** HashMap<>();

**for** (Map.Entry<String, String> entry : map.entrySet()) {

**if** (entry.getKey().toString().equals("m") || entry.getKey().toString().equals("c") || entry.getKey().toString().equals("a")) {

**continue**;

} **else** {

retMap.put(entry.getKey().toString(), T.*htmlspecialchars*(entry.getValue().toString()));

}

}

**return** retMap;

}

**public** String I(String name) {

String string = "";

**if** (routerMap.containsKey(name)) {

string = routerMap.get(name);

} **else** {

string = "";

}

string = T.*htmlspecialchars*(string);

**return** string;

}

**public** String U(String path) {

Map<String, String> map = **this**.routerMap;

String url = "";

String[] paths = path.split("/");

String root\_path = request.getRequestURI();

**if** (paths.length == 3) {

url = root\_path + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

**if** (paths.length == 2) {

url = root\_path + "?m=" + map.get("m") + "&c=" + paths[0] + "&a=" + paths[1];

}

**if** (paths.length == 1) {

url = root\_path + "?m=" + map.get("m") + "&c=" + map.get("c") + "&a=" + paths[0];

}

**return** url;

}

**public** String U(Map<String, String> map) {

String url = request.getRequestURI();

**if** (map.containsKey("m") && map.containsKey("c") && map.containsKey("a")) {

String paramUrl = "";

Integer i = 0;

**for** (Map.Entry<String, String> entry : map.entrySet()) {

**if** (i == 0) {

paramUrl = "?" + entry.getKey() + "=" + entry.getValue();

} **else** {

paramUrl = paramUrl + "&" + entry.getKey() + "=" + entry.getValue();

}

i = i + 1;

}

url = url + paramUrl;

}

**return** url;

}

**public** String U(String path, Map<String, Object> map) {

String url = "";

String[] paths = path.split("/");

String root\_path = request.getRequestURI();

**if** (paths.length == 3) {

url = root\_path + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

**if** (paths.length == 2) {

url = root\_path + "?m=" + **this**.routerMap.get("m") + "&c=" + paths[0] + "&a=" + paths[1];

}

**if** (paths.length == 1) {

url = root\_path + "?m=" + **this**.routerMap.get("m") + "&c=" + **this**.routerMap.get("c") + "&a=" + paths[0];

}

**for** (Entry<String, Object> entry : map.entrySet()) {

url = url + "&" + entry.getKey() + "=" + entry.getValue().toString();

}

**return** url;

}

**public** **void** redirect(String url) **throws** IOException {

response.sendRedirect(url);

}

**public** **void** Verify() **throws** IOException {

response.setContentType("image/jpeg");

response.setHeader("Pragma", "no-cache");

response.setHeader("Cache-Control", "no-cache");

response.setDateHeader("Expires", 0);

Captcha captcha = **new** Captcha();

session("verify\_code", captcha.getCode());

session("verify\_time", System.*currentTimeMillis*());

captcha.write(response.getOutputStream());

}

**public** **boolean** check\_verify(String code) {

**boolean** b = **false**;

String verify\_code = session("verify\_code").toString();

**long** dtime = System.*currentTimeMillis*() - Long.*parseLong*(session("verify\_time").toString());

**if** (code.toLowerCase().equals(verify\_code.toLowerCase()) && dtime < 1000 \* 60 \* 30) {

b = **true**;

} **else** {

b = **false**;

session("verify\_code", **null**);

session("verify\_time", **null**);

}

**return** b;

}

**public** **void** assign(String name, Object value) {

**this**.assignMap.put(name, value);

**this**.view.assign(name, value);

}

**private** **void** tempConstant() {

**this**.view.assign("IS\_POST", IS\_POST);

**this**.view.assign("IS\_AJAX", IS\_AJAX);

**this**.view.assign("MODULE\_NAME", MODULE\_NAME);

**this**.view.assign("CONTROLLER\_NAME", CONTROLLER\_NAME);

**this**.view.assign("ACTION\_NAME", ACTION\_NAME);

**this**.view.assign("ROOT", ROOT);

**this**.view.assign("PUBLIC", PUBLIC);

**this**.view.assign("controller", **this**);

**this**.view.assign("input", routerMap);

}

**private** String parseJsp(String path) **throws** ServletException, IOException {

ServletContext sc = **this**.servlet.getServletContext();

RequestDispatcher rd = sc.getRequestDispatcher(path);

**final** ByteArrayOutputStream os = **new** ByteArrayOutputStream();

**final** ServletOutputStream stream = **new** ServletOutputStream() {

**public** **void** write(**byte**[] data, **int** offset, **int** length) {

os.write(data, offset, length);

}

**public** **void** write(**int** b) **throws** IOException {

os.write(b);

}

@Override

**public** **boolean** isReady() {

// **TODO** Auto-generated method stub

**return** **false**;

}

@Override

**public** **void** setWriteListener(WriteListener arg0) {

// **TODO** Auto-generated method stub

}

};

**final** PrintWriter pw = **new** PrintWriter(**new** OutputStreamWriter(os));

HttpServletResponse rep = **new** HttpServletResponseWrapper(response) {

**public** ServletOutputStream getOutputStream() {

**return** stream;

}

**public** PrintWriter getWriter() {

**return** pw;

}

};

rd.include(request, rep);

pw.flush();

**return** os.toString();

}

**public** String parse() **throws** ServletException, IOException, TemplateException {

String path = "";

**this**.tempConstant();

**if** (**this**.templateSuffix.equals(".jsp")) {

path = "/" + **this**.templatePath;

path = path + "/" + routerMap.get("m").toString() + "/" + routerMap.get("c").toString() + "/" + routerMap.get("a").toString() + **this**.templateSuffix;

**return** **this**.parseJsp(path);

} **else** {

path = servlet.getServletContext().getRealPath("/") + **this**.templatePath;

path = path + "\\" + routerMap.get("m").toString() + "\\" + routerMap.get("c").toString() + "\\" + routerMap.get("a").toString() + **this**.templateSuffix;

**return** **this**.view.parseString(path, routerMap.get("m").toString() + "." + routerMap.get("c").toString() + "." + routerMap.get("a").toString() + ".log");

}

}

**public** String parse(String path) **throws** ServletException, IOException, TemplateException {

String root\_path = "";

**if** (**this**.templateSuffix.equals(".jsp")) {

root\_path = "/" + **this**.templatePath;

} **else** {

root\_path = servlet.getServletContext().getRealPath("/") + **this**.templatePath;

}

String[] paths = path.split("/");

Map<String, String> map = **new** HashMap<>();

**if** (paths.length == 3) {

map.put("m", RC.*getRC*(paths[0]));

map.put("c", RC.*getRC*(paths[0] + "!--javatodo--!" + paths[1]));

map.put("a", RC.*getRC*(paths[0] + "!--javatodo--!" + paths[1] + "!--javatodo--!" + paths[2]));

}

**if** (paths.length == 2) {

map.put("m", MODULE\_NAME);

map.put("c", RC.*getRC*(MODULE\_NAME + "!--javatodo--!" + paths[0]));

map.put("a", RC.*getRC*(MODULE\_NAME + "!--javatodo--!" + paths[0] + "!--javatodo--!" + paths[1]));

}

**if** (paths.length == 1) {

map.put("m", MODULE\_NAME);

map.put("m", CONTROLLER\_NAME);

map.put("a", RC.*getRC*(MODULE\_NAME + "!--javatodo--!" + CONTROLLER\_NAME + "!--javatodo--!" + paths[0]));

}

path = root\_path + "\\" + map.get("m").toString() + "\\" + map.get("c").toString() + "\\" + map.get("a").toString() + **this**.templateSuffix;

**this**.tempConstant();

**if** (**this**.templateSuffix.equals(".jsp")) {

**return** **this**.parseJsp(path);

} **else** {

**return** **this**.view.parseString(path, routerMap.get("m").toString() + "." + routerMap.get("c").toString() + "." + routerMap.get("a").toString() + ".log");

}

}

**public** **void** display() **throws** IOException, ServletException, TemplateException {

String path = "";

**if** (**this**.templateSuffix.equals(".jsp")) {

path = "/" + **this**.templatePath;

} **else** {

path = servlet.getServletContext().getRealPath("/") + **this**.templatePath;

}

path = path + "\\" + PACKAGE\_NAME + "\\" + CLASS\_NAME + "\\" + FUNCTION\_NAME + **this**.templateSuffix;

**this**.tempConstant();

**this**.view.flush(request, response, servlet, path);

}

**public** **void** display(String path) **throws** IOException, ServletException, TemplateException {

String root\_path = "";

**if** (**this**.templateSuffix.equals(".jsp")) {

root\_path = "/" + **this**.templatePath;

} **else** {

root\_path = servlet.getServletContext().getRealPath("/") + **this**.templatePath;

}

String[] paths = path.split("/");

Map<String, String> map = **new** HashMap<>();

**if** (paths.length == 3) {

map.put("m", RC.*getRC*(paths[0]));

map.put("c", RC.*getRC*(paths[0], paths[1]));

map.put("a", RC.*getRC*(paths[0], paths[1], paths[2]));

}

**if** (paths.length == 2) {

map.put("m", PACKAGE\_NAME);

map.put("c", RC.*getRC*(MODULE\_NAME, paths[0]));

map.put("a", RC.*getRC*(MODULE\_NAME, paths[0], paths[1]));

}

**if** (paths.length == 1) {

map.put("m", PACKAGE\_NAME);

map.put("c", CLASS\_NAME);

map.put("a", RC.*getRC*(MODULE\_NAME, CONTROLLER\_NAME, paths[0]));

}

path = root\_path + "\\" + map.get("m").toString() + "\\" + map.get("c").toString() + "\\" + map.get("a").toString() + **this**.templateSuffix;

**this**.tempConstant();

**this**.view.flush(request, response, servlet, path);

}

// 以json方式输出内容

**public** **void** jsonDisplay() {

String json = JSON.*toJSONString*(**this**.assignMap);

PrintWriter writer = **null**;

**try** {

**this**.response.setHeader("Pragma", "no-cache");

**this**.response.setHeader("Cache-Control", "no-cache");

**this**.response.setDateHeader("Expires", 0);

**this**.response.setContentType("application/json; charset=" + C.*default\_encoding*);

writer = response.getWriter();

writer.write(json);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **finally** {

**if** (writer != **null**) {

writer.close();

}

}

}

// 以json方式输出内容

**public** **void** jsonDisplay(String text) {

String json = text;

PrintWriter writer = **null**;

**try** {

**this**.response.setHeader("Pragma", "no-cache");

**this**.response.setHeader("Cache-Control", "no-cache");

**this**.response.setDateHeader("Expires", 0);

**this**.response.setContentType("application/json; charset=" + C.*default\_encoding*);

writer = response.getWriter();

writer.write(json);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **finally** {

**if** (writer != **null**) {

writer.close();

}

}

}

**public** Page page(**int** total, **int** num) {

**return** **new** Page(total, num, **this**.entrance, **this**.routerMap);

}

**public** **void** error(String errMsg) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "error");

session("javatodo\_jump\_msg", errMsg);

session("javatodo\_jump\_seconds", 3);

session("javatodo\_jump\_url", "javascript:history.go(-1)");

Map<String, String> map = **new** HashMap<>();

map.put("m", "com.javatodo.core.controller");

map.put("c", "Controller");

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 0);

**this**.assignMap.put("info", errMsg);

**this**.assignMap.put("url", "");

**this**.jsonDisplay();

}

}

/\*\*

\* 带有错误信息的跳转页面，例如“操作错误”，并且自动跳转到另外一个目标页面

\*

\* **@param** errMsg

\* String 要提示给用户的错误信息

\* **@param** url

\* String 要跳转的目标链接

\* **@throws** IOException

\*/

**public** **void** error(String errMsg, String url) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "error");

session("javatodo\_jump\_msg", errMsg);

session("javatodo\_jump\_seconds", 3);

session("javatodo\_jump\_url", url);

Map<String, String> map = **new** HashMap<>();

map.put("m", "com.javatodo.core.controller");

map.put("c", "Controller");

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 0);

**this**.assignMap.put("info", errMsg);

**this**.assignMap.put("url", url);

**this**.jsonDisplay();

}

}

**public** **void** error(String errMsg, String url, Integer seconds) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "error");

session("javatodo\_jump\_msg", errMsg);

session("javatodo\_jump\_seconds", seconds);

session("javatodo\_jump\_url", url);

Map<String, String> map = **new** HashMap<>();

map.put("m", **this**.MODULE\_NAME);

map.put("c", **this**.CONTROLLER\_NAME);

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 0);

**this**.assignMap.put("info", errMsg);

**this**.assignMap.put("url", url);

**this**.jsonDisplay();

}

}

**public** **void** success(String sucMsg) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "success");

session("javatodo\_jump\_msg", sucMsg);

session("javatodo\_jump\_seconds", 3);

session("javatodo\_jump\_url", request.getHeader("Referer"));

Map<String, String> map = **new** HashMap<>();

map.put("m", **this**.MODULE\_NAME);

map.put("c", **this**.CONTROLLER\_NAME);

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 1);

**this**.assignMap.put("info", sucMsg);

**this**.assignMap.put("url", "");

**this**.jsonDisplay();

}

}

**public** **void** success(String sucMsg, String url) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "success");

session("javatodo\_jump\_msg", sucMsg);

session("javatodo\_jump\_seconds", 3);

session("javatodo\_jump\_url", url);

Map<String, String> map = **new** HashMap<>();

map.put("m", **this**.MODULE\_NAME);

map.put("c", **this**.CONTROLLER\_NAME);

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 1);

**this**.assignMap.put("info", sucMsg);

**this**.assignMap.put("url", url);

**this**.jsonDisplay();

}

}

**public** **void** success(String sucMsg, String url, Integer seconds) **throws** IOException {

**if** (!IS\_AJAX) {

session("javatodo\_jump\_type", "success");

session("javatodo\_jump\_msg", sucMsg);

session("javatodo\_jump\_seconds", seconds);

session("javatodo\_jump\_url", url);

Map<String, String> map = **new** HashMap<>();

map.put("m", **this**.MODULE\_NAME);

map.put("c", **this**.CONTROLLER\_NAME);

map.put("a", "jump");

**this**.redirect(**this**.U(map));

} **else** {

**this**.assignMap.clear();

**this**.assignMap.put("status", 1);

**this**.assignMap.put("info", sucMsg);

**this**.assignMap.put("url", url);

**this**.jsonDisplay();

}

}

**public** **void** jump() **throws** IOException, ServletException, TemplateException {

String root\_path = "";

**if** (**this**.templateSuffix.equals(".jsp")) {

root\_path = "/" + **this**.templatePath;

} **else** {

root\_path = servlet.getServletContext().getRealPath("/") + **this**.templatePath;

}

String path = root\_path + "\\system\\jump" + **this**.templateSuffix;

**this**.assign("type", session("javatodo\_jump\_type"));

**this**.assign("msg", session("javatodo\_jump\_msg"));

**this**.assign("seconds", session("javatodo\_jump\_seconds"));

**this**.assign("url", session("javatodo\_jump\_url"));

**this**.view.flush(request, response, servlet, path);

}

}

**package** com.javatodo.config;

**import** com.javatodo.core.router.RC;

**public** **class** C {

// 数据库配置项

**public** **static** String *db\_type* = "mysql";// 数据库类型

// 可以同时连接多个数据库。

**public** **static** String[] *db\_host* = {};// 数据库地址

**public** **static** String[] *db\_port* = {};// 数据库端口

**public** **static** String[] *db\_name* = {};// 数据库名称

**public** **static** String[] *db\_username* = {};// 数据库用户名

**public** **static** String[] *db\_password* = {};// 数据库密码

**public** **static** String[] *table\_pre* = {};// 数据表前缀

// 是否调试模式

**public** **static** **boolean** *is\_debug* = **true**;

// 日志文件夹

**public** **static** String *log\_file\_path* = "";

// 设置使用的模版引擎：velocity、jsp、freemaker

**public** **static** String *template\_engines* = "jsp";

// 设置编码

**public** **static** String *default\_encoding* = "utf-8";

// 默认路由配置项

**public** **static** String *default\_module* = "index";

**public** **static** String *default\_controller* = "Index";

**public** **static** String *default\_action* = "index";

// 路由

**public** **static** **void** set\_router() {

**new** RC("com.javatodo.app.index", "index");

}

/\*\* 以下配置项如无特殊需要请不要更改 \*\*/

// 默认模版文件目录

**public** **static** String *default\_template\_path* = "WEB-INF/Template";

**public** **static** String *default\_template\_public* = "Public";

// 数据库连接池C3P0配置信息

**public** **static** **int** *MaxPoolSize* = 100;

**public** **static** **int** *MinPoolSize* = 10;

**public** **static** **int** *InitialPoolSize* = 10;

**public** **static** **int** *MaxIdleTime* = 20;

**public** **static** **int** *AcquireIncrement* = 2;

// 验证码图片配置信息

**public** **static** **int** *CaptchaWidth* = 280;// 验证码默认图片宽度

**public** **static** **int** *CaptchaHeight* = 90;// 验证码默认图片高度

**public** **static** **int** *CaptchaCodeNum* = 5;// 验证码默认字符数

**public** **static** **int** *CaptchaLineNum* = 250;// 验证码默认干扰线条数

**public** **static** String *CaptchaFont* = "宋体";// 验证码默认字体

// 文件上传配置

**public** **static** **long** *UploadMaxSize* = 1024 \* 1024 \* 5;// 默认文件上传最大限制为5M

}

**package** com.javatodo.core.model;

**import** java.util.List;

**import** java.util.Map;

**public** **abstract** **class** Driver {

**public** **abstract** **void** table(String table\_name);

**public** **abstract** Driver where(Map<String, W> where);

**public** **abstract** Driver where(String where\_str);

**public** **abstract** Driver where(String where\_str, Object... params);

**public** **abstract** Driver order(String order\_str);

**public** **abstract** Driver limit(String limit\_str);

**public** **abstract** Driver data(Map<String, Object> data);

**public** **abstract** Driver alias(String as\_str);

**public** **abstract** Driver join(String table\_name, String on\_sql);

**public** **abstract** Driver join(String table\_name, String on\_sql, String type);

**public** **abstract** Driver field(String field\_str);

**public** **abstract** **void** group(String field\_str);

**public** **abstract** Driver add();

**public** **abstract** Driver save(Map<String, Object> data);

**public** **abstract** Driver delete();

**public** **abstract** Driver select();

**public** **abstract** Driver find();

**public** **abstract** String get\_sql();

**public** **abstract** List<Object> get\_add\_data();

**public** **abstract** List<Object> get\_update\_data();

**public** **abstract** List<Object> get\_where\_data();

**public** **abstract** Driver setInc(String field, Integer value);

**public** **abstract** Driver setDec(String field, Integer value);

**public** **abstract** Driver setInc(String field);

**public** **abstract** Driver setDec(String field);

**public** **abstract** **void** clear();

}

**package** com.javatodo.core.model;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** org.apache.commons.dbutils.QueryRunner;

**import** org.apache.commons.dbutils.handlers.MapHandler;

**import** org.apache.commons.dbutils.handlers.MapListHandler;

**public** **class** M {

**private** Connection connection = **null**;

**private** QueryRunner queryRunner = **new** QueryRunner();

**private** Driver db = **null**;

**private** String lastSql = "";

**private** Object sql\_params;

**private** **boolean** is\_transaction = **false**;

**public** M() {

**this**.connection = MC.*get\_connection*();

**if** (**this**.connection == **null**) {

System.***out***.println("找不到数据源");

} **else** {

**if** ("mysql".equals(MC.*db\_type*)) {

**this**.db = **new** MysqlDriver();

}

**if** ("postgresql".equals(MC.*db\_type*)) {

**this**.db = **new** PgsqlDriver();

}

}

}

**public** M(Integer dbIndex) {

**this**.connection = MC.*get\_connection*(dbIndex);

**if** (**this**.connection == **null**) {

System.***out***.println("找不到数据源");

} **else** {

**if** ("mysql".equals(MC.*db\_type*)) {

**this**.db = **new** MysqlDriver(dbIndex);

}

**if** ("postgresql".equals(MC.*db\_type*)) {

**this**.db = **new** PgsqlDriver(dbIndex);

}

}

}

**public** M(String table\_name) {

**this**.connection = MC.*get\_connection*();

**if** (**this**.connection == **null**) {

System.***out***.println("找不到数据源");

} **else** {

**if** ("mysql".equals(MC.*db\_type*)) {

**this**.db = **new** MysqlDriver(table\_name);

}

**if** ("postgresql".equals(MC.*db\_type*)) {

**this**.db = **new** PgsqlDriver(table\_name);

}

}

}

**public** M(String table\_name, Integer dbIndex) {

**this**.connection = MC.*get\_connection*(dbIndex);

**if** (**this**.connection == **null**) {

System.***out***.println("找不到数据源");

} **else** {

**if** ("mysql".equals(MC.*db\_type*)) {

**this**.db = **new** MysqlDriver(table\_name, dbIndex);

}

**if** ("postgresql".equals(MC.*db\_type*)) {

**this**.db = **new** PgsqlDriver(table\_name, dbIndex);

}

}

}

**public** M table(String table\_name) {

**this**.db.table(table\_name);

**return** **this**;

}

**public** **void** transaction() **throws** SQLException {

**this**.is\_transaction = **true**;

**if** (**this**.connection != **null**) {

**this**.connection.setAutoCommit(**false**);

} **else** {

**this**.connection = MC.*get\_connection*();

**if** (**this**.connection != **null**) {

**this**.connection.setAutoCommit(**false**);

}

}

}

**public** **void** commit() **throws** SQLException {

**if** (is\_transaction == **false**) {

**return**;

} **else** {

**this**.is\_transaction = **false**;

**if** (**this**.connection != **null**) {

**this**.connection.commit();

**this**.close();

}

}

}

**public** **void** rollback() {

**if** (is\_transaction == **false**) {

**return**;

} **else** {

**this**.is\_transaction = **false**;

**if** (**this**.connection != **null**) {

**try** {

**this**.connection.rollback();

**this**.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** M where(Map<String, W> where) {

**this**.db.where(where);

**return** **this**;

}

**public** M where(String where\_str) {

**this**.db.where(where\_str);

**return** **this**;

}

**public** M where(String where\_str, Object... params) {

**this**.db.where(where\_str, params);

**return** **this**;

}

**public** M order(String order\_str) {

**this**.db.order(order\_str);

**return** **this**;

}

**public** M limit(String limit\_str) {

**this**.db.limit(limit\_str);

**return** **this**;

}

**public** M data(Map<String, Object> data) {

**this**.db.data(data);

**return** **this**;

}

**public** M alias(String as\_str) {

**this**.db.alias(as\_str);

**return** **this**;

}

**public** M join(String table\_name, String on\_sql) {

**this**.db.join(table\_name, on\_sql);

**return** **this**;

}

**public** M join(String table\_name, String on\_sql, String type) {

**this**.db.join(table\_name, on\_sql, type);

**return** **this**;

}

**public** M field(String field\_str) {

**this**.db.field(field\_str);

**return** **this**;

}

**public** M group(String field\_str) {

**this**.db.group(field\_str);

**return** **this**;

}

**public** Object add() **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

Object lastId = **null**;

**if** (**this**.connection != **null**) {

**this**.db.add();

String sql = **this**.db.get\_sql();

List<Object> add\_data\_list = **this**.db.get\_add\_data();

Object[] params = **new** Object[add\_data\_list.size()];

**for** (Integer integer = 0; integer < add\_data\_list.size(); integer = integer + 1) {

params[integer] = add\_data\_list.get(integer);

}

**this**.queryRunner.update(**this**.connection, sql, params);

List<Map<String, Object>> list = **this**.query("SELECT LAST\_INSERT\_ID()");

**if** (list.size() > 0) {

lastId = list.get(0).get("LAST\_INSERT\_ID()");

}

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

**return** lastId;

}

**public** **void** save(Map<String, Object> data) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.save(data);

String sql = **this**.db.get\_sql();

List<Object> update\_data\_list = **this**.db.get\_update\_data();

List<Object> where\_data\_list = **this**.db.get\_where\_data();

Integer all\_total = update\_data\_list.size() + where\_data\_list.size();

Object[] params = **new** Object[all\_total];

**for** (Integer integer = 0; integer < update\_data\_list.size(); integer = integer + 1) {

params[integer] = update\_data\_list.get(integer);

}

**for** (Integer integer = update\_data\_list.size(); integer < all\_total; integer = integer + 1) {

params[integer] = where\_data\_list.get(integer - update\_data\_list.size());

}

**this**.queryRunner.update(**this**.connection, sql, params);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** setInc(String field, Integer value) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.setInc(field, value);

String sql = **this**.db.get\_sql();

**this**.queryRunner.update(**this**.connection, sql);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** setInc(String field) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.setInc(field);

String sql = **this**.db.get\_sql();

**this**.queryRunner.update(**this**.connection, sql);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** setDec(String field, Integer value) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.setDec(field, value);

String sql = **this**.db.get\_sql();

**this**.queryRunner.update(**this**.connection, sql);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** setDec(String field) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.setDec(field);

String sql = **this**.db.get\_sql();

**this**.queryRunner.update(**this**.connection, sql);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** delete() **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.delete();

String sql = **this**.db.get\_sql();

List<Object> where\_data\_list = **this**.db.get\_where\_data();

Object[] params = **new** Object[where\_data\_list.size()];

**for** (Integer integer = 0; integer < where\_data\_list.size(); integer = integer + 1) {

params[integer] = where\_data\_list.get(integer);

}

**this**.queryRunner.update(**this**.connection, sql, params);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** List<Map<String, Object>> select() **throws** SQLException {

List<Map<String, Object>> list = **new** ArrayList<>();

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.db.select();

String sql = **this**.db.get\_sql();

List<Object> where\_data\_list = **this**.db.get\_where\_data();

Object[] params = **new** Object[where\_data\_list.size()];

**for** (Integer integer = 0; integer < where\_data\_list.size(); integer = integer + 1) {

params[integer] = where\_data\_list.get(integer);

}

list = queryRunner.query(**this**.connection, sql, **new** MapListHandler(), params);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

**return** list;

} **else** {

**return** list;

}

}

**public** Map<String, Object> find() **throws** SQLException {

Map<String, Object> map = **null**;

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

map = **new** HashMap<>();

**this**.db.find();

String sql = **this**.db.get\_sql();

List<Object> where\_data\_list = **this**.db.get\_where\_data();

Object[] params = **new** Object[where\_data\_list.size()];

**for** (Integer integer = 0; integer < where\_data\_list.size(); integer = integer + 1) {

params[integer] = where\_data\_list.get(integer);

}

map = **this**.queryRunner.query(**this**.connection, sql, **new** MapHandler(), params);

**this**.db.clear();

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

**return** map;

} **else** {

**return** map;

}

}

**public** Object getField(String field\_name) **throws** SQLException {

Object object = **null**;

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

object = **new** Object();

**this**.db.field(field\_name);

Map<String, Object> map = **this**.find();

**if** (map == **null**) {

**return** **null**;

}

**if** (map.containsKey(field\_name)) {

object = map.get(field\_name);

} **else** {

object = **null**;

}

**if** (!**this**.is\_transaction) {

**this**.close();

}

**return** object;

} **else** {

**return** object;

}

}

**public** List<Map<String, Object>> query(String sql) **throws** SQLException {

List<Map<String, Object>> list = **new** ArrayList<>();

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

list = queryRunner.query(**this**.connection, sql, **new** MapListHandler());

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

**return** list;

} **else** {

**return** list;

}

}

**public** List<Map<String, Object>> query(String sql, Object... params) **throws** SQLException {

List<Map<String, Object>> list = **new** ArrayList<>();

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

list = queryRunner.query(**this**.connection, sql, **new** MapListHandler(), params);

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

**return** list;

} **else** {

**return** list;

}

}

**public** Integer count() **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

Object count = **this**.getField("count(\*)");

**if** (!**this**.is\_transaction) {

**this**.close();

}

**if** (count == **null**) {

**return** **null**;

} **else** {

**return** Integer.*parseInt*(count.toString());

}

} **else** {

**return** **null**;

}

}

**public** **void** execute(String sql) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.queryRunner.update(**this**.connection, sql);

**this**.lastSql = sql;

**this**.sql\_params = "";

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** execute(String sql, Object... params) **throws** SQLException {

**if** (**this**.connection == **null**) {

**this**.connection = MC.*get\_connection*();

}

**if** (**this**.connection != **null**) {

**this**.queryRunner.update(**this**.connection, sql, params);

**this**.lastSql = sql;

**this**.sql\_params = params;

**if** (!**this**.is\_transaction) {

**this**.close();

}

}

}

**public** **void** getLastSql() {

System.***out***.println(**this**.lastSql);

**if** (**this**.sql\_params.getClass().getName().contains("String")) {

System.***out***.println(**this**.sql\_params);

} **else** {

Object[] objects = (Object[]) **this**.sql\_params;

List<String> list = **new** ArrayList<>();

**for** (Integer i = 0; i < objects.length; i = i + 1) {

list.add(objects[i].toString());

}

System.***out***.println(list);

}

}

**private** **void** close() {

**if** (**this**.connection != **null**) {

**try** {

**this**.connection.close();

**this**.connection = **null**;

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**package** com.javatodo.core.model;

**import** java.beans.PropertyVetoException;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** com.javatodo.config.C;

**import** com.mchange.v2.c3p0.ComboPooledDataSource;

**public** **class** MC {

**public** **static** List<ComboPooledDataSource> *dataSource* = **null**;

**public** **static** String *db\_type* = C.*db\_type*;// 数据库类型

**public** **static** List<String> *table\_pre* = **new** ArrayList<String>(Arrays.*asList*(C.*table\_pre*));// 数据表前缀

**public** **static** List<String> *db\_host* = **new** ArrayList<String>(Arrays.*asList*(C.*db\_host*));// 数据库地址

**public** **static** List<String> *db\_port* = **new** ArrayList<String>(Arrays.*asList*(C.*db\_port*));// 数据库端口

**public** **static** List<String> *db\_name* = **new** ArrayList<String>(Arrays.*asList*(C.*db\_name*));// 数据库名称

**public** **static** List<String> *db\_username* = **new** ArrayList<String>(Arrays.*asList*(C.*db\_username*));// 数据库用户名

**public** **static** List<String> *db\_password* = **new** ArrayList<String>(Arrays.*asList*(C.*db\_password*));// 数据库密码

**static** {

**if** (MC.*dataSource* == **null** && MC.*db\_host*.size() > 0) {

**try** {

MC.*dataSource* = **new** ArrayList<ComboPooledDataSource>();

**for** (Integer i = 0; i < MC.*db\_host*.size(); i = i + 1) {

ComboPooledDataSource source = **new** ComboPooledDataSource();

**if** (MC.*db\_type*.equals("mysql")) {

source.setDriverClass("com.mysql.jdbc.Driver");

}

**if** (MC.*db\_type*.equals("postgresql")) {

source.setDriverClass("org.postgresql.Driver");

}

source.setMaxPoolSize(C.*MaxPoolSize*);

source.setMinPoolSize(C.*MinPoolSize*);

source.setInitialPoolSize(C.*InitialPoolSize*);

source.setMaxIdleTime(C.*MaxIdleTime*);

source.setAcquireIncrement(C.*AcquireIncrement*);

String port = "";

**if** (MC.*db\_port*.size() - 1 < i) {

port = "3306";

} **else** {

port = MC.*db\_port*.get(i);

}

source.setJdbcUrl("jdbc:" + MC.*db\_type* + "://" + MC.*db\_host*.get(i) + ":" + port + "/"

+ MC.*db\_name*.get(i) + "??useUnicode=true&characterEncoding=" + C.*default\_encoding*

+ "&zeroDateTimeBehavior=convertToNull");

source.setUser(MC.*db\_username*.get(i));

source.setPassword(MC.*db\_password*.get(i));

MC.*dataSource*.add(source);

}

} **catch** (PropertyVetoException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **static** Connection get\_connection() {

Connection connection = **null**;

**if** (MC.*dataSource* == **null**) {

System.***out***.println("数据源不存在");

} **else** {

**try** {

connection = MC.*dataSource*.get(0).getConnection();

} **catch** (SQLException e) {

}

}

**return** connection;

}

**public** **static** Connection get\_connection(Integer dbIndex) {

Connection connection = **null**;

**if** (MC.*dataSource* == **null**) {

System.***out***.println("数据源不存在");

} **else** {

**try** {

**if** (MC.*dataSource*.size() < dbIndex) {

System.***out***.println("不存在该数据库链接");

} **else** {

connection = MC.*dataSource*.get(dbIndex).getConnection();

}

} **catch** (SQLException e) {

}

}

**return** connection;

}

}

**package** com.javatodo.core.model;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**public** **class** MysqlDriver **extends** Driver {

**private** String table\_pre = "";// 表前缀

**private** String sql = "";

**private** String table\_name = "";

**private** String where\_str = " where 1 ";

**private** List<Object> where\_value\_list = **new** ArrayList<>();

**private** List<Object> data\_value\_list = **new** ArrayList<>();

**private** List<Object> update\_value\_list = **new** ArrayList<>();

**private** String order\_str = "";

**private** String field\_str = " \* ";

**private** String limit\_str = "";

**private** String add\_str = "";

**private** String as\_str = "";

**private** List<String> join\_str = **new** ArrayList<String>();

**private** String group\_str = "";

**public** MysqlDriver() {

**this**.table\_pre = MC.*table\_pre*.get(0);

}

**public** MysqlDriver(Integer dbIndex) {

**this**.table\_pre = MC.*table\_pre*.get(dbIndex);

}

// 初始化当前表

**public** MysqlDriver(String table\_name) {

**this**.table\_pre = MC.*table\_pre*.get(0);

**this**.table\_name = table\_name;

}

// 初始化当前表

**public** MysqlDriver(String table\_name, Integer dbIndex) {

**this**.table\_pre = MC.*table\_pre*.get(dbIndex);

**this**.table\_name = table\_name;

}

// 设置表名称

**public** **void** table(String table\_name) {

**this**.table\_name = table\_name;

}

// where方法

**public** MysqlDriver where(Map<String, W> where) {

**for** (String key : where.keySet()) {

**switch** (where.get(key).get\_relation().toLowerCase().trim()) {

**case** "eq":

**this**.where\_str = **this**.where\_str + " and " + key + " =? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "=":

**this**.where\_str = **this**.where\_str + " and " + key + " =? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "!=":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<>":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "neq":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** ">":

**this**.where\_str = **this**.where\_str + " and " + key + " >? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "gt":

**this**.where\_str = **this**.where\_str + " and " + key + " >? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** ">=":

**this**.where\_str = **this**.where\_str + " and " + key + " >=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "egt":

**this**.where\_str = **this**.where\_str + " and " + key + " >=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "lt":

**this**.where\_str = **this**.where\_str + " and " + key + " <? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "elt":

**this**.where\_str = **this**.where\_str + " and " + key + " <=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<":

**this**.where\_str = **this**.where\_str + " and " + key + " <? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<=":

**this**.where\_str = **this**.where\_str + " and " + key + " <=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "like":

**this**.where\_str = **this**.where\_str + " and " + key + " like ? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "between":

**this**.where\_str = **this**.where\_str + " and " + key + " between ? and ? ";

**if** (where.get(key).get\_value\_list().get(0) != **null** && where.get(key).get\_value\_list().get(1) != **null**) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

} **else** **if** (where.get(key).get\_value\_list().size() == 2) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

}

**break**;

**case** "not between":

**this**.where\_str = **this**.where\_str + " and " + key + " not between ? and ? ";

**if** (where.get(key).get\_value\_list().get(0) != **null** && where.get(key).get\_value\_list().get(1) != **null**) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

} **else** **if** (where.get(key).get\_value\_list().size() == 2) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

}

**break**;

**case** "in":

**if** (where.get(key).get\_value\_list().size() > 1) {

String wenhao\_str = "?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**for** (Integer integer = 1; integer < where.get(key).get\_value\_list().size(); integer = integer + 1) {

wenhao\_str = wenhao\_str + ",?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(integer));

}

**this**.where\_str = **this**.where\_str + " and " + key + " in (" + wenhao\_str + ") ";

} **else** **if** (where.get(key).get\_value\_list().size() == 1) {

**this**.where\_str = **this**.where\_str + " and " + key + "=? ";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

}

**break**;

**case** "not in":

**if** (where.get(key).get\_value\_list().size() > 1) {

String wenhao\_str = "?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**for** (Integer integer = 1; integer < where.get(key).get\_value\_list().size(); integer = integer + 1) {

wenhao\_str = wenhao\_str + ",?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(integer));

}

**this**.where\_str = **this**.where\_str + " and " + key + " not in (" + wenhao\_str + ") ";

} **else** **if** (where.get(key).get\_value\_list().size() == 1) {

**this**.where\_str = **this**.where\_str + " and " + key + "!=? ";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

}

**break**;

**default**:

**break**;

}

}

**return** **this**;

}

// where方法

**public** MysqlDriver where(String where\_str) {

**this**.where\_str = **this**.where\_str + " and " + where\_str + " ";

**return** **this**;

}

// where方法

**public** MysqlDriver where(String where\_str, Object... params) {

**this**.where\_str = **this**.where\_str + " and " + where\_str + " ";

**for** (Integer integer = 0; integer < params.length; integer = integer + 1) {

**this**.where\_value\_list.add(params[integer]);

}

**return** **this**;

}

// order方法

**public** MysqlDriver order(String order\_str) {

**this**.order\_str = " order by " + order\_str;

**return** **this**;

}

// limit方法

**public** MysqlDriver limit(String limit\_str) {

**this**.limit\_str = " limit " + limit\_str + " ";

**return** **this**;

}

// data方法

**public** MysqlDriver data(Map<String, Object> data) {

String key\_str = "";

String value\_str = "";

**for** (String key : data.keySet()) {

**if** (key\_str.equals("")) {

key\_str = key;

value\_str = "?";

**this**.data\_value\_list.add(data.get(key));

} **else** {

key\_str = key\_str + "," + key;

value\_str = value\_str + ",?";

**this**.data\_value\_list.add(data.get(key));

}

**public** **class** PgsqlDriver **extends** Driver {

**private** String table\_pre = "";// 表前缀

**private** String sql = "";

**private** String table\_name = "";

**private** String where\_str = " where true ";

**private** List<Object> where\_value\_list = **new** ArrayList<>();

**private** List<Object> data\_value\_list = **new** ArrayList<>();

**private** List<Object> update\_value\_list = **new** ArrayList<>();

**private** String order\_str = "";

**private** String field\_str = " \* ";

**private** String limit\_str = "";

**private** String add\_str = "";

**private** String as\_str = "";

**private** List<String> join\_str = **new** ArrayList<String>();

**private** String group\_str = "";

**public** PgsqlDriver() {

**this**.table\_pre = MC.*table\_pre*.get(0);

}

**public** PgsqlDriver(Integer dbIndex) {

**this**.table\_pre = MC.*table\_pre*.get(dbIndex);

}

// 初始化当前表

**public** PgsqlDriver(String table\_name) {

**this**.table\_pre = MC.*table\_pre*.get(0);

**this**.table\_name = table\_name;

}

**public** PgsqlDriver(String table\_name, Integer dbIndex) {

**this**.table\_pre = MC.*table\_pre*.get(dbIndex);

**this**.table\_name = table\_name;

}

// 设置表名称

**public** **void** table(String table\_name) {

**this**.table\_name = table\_name;

}

// where方法

**public** PgsqlDriver where(Map<String, W> where) {

**for** (String key : where.keySet()) {

**switch** (where.get(key).get\_relation().toLowerCase().trim()) {

**case** "eq":

**this**.where\_str = **this**.where\_str + " and " + key + " =? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "=":

**this**.where\_str = **this**.where\_str + " and " + key + " =? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "!=":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<>":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "neq":

**this**.where\_str = **this**.where\_str + " and " + key + " !=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** ">":

**this**.where\_str = **this**.where\_str + " and " + key + " >? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "gt":

**this**.where\_str = **this**.where\_str + " and " + key + " >? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** ">=":

**this**.where\_str = **this**.where\_str + " and " + key + " >=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "egt":

**this**.where\_str = **this**.where\_str + " and " + key + " >=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "lt":

**this**.where\_str = **this**.where\_str + " and " + key + " <? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "elt":

**this**.where\_str = **this**.where\_str + " and " + key + " <=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<":

**this**.where\_str = **this**.where\_str + " and " + key + " <? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "<=":

**this**.where\_str = **this**.where\_str + " and " + key + " <=? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "like":

**this**.where\_str = **this**.where\_str + " and " + key + " like ? ";

**this**.where\_value\_list.add(where.get(key).get\_value());

**break**;

**case** "between":

**this**.where\_str = **this**.where\_str + " and " + key + " between ? and ? ";

**if** (where.get(key).get\_value\_list().get(0) != **null** && where.get(key).get\_value\_list().get(1) != **null**) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

} **else** **if** (where.get(key).get\_value\_list().size() == 2) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

}

**break**;

**case** "not between":

**this**.where\_str = **this**.where\_str + " and " + key + " not between ? and ? ";

**if** (where.get(key).get\_value\_list().get(0) != **null** && where.get(key).get\_value\_list().get(1) != **null**) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

} **else** **if** (where.get(key).get\_value\_list().size() == 2) {

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(1));

}

**break**;

**case** "in":

**if** (where.get(key).get\_value\_list().size() > 1) {

String wenhao\_str = "?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**for** (Integer integer = 1; integer < where.get(key).get\_value\_list().size(); integer = integer + 1) {

wenhao\_str = wenhao\_str + ",?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(integer));

}

**this**.where\_str = **this**.where\_str + " and " + key + " in (" + wenhao\_str + ") ";

} **else** **if** (where.get(key).get\_value\_list().size() == 1) {

**this**.where\_str = **this**.where\_str + " and " + key + "=? ";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

}

**break**;

**case** "not in":

**if** (where.get(key).get\_value\_list().size() > 1) {

String wenhao\_str = "?";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

**for** (Integer integer = 1; integer < where.get(key).get\_value\_list().size(); integer = integer + 1) {

wenhao\_str = wenhao\_str + ",?"; **this**.where\_value\_list.add(where.get(key).get\_value\_list().get(integer));

}

**this**.where\_str = **this**.where\_str + " and " + key + " not in (" + wenhao\_str + ") ";

} **else** **if** (where.get(key).get\_value\_list().size() == 1) {

**this**.where\_str = **this**.where\_str + " and " + key + "!=? ";

**this**.where\_value\_list.add(where.get(key).get\_value\_list().get(0));

}

**break**;

**default**:

**break**;

}

}

**return** **this**;

}

// where方法

**public** PgsqlDriver where(String where\_str) {

**this**.where\_str = **this**.where\_str + " and " + where\_str + " ";

**return** **this**;

}

// where方法

**public** PgsqlDriver where(String where\_str, Object... params) {

**this**.where\_str = **this**.where\_str + " and " + where\_str + " ";

**for** (Integer integer = 0; integer < params.length; integer = integer + 1) {

**this**.where\_value\_list.add(params[integer]);

}

**return** **this**;

}

// order方法

**public** PgsqlDriver order(String order\_str) {

**this**.order\_str = " order by " + order\_str;

**return** **this**;

}

// limit方法

**public** PgsqlDriver limit(String limit\_str) {

String[] limit\_arr = **new** String[2];

**if** (limit\_str.contains(",")) {

limit\_arr = limit\_str.split(",");

}

**this**.limit\_str = " limit " + limit\_arr[1] + " offset " + limit\_arr[0] + " ";

**return** **this**;

}

// data方法

**public** PgsqlDriver data(Map<String, Object> data) {

String key\_str = "";

String value\_str = "";

**for** (String key : data.keySet()) {

**if** (key\_str.equals("")) {

key\_str = key;

value\_str = "?";

**this**.data\_value\_list.add(data.get(key));

} **else** {

key\_str = key\_str + "," + key;

value\_str = value\_str + ",?";

**this**.data\_value\_list.add(data.get(key));

}

}

key\_str = "(" + key\_str + ")";

value\_str = "(" + value\_str + ")";

**this**.add\_str = " " + key\_str + " values " + value\_str + " ";

**return** **this**;

}

// alias方法

**public** PgsqlDriver alias(String as\_str) {

**this**.as\_str = " as " + as\_str + " ";

**return** **this**;

}

// join

**public** PgsqlDriver join(String table\_name, String on\_sql) {

table\_name = **this**.table\_pre + table\_name;

String sql;

sql = " inner join " + table\_name + " " + on\_sql + " ";

**this**.join\_str.add(sql);

**return** **this**;

}

// join

**public** PgsqlDriver join(String table\_name, String on\_sql, String type) {

table\_name = **this**.table\_pre + table\_name;

String sql;

sql = " " + type + " join " + table\_name + " " + on\_sql + " ";

**this**.join\_str.add(sql);

**return** **this**;

}

// filed方法

**public** PgsqlDriver field(String field\_str) {

**this**.field\_str = " " + field\_str + " ";

**return** **this**;

}

// add方法

**public** PgsqlDriver add() {

**this**.sql = "insert into " + **this**.table\_pre + **this**.table\_name + " " + **this**.add\_str + ";";

**return** **this**;

}

// save方法

**public** PgsqlDriver save(Map<String, Object> data) {

String key\_str = "";

String value\_str = "";

String str = "";

**for** (String key : data.keySet()) {

**if** (key\_str == "") {

key\_str = key;

value\_str = "?";

**this**.update\_value\_list.add(data.get(key));

str = " " + key\_str + "=" + value\_str + " ";

} **else** {

key\_str = key;

value\_str = "?";

**this**.update\_value\_list.add(data.get(key));

str = str + "," + key\_str + "=" + value\_str + " ";

}

}

str = " set " + str;

**this**.sql = "update " + **this**.table\_pre + **this**.table\_name + str + **this**.where\_str + ";";

**return** **this**;

}

// delete方法

**public** PgsqlDriver delete() {

**this**.sql = "delete from " + **this**.table\_pre + **this**.table\_name + **this**.where\_str + ";";

**return** **this**;

}

// select方法

**public** PgsqlDriver select() {

String join\_sql = "";

**int** i = 0;

**while** (i < **this**.join\_str.size()) {

join\_sql = join\_sql + **this**.join\_str.get(i);

i = i + 1;

}

**this**.sql = "select " + **this**.field\_str + " from " + **this**.table\_pre + **this**.table\_name + **this**.as\_str + join\_sql + **this**.where\_str + **this**.group\_str + **this**.order\_str + **this**.limit\_str + ";";

**return** **this**;

}

// find方法

**public** PgsqlDriver find() {

String join\_sql = "";

**int** i = 0;

**while** (i < **this**.join\_str.size()) {

join\_sql = join\_sql + **this**.join\_str.get(i);

i = i + 1;

}

**this**.sql = "select " + **this**.field\_str + " from " + **this**.table\_pre + **this**.table\_name + **this**.as\_str + join\_sql + **this**.where\_str + **this**.group\_str + **this**.order\_str + " limit 1;";

**return** **this**;

}

// 获取sql

**public** String get\_sql() {

**return** **this**.sql;

}

// 获取add参数

**public** List<Object> get\_add\_data() {

**return** **this**.data\_value\_list;

}

// 获取update参数

**public** List<Object> get\_update\_data() {

**return** **this**.update\_value\_list;

}

// 获取where参数

**public** List<Object> get\_where\_data() {

**return** **this**.where\_value\_list;

}

// group方法

**public** **void** group(String fields) {

**this**.group\_str = " group by " + fields + " ";

}

**public** PgsqlDriver setInc(String field, Integer value) {

String str = "";

str = " set " + field + "=" + field + "+" + value;

**this**.sql = "update `" + **this**.table\_pre + **this**.table\_name + "`" + str + **this**.where\_str + ";";

**return** **this**;

}

**public** PgsqlDriver setDec(String field, Integer value) {

String str = "";

str = " set " + field + "=" + field + "-" + value;

**this**.sql = "update `" + **this**.table\_pre + **this**.table\_name + "`" + str + **this**.where\_str + ";";

**return** **this**;

}

**public** PgsqlDriver setInc(String field) {

String str = "";

str = " set " + field + "=" + field + "+1";

**this**.sql = "update `" + **this**.table\_pre + **this**.table\_name + "`" + str + **this**.where\_str + ";";

**return** **this**;

}

**public** PgsqlDriver setDec(String field) {

String str = "";

str = " set " + field + "=" + field + "-1";

**this**.sql = "update `" + **this**.table\_pre + **this**.table\_name + "`" + str + **this**.where\_str + ";";

**return** **this**;

}

// 清理数据

**public** **void** clear() {

// this.table\_pre = "";

**this**.sql = "";

**this**.where\_str = " where true ";

**this**.where\_value\_list = **new** ArrayList<>();

**this**.data\_value\_list = **new** ArrayList<>();

**this**.update\_value\_list = **new** ArrayList<>();

**this**.order\_str = "";

**this**.field\_str = " \* ";

**this**.limit\_str = "";

**this**.add\_str = "";

**this**.as\_str = "";

**this**.group\_str = "";

**this**.join\_str = **new** ArrayList<String>();

}

}

**package** com.javatodo.core.model;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** W {

**private** String relation = "";

**private** Object value = **new** Object();

**private** List<Object> value\_list = **new** ArrayList<>();

@SuppressWarnings("unchecked")

**public** W(String relation, Object value) {

**if** ((!value.getClass().isArray())&&(!value.getClass().getName().toLowerCase().contains("list"))) {

**this**.relation = relation;

**this**.value = value;

}**else**{

**if**(value.getClass().getName().toLowerCase().contains("list")){

**this**.relation = relation;

**this**.value\_list = (ArrayList<Object>)value;

}

}

}

**public** W(String relation, Object[] values) {

**this**.relation = relation;

**this**.value\_list = **new** ArrayList<>();

**for** (Integer i = 0; i < values.length; i = i + 1) {

**this**.value\_list.add(values[i]);

}

}

**public** String get\_relation() {

**return** **this**.relation;

}

**public** Object get\_value() {

**return** **this**.value;

}

**public** List<Object> get\_value\_list() {

**return** **this**.value\_list;

}

}

**package** com.javatodo.core.router;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** RC {

**private** **static** Map<String, String> *C* = **new** HashMap<>();

**public** RC(String package\_name, String module\_name) {

**if** (!("".equals(package\_name.trim()) || "".equals(module\_name.trim()))) {

RC.*C*.put(module\_name, package\_name);

}

}

**public** RC(String package\_name, String module\_name, String class\_name, String controller\_name) {

**if** (!("".equals(package\_name.trim()) || "".equals(module\_name.trim()) || "".equals(class\_name.trim())

|| "".equals(controller\_name.trim()))) {

RC.*C*.put(module\_name, package\_name);

RC.*C*.put(module\_name + "!--javatodo--!" + controller\_name, class\_name);

}

}

**public** RC(String package\_name, String module\_name, String class\_name, String controller\_name, String function\_name,

String action\_name) {

**if** (!("".equals(package\_name.trim()) || "".equals(module\_name.trim()) || "".equals(class\_name.trim())

|| "".equals(controller\_name.trim()) || "".equals(function\_name.trim()) || "".equals(action\_name))) {

RC.*C*.put(module\_name, package\_name);

RC.*C*.put(module\_name + "!--javatodo--!" + controller\_name, class\_name);

RC.*C*.put(module\_name + "!--javatodo--!" + controller\_name + "!--javatodo--!" + action\_name, function\_name);

}

}

**public** **static** String getRC(String module\_name) {

**if** (RC.*C*.containsKey(module\_name)) {

**return** RC.*C*.get(module\_name);

} **else** {

**return** module\_name;

}

}

**public** **static** String getRC(String module\_name, String controller\_name) {

**if** (RC.*C*.containsKey(module\_name + "!--javatodo--!" + controller\_name)) {

**return** RC.*C*.get(module\_name + "!--javatodo--!" + controller\_name);

} **else** {

**return** controller\_name;

}

}

**public** **static** String getRC(String module\_name, String controller\_name, String action\_name) {

**if** (RC.*C*.containsKey(module\_name + "!--javatodo--!" + controller\_name + "!--javatodo--!" + action\_name)) {

**return** RC.*C*.get(module\_name + "!--javatodo--!" + controller\_name + "!--javatodo--!" + action\_name);

} **else** {

**return** action\_name;

}

}

**public** **static** **void** clear() {

RC.*C*.clear();

}

}

**package** com.javatodo.core.router;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** com.javatodo.config.C;

**public** **class** Router {

HttpServletRequest request=**null**;

**public** String MODULE\_NAME=C.*default\_module*;

**public** String CONTROLLER\_NAME=C.*default\_controller*;

**public** String ACTION\_NAME=C.*default\_action*;

**public** String PACKAGE\_NAME=RC.*getRC*(C.*default\_module*);

**public** String CLASS\_NAME=RC.*getRC*(C.*default\_module*, C.*default\_controller*);

**public** String FUNCTION\_NAME=RC.*getRC*(C.*default\_module*, C.*default\_controller*, C.*default\_action*);

**public** Router(HttpServletRequest request){

**this**.request=request;

}

**public** Map<String, String> parse(){

Map<String, String>map=**new** HashMap<>();

Map<String, String[]>queryMap=request.getParameterMap();

**for**(Map.Entry<String, String[]> entry:queryMap.entrySet()){

String[] value\_arr=entry.getValue();

**if**(value\_arr==**null**){

**continue**;

}**else**{

**if**(value\_arr.length==0){

**continue**;

}

}

map.put(entry.getKey(), value\_arr[0]);

}

**if**(!map.containsKey("m")) {

map.put("m", **this**.MODULE\_NAME);

}**else** {

**this**.MODULE\_NAME = map.get("m");

**this**.PACKAGE\_NAME = RC.*getRC*(map.get("m"));

}

**if**(!map.containsKey("c")) {

map.put("c", **this**.CONTROLLER\_NAME);

}**else** {

**this**.CONTROLLER\_NAME = map.get("c");

**this**.CLASS\_NAME = RC.*getRC*(map.get("m"), map.get("c"));

}

**if**(!map.containsKey("a")) {

map.put("a", **this**.ACTION\_NAME);

}**else** {

**this**.ACTION\_NAME = map.get("a");

**this**.FUNCTION\_NAME = RC.*getRC*(map.get("m"), map.get("c"), map.get("a"));

}

**return** map;

}

}

**package** com.javatodo.core.tools;

**import** java.io.ByteArrayOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**import** java.util.Map;

**public** **class** Http {

**public** Integer code = **null**;

**public** String html = **null**;

**public** String get(String url) {

**try** {

URL httpurl = **new** URL(url);

HttpURLConnection conn = (HttpURLConnection) httpurl.openConnection();

conn.setRequestMethod("GET");

conn.setConnectTimeout(10 \* 1000);

**this**.code = conn.getResponseCode();

**if** (code == 200) {

InputStream inputStream = conn.getInputStream();

**this**.html = **this**.stremToString(inputStream, "utf-8");

**return** html;

} **else** {

**return** **null**;

}

} **catch** (Exception e) {

// **TODO**: handle exception

**return** **null**;

}

}

**public** String get(String url, Map<String, String> header) {

**try** {

URL httpurl = **new** URL(url);

HttpURLConnection conn = (HttpURLConnection) httpurl.openConnection();

conn.setRequestMethod("GET");

conn.setConnectTimeout(10 \* 1000);

**for** (String key : header.keySet()) {

conn.setRequestProperty(key, header.get(key));

}

**this**.code = conn.getResponseCode();

**if** (code == 200) {

InputStream inputStream = conn.getInputStream();

**this**.html = **this**.stremToString(inputStream, "utf-8");

**return** html;

} **else** {

**return** **null**;

}

} **catch** (Exception e) {

// **TODO**: handle exception

**return** **null**;

}

}

**public** String post(String url, String param) {

**try** {

URL httpurl = **new** URL(url);

HttpURLConnection conn = (HttpURLConnection) httpurl.openConnection();

conn.setRequestMethod("POST");

conn.setReadTimeout(5000);

conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");

String data = param;

conn.setRequestProperty("Content-Length", String.*valueOf*(data.getBytes().length));

conn.setDoOutput(**true**);

conn.getOutputStream().write(data.getBytes("UTF-8"));

**this**.code = conn.getResponseCode();

**if** (code == 200) {

InputStream inputStream = conn.getInputStream();

**this**.html = **this**.stremToString(inputStream, "UTF-8");

**return** **this**.html;

} **else** {

**return** **null**;

}

} **catch** (Exception e) {

e.printStackTrace();

**return** **null**;

}

}

**public** String post(String url, Map<String, String> header, String param) {

**try** {

URL httpurl = **new** URL(url);

HttpURLConnection conn = (HttpURLConnection) httpurl.openConnection();

conn.setRequestMethod("POST");

conn.setReadTimeout(5000);

conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");

**for** (String key : header.keySet()) {

conn.setRequestProperty(key, header.get(key));

}

String data = param;

conn.setRequestProperty("Content-Length", String.*valueOf*(data.getBytes().length));

conn.setDoOutput(**true**);

conn.getOutputStream().write(data.getBytes("UTF-8"));

**this**.code = conn.getResponseCode();

**if** (code == 200) {

InputStream inputStream = conn.getInputStream();

**this**.html = **this**.stremToString(inputStream, "UTF-8");

**return** **this**.html;

} **else** {

**return** **null**;

}

} **catch** (Exception e) {

e.printStackTrace();

**return** **null**;

}

}

**private** String stremToString(InputStream is, String encoding) **throws** IOException {

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

**int** i = -1;

**while** ((i = is.read()) != -1) {

baos.write(i);

}

**return** baos.toString(encoding);

}

}

**package** com.javatodo.core.tools;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** Page {

**public** String entrance = "";

**public** **int** total = 0;// 总条数

**public** **int** listRows = 0;// 每页条数

**public** **int** current = 1;// 当前页码

**public** **int** firstRow = 0;// 起始条数

**public** **int** allPageNum = 0;// 总页数

**public** Map<String, String> map = **new** HashMap<>();

**public** Page(**int** total, **int** num, String entrance, Map<String, String> routerMap) {

**this**.total = total;

**this**.listRows = num;

**if** (routerMap.containsKey("p")) {

current = T.*toInt*(routerMap.get("p").toString());

**if** (current == 0) {

current = 1;

}

} **else** {

current = 1;

}

**this**.firstRow = (current - 1) \* listRows;

**for** (Map.Entry<String, String> entry : routerMap.entrySet()) {

**this**.map.put(entry.getKey().toString(), entry.getValue().toString());

}

**this**.entrance = entrance;

}

/\*\*

\* 生成分页的html代码

\*

\* **@return** String 分页的html代码

\*/

**public** String show() {

**double** zhi = (**double**) total / (**double**) listRows;

allPageNum = (**int**) Math.*ceil*(zhi);

String page\_str = "<span>共" + allPageNum + "页，每页" + listRows + "条，当前第" + **this**.current + "页</span>";

**if** (current == 1) {

map.put("p", (**this**.current + 1) + "");

page\_str = page\_str + "<span>首页</span><span>上一页</span><a href='" + T.*U*(map, entrance) + "' >下一页</a>";

map.put("p", allPageNum + "");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >尾页</a>";

} **else** **if** (current == allPageNum) {

map.put("p", "1");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "'>首页</a>";

map.put("p", (**this**.current - 1) + "");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >上一页</a><span>下一页</span><span>尾页</span>";

} **else** **if** (current < 1) {

page\_str = "";

} **else** **if** (current > allPageNum) {

page\_str = "";

} **else** {

map.put("p", "1");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >首页</a>";

map.put("p", (**this**.current - 1) + "");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >上一页</a>";

map.put("p", (**this**.current + 1) + "");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >下一页</a>";

map.put("p", allPageNum + "");

page\_str = page\_str + "<a href='" + T.*U*(map, entrance) + "' >尾页</a>";

}

String gotopage = "<span>转到：<select onchange=\"self.location.href=this.options[this.selectedIndex].value\">";

**for** (Integer i = 0; i < allPageNum; i = i + 1) {

map.put("p", (i + 1) + "");

String selected = "";

**if** (**this**.current == i + 1) {

selected = "selected=\"selected\"";

} **else** {

selected = "";

}

gotopage = gotopage + "<option " + selected + " value=\"" + T.*U*(map, entrance) + "\">第 " + (i + 1) + " 页</option>";

}

gotopage = gotopage + "</select></span>";

**if** (zhi < 1) {

**return** "";

} **else** {

**return** "<div>" + page\_str + gotopage + "</div>";

}

}

}

**package** com.javatodo.core.tools;

**import** com.alibaba.fastjson.JSONObject;

**public** **class** Wechat {

**private** Http http = **new** Http();

**public** String errorMsg = "";

**public** String getAccessToken(String appid, String appsecret) {

String url = "https://api.weixin.qq.com/cgi-bin/token?grant\_type=client\_credential&appid=" + appid + "&secret="

+ appsecret;

String ret = **this**.http.get(url);

**if** (ret == **null**) {

**this**.errorMsg = "网络连接错误";

**return** **null**;

} **else** {

JSONObject jsonObject = (JSONObject) JSONObject.*parse*(ret);

**if** (jsonObject.containsKey("access\_token")) {

**return** jsonObject.getString("access\_token");

} **else** {

**this**.errorMsg = jsonObject.getString("errmsg");

**return** **null**;

}

}

}

}

**package** com.javatodo.core.view;

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.io.StringWriter;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** com.javatodo.config.C;

**import** freemarker.template.Configuration;

**import** freemarker.template.ObjectWrapper;

**import** freemarker.template.Template;

**import** freemarker.template.TemplateException;

**import** freemarker.template.TemplateExceptionHandler;

**public** **class** FreeMakerView **extends** View {

**private** Configuration configuration = **new** Configuration();

**private** Map<String, Object>data=**new** HashMap<>();

**public** FreeMakerView(){

configuration.setTemplateExceptionHandler(TemplateExceptionHandler.***RETHROW\_HANDLER***);

configuration.setObjectWrapper(ObjectWrapper.***BEANS\_WRAPPER***);

configuration.setDefaultEncoding(C.*default\_encoding*);

configuration.setOutputEncoding(C.*default\_encoding*);

configuration.setLocalizedLookup(**false**);

configuration.setNumberFormat("#0.#####");

configuration.setDateFormat("yyyy-MM-dd");

configuration.setTimeFormat("HH:mm:ss");

configuration.setDateTimeFormat("yyyy-MM-dd HH:mm:ss");

}

@Override

**public** **void** assign(String name, Object value) {

// **TODO** Auto-generated method stub

**this**.data.put(name, value);

}

@Override

**public** **void** flush(HttpServletRequest request,HttpServletResponse response,HttpServlet servlet,String view\_path) **throws** IOException, TemplateException{

view\_path=view\_path.replace ("/", "\\");

String[] path\_arr=view\_path.split("\\\\");

String template\_file=path\_arr[path\_arr.length-1];

String template\_floder="";

**for**(Integer integer=0; integer<path\_arr.length-1; integer=integer+1){

**if**(integer==0){

template\_floder=path\_arr[0];

}**else**{

template\_floder=template\_floder+"\\"+path\_arr[integer];

}

}

configuration.setDirectoryForTemplateLoading(**new** File(template\_floder));

Template template=configuration.getTemplate(template\_file);

response.setContentType(request.getContentType());

PrintWriter writer=response.getWriter();

template.process(**this**.data, writer);

}

@Override

**public** String parseString(String view\_path,String logName) **throws** IOException, TemplateException{

view\_path=view\_path.replace ("/", "\\");

String[] path\_arr=view\_path.split("\\\\");

String template\_file=path\_arr[path\_arr.length-1];

String template\_floder="";

**for**(Integer integer=0; integer<path\_arr.length-1; integer=integer+1){

**if**(integer==0){

template\_floder=path\_arr[0];

}**else**{

template\_floder=template\_floder+"\\"+path\_arr[integer];

}

}

configuration.setDirectoryForTemplateLoading(**new** File(template\_floder));

StringWriter writer=**new** StringWriter();

Template template=configuration.getTemplate(template\_file);

template.process(**this**.data, writer);

**return** writer.toString();

}

}

**package** com.javatodo.core.view;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** JspView **extends** View {

**private** HttpServletRequest request=**null**;

**public** JspView(HttpServletRequest request) {

**this**.request=request;

}

@Override

**public** **void** assign(String name, Object value) {

// **TODO** Auto-generated method stub

**this**.request.setAttribute(name, value);

}

@Override

**public** **void** flush(HttpServletRequest request,HttpServletResponse response,HttpServlet servlet,String view\_path) **throws** IOException, ServletException{

**this**.request.getRequestDispatcher(view\_path).forward(**this**.request, response);

}

@Override

**public** String parseString(String view\_path,String logName){

**return** "";

}

}

**package** com.javatodo.core.view;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.io.StringWriter;

**import** java.util.Properties;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.velocity.Template;

**import** org.apache.velocity.VelocityContext;

**import** org.apache.velocity.app.Velocity;

**import** com.javatodo.config.C;

**import** com.javatodo.core.tools.T;

**public** **class** VelocityView **extends** View {

**public** VelocityContext context = **new** VelocityContext();

**private** **static** **final** Properties ***properties*** = **new** Properties();

**private** **static** **boolean** *is\_init* = **false**;

**private** **static** **void** init() {

***properties***.setProperty(Velocity.***FILE\_RESOURCE\_LOADER\_PATH***, "");

***properties***.setProperty(Velocity.***ENCODING\_DEFAULT***, C.*default\_encoding*);

***properties***.setProperty(Velocity.***INPUT\_ENCODING***, C.*default\_encoding*);

***properties***.setProperty(Velocity.***OUTPUT\_ENCODING***, C.*default\_encoding*);

VelocityView.*is\_init* = **true**;

}

**public** VelocityView() {

// **TODO** Auto-generated constructor stub

**if** (!VelocityView.*is\_init*) {

VelocityView.*init*();

Velocity.*init*(***properties***);

}

}

@Override

**public** **void** assign(String name, Object value) {

// **TODO** Auto-generated method stub

**this**.context.put(name, value);

}

@Override

**public** **void** flush(HttpServletRequest request, HttpServletResponse response, HttpServlet servlet, String view\_path) **throws** IOException {

Template template = Velocity.*getTemplate*(view\_path);

response.setContentType(request.getContentType());

PrintWriter writer = response.getWriter();

template.merge(context, writer);

writer.flush();

}

@Override

**public** String parseString(String view\_path, String logName) **throws** IOException {

StringWriter stringWriter = **new** StringWriter();

String content = T.*readFile*(view\_path, C.*default\_encoding*);

Velocity.*evaluate*(context, stringWriter, logName, content);

**return** stringWriter.getBuffer().toString();

}

}

**package** com.javatodo.core.view;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** freemarker.template.TemplateException;

**public** **abstract** **class** View {

**public** **abstract** **void** assign(String name,Object value);

**public** **abstract** **void** flush(HttpServletRequest request,HttpServletResponse response,HttpServlet servlet,String view\_path) **throws** IOException, ServletException, TemplateException;

**public** **abstract** String parseString(String view\_path,String logName) **throws** IOException, TemplateException;

}

**package** com.javatodo.core.tools;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader;

**import** java.math.BigDecimal;

**import** java.security.MessageDigest;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Properties;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**import** javax.servlet.http.HttpServletRequest;

**import** com.alibaba.fastjson.JSON;

**import** com.alibaba.fastjson.JSONObject;

**import** com.javatodo.config.C;

**public** **class** T {

**public** **static** String now() {

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

**return** df.format(**new** Date());

}

**public** **static** String now(String format) {

SimpleDateFormat df = **new** SimpleDateFormat(format);

**return** df.format(**new** Date());

}

**public** **static** Integer strtotime(String date, String format) **throws** ParseException {

**if** (format.trim().equals("")) {

format = "yyyy-MM-dd HH:mm:ss";

}

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(format);

Date dateFormat = simpleDateFormat.parse(date);

**long** ts = dateFormat.getTime();

Integer timestamp = (**int**) (ts / 1000);

**return** timestamp;

}

**public** **static** String date(String format, Integer time) {

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(format);

String timestamp = time.toString() + "000";

Date date = **new** Date(Long.*valueOf*(timestamp));

**return** simpleDateFormat.format(date);

}

**public** **static** String date(String format, Long time) {

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(format);

Date date = **new** Date(time);

**return** simpleDateFormat.format(date);

}

**public** **static** Integer time() {

**long** ts = System.*currentTimeMillis*();

Integer timestamp = (**int**) (ts / 1000);

**return** timestamp;

}

**public** **static** String htmlspecialchars(String string) {

string = string.replaceAll("&", "&amp;");

string = string.replaceAll("<", "&lt;");

string = string.replaceAll(">", "&gt;");

string = string.replaceAll("\"", "&quot;");

string = string.replaceAll("'", "&apos;");

**return** string;

}

**public** **static** String htmlspecialchars\_decode(String string) {

string = string.replaceAll("&amp;", "&");

string = string.replaceAll("&lt;", "<");

string = string.replaceAll("&gt;", ">");

string = string.replaceAll("&quot;", "\"");

string = string.replaceAll("&apos;", "'");

**return** string;

}

**public** **static** **boolean** detect(Object value, String type) {

**boolean** ret = **false**;

**if** (value == **null**) {

**return** ret;

}

String regExp = "";

Pattern p = **null**;

Matcher m = **null**;

**switch** (type) {

**case** "mobile":

regExp = "^((13[0-9]|15[0-9]|17[0-9]|18[0-9])+\\d{8})$";

**break**;

**case** "email":

regExp = "^([a-z0-9A-Z]+[-|\_|\\.]?)+[a-z0-9A-Z]@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-zA-Z]{2,}$";

**break**;

**case** "require":

**if** (value.toString().length() > 0) {

**return** **true**;

} **else** {

**return** **false**;

}

**default**:

regExp = type;

**break**;

}

**if** (value.toString().length() == 0) {

**return** **false**;

}

**if** (regExp.length() > 0) {

p = Pattern.*compile*(regExp);

m = p.matcher(value.toString());

ret = m.find();

}

**return** ret;

}

**public** **static** String md5(String string) {

**char** hexDigits[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };

**try** {

**byte**[] strTemp = string.getBytes("utf-8");

// 使用MD5创建MessageDigest对象

MessageDigest mdTemp = MessageDigest.*getInstance*("MD5");

mdTemp.update(strTemp);

**byte**[] md = mdTemp.digest();

**int** j = md.length;

**char** str[] = **new** **char**[j \* 2];

**int** k = 0;

**for** (**int** i = 0; i < j; i++) {

**byte** b = md[i];

str[k++] = hexDigits[b >> 4 & 0xf];

str[k++] = hexDigits[b & 0xf];

}

**return** **new** String(str);

} **catch** (Exception e) {

**return** **null**;

}

}

**public** **static** String U(String path, String entrance) {

String url = "./" + entrance;

String[] paths = path.split("/");

**if** (paths.length == 3) {

url = url + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

**return** url;

}

**public** **static** String U(Map<String, String> map, String entrance) {

String url = "./" + entrance;

Integer i = 0;

**for** (Entry<String, String> entry : map.entrySet()) {

**if** (i == 0) {

url = url + "?" + entry.getKey() + "=" + entry.getValue().toString();

} **else** {

url = url + "&" + entry.getKey() + "=" + entry.getValue().toString();

}

i = i + 1;

}

**return** url;

}

**public** **static** String UJ(String path, String param, String entrance) {

String url = "./" + entrance;

Map<String, Object> map = JSON.*parseObject*(param);

String[] paths = path.split("/");

**if** (paths.length == 3) {

url = url + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

**for** (Entry<String, Object> entry : map.entrySet()) {

url = url + "&" + entry.getKey() + "=" + entry.getValue().toString();

}

**return** url;

}

**public** **static** String U(String path, String param, String entrance) {

String url = "./" + entrance;

String[] paths = path.split("/");

**if** (paths.length == 3) {

url = url + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

url = url + "&" + param;

**return** url;

}

**public** **static** String U(String path, Map<String, Object> map, String entrance) {

String url = "./" + entrance;

String[] paths = path.split("/");

**if** (paths.length == 3) {

url = url + "?m=" + paths[0] + "&c=" + paths[1] + "&a=" + paths[2];

}

**for** (Entry<String, Object> entry : map.entrySet()) {

url = url + "&" + entry.getKey() + "=" + entry.getValue().toString();

}

**return** url;

}

**public** **static** Integer toInt(String string) {

Integer integer = 0;

**try** {

integer = Integer.*parseInt*(string.trim());

} **catch** (Exception e) {

integer = 0;

}

**return** integer;

}

**public** **static** String toString(Object object) {

**if** (object == **null**) {

**return** "";

} **else** {

**return** object.toString();

}

}

@SuppressWarnings("unchecked")

**public** **static** Map<String, Object> toMap(Object object) {

Map<String, Object> map = **new** HashMap<>();

**if** (object == **null**) {

**return** map;

} **else** {

**return** (Map<String, Object>) object;

}

}

**public** **static** String json\_encode(Object object) {

String json = JSON.*toJSONString*(object);

**return** json;

}

**public** **static** List<String> str\_to\_list(String string, String regex) {

String[] strings = string.split(regex);

List<String> list = **new** ArrayList<>();

**for** (Integer i = 0; i < strings.length; i = i + 1) {

list.add(strings[i]);

}

**return** list;

}

**public** **static** String readFile(String filePath, String charset) {

**try** {

String pathname = filePath; // 绝对路径或相对路径都可以，这里是绝对路径，写入文件时演示相对路径

File filename = **new** File(pathname); // 要读取以上路径的input。txt文件

InputStreamReader reader = **new** InputStreamReader(**new** FileInputStream(filename), charset); // 建立一个输入流对象reader

BufferedReader br = **new** BufferedReader(reader); // 建立一个对象，它把文件内容转成计算机能读懂的语言

String content = "";

String line = "";

line = br.readLine();

**if** (line != **null**) {

content = content + line;

}

**while** (line != **null**) {

line = br.readLine(); // 一次读入一行数据

**if** (line != **null**) {

content = content + line;

}

}

**return** content;

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

**return** "";

}

**public** **static** **void** coverFile(String file\_path, String content) {

FileOutputStream fop = **null**;

**try** {

File file = **new** File(file\_path);

**if** (!file.getParentFile().isDirectory()) {

**new** File(file.getParent()).mkdirs();

}

**if** (!file.exists()) {

file.createNewFile();

}

fop = **new** FileOutputStream(file);

**if** (!file.exists()) {

file.createNewFile();

}

**byte**[] contentInBytes = content.getBytes();

fop.write(contentInBytes);

fop.flush();

fop.close();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (fop != **null**) {

fop.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**public** **static** **void** writeFile(String file\_path, String content) {

FileOutputStream fop = **null**;

**try** {

File file = **new** File(file\_path);

**if** (!file.getParentFile().isDirectory()) {

**new** File(file.getParent()).mkdirs();

}

**if** (!file.exists()) {

file.createNewFile();

}

fop = **new** FileOutputStream(file, **true**);

**if** (!file.exists()) {

file.createNewFile();

}

**byte**[] contentInBytes = content.getBytes();

fop.write(contentInBytes);

fop.flush();

fop.close();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (fop != **null**) {

fop.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**public** **static** **void** create\_log(String log\_file, String content) {

*writeFile*(C.*log\_file\_path* + log\_file, content + "--------" + T.*now*() + "\n\r");

}

**public** **static** **void** deleteDir(String dirPath) {

File file = **new** File(dirPath);

**if** (file.isFile()) {

file.delete();

} **else** {

File[] files = file.listFiles();

**if** (files == **null**) {

file.delete();

} **else** {

**for** (**int** i = 0; i < files.length; i++) {

*deleteDir*(files[i].getAbsolutePath());

}

file.delete();

}

}

}

**public** **static** **void** deleteFile(String filePath) {

File file = **new** File(filePath);

**if** (file.isFile()) {

file.delete();

} **else** {

*deleteDir*(filePath);

}

}

**public** **static** String getClientIp(HttpServletRequest request) {

String ip = request.getHeader("x-forwarded-for");

**if** (ip == **null** || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("Proxy-Client-IP");

}

**if** (ip == **null** || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {

ip = request.getHeader("WL-Proxy-Client-IP");

}

**if** (ip == **null** || ip.length() == 0 || "unknown".equalsIgnoreCase(ip)) {

ip = request.getRemoteAddr();

}

**return** ip;

}

**public** **static** String getPriKey() {

String[] arr = **new** String[] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z" };

BigDecimal seed = **new** BigDecimal(System.*currentTimeMillis*());

BigDecimal hexadecimal = **new** BigDecimal(36);

List<String> retList = **new** ArrayList<>();

**while** (seed.compareTo(hexadecimal) > 0) {

BigDecimal[] res = seed.divideAndRemainder(hexadecimal);

seed = res[0];

retList.add(0, arr[res[1].intValue()]);

}

retList.add(0, arr[seed.intValue()]);

String ret = "";

**for** (Integer i = 0; i < retList.size(); i = i + 1) {

ret = ret + retList.get(i).toString();

}

**return** ret;

}

**public** **static** String int36Hash(Integer n) {

String[] arr = **new** String[] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z" };

BigDecimal seed = **new** BigDecimal(n);

BigDecimal hexadecimal = **new** BigDecimal(36);

List<String> retList = **new** ArrayList<>();

**while** (seed.compareTo(hexadecimal) > 0) {

BigDecimal[] res = seed.divideAndRemainder(hexadecimal);

seed = res[0];

retList.add(0, arr[res[1].intValue()]);

}

retList.add(0, arr[seed.intValue()]);

String ret = "";

**for** (Integer i = 0; i < retList.size(); i = i + 1) {

ret = ret + retList.get(i).toString();

}

**return** ret;

}

**public** **static** String getString(String key, Map<String, Object> info) {

**if** (info == **null**) {

**return** "";

}

**if** (info.containsKey(key)) {

**return** info.get(key).toString();

} **else** {

**return** "";

}

}

**public** **static** String getString(String key, JSONObject info) {

**if** (info == **null**) {

**return** "";

}

**if** (info.containsKey(key)) {

**return** info.getString(key);

} **else** {

**return** "";

}

}

**public** **static** String getProperties(String PropertiesFile, String Key) {

String value = "";

**try** {

Properties properties = **new** Properties();

InputStream in = **new** FileInputStream(PropertiesFile);

properties.load(in);

value = properties.getProperty(Key);

in.close();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** value.trim();

}

**public** **static** String getDefaultProperties(String Key) {

**return** T.*getProperties*("config.properties", Key);

}

}

**package** com.javatodo.core.tools;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.security.InvalidKeyException;

**import** java.security.Key;

**import** java.security.KeyFactory;

**import** java.security.KeyPair;

**import** java.security.KeyPairGenerator;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.security.spec.PKCS8EncodedKeySpec;

**import** java.security.spec.X509EncodedKeySpec;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.crypto.BadPaddingException;

**import** javax.crypto.Cipher;

**import** javax.crypto.IllegalBlockSizeException;

**import** javax.crypto.NoSuchPaddingException;

**import** sun.misc.BASE64Decoder;

**import** sun.misc.BASE64Encoder;

**public** **class** RSATools {

**private** **static** Cipher *cipher*;

**static** {

**try** {

*cipher* = Cipher.*getInstance*("RSA");

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (NoSuchPaddingException e) {

e.printStackTrace();

}

}

**public** **static** Map<String, String> createGenerateKeyPair() {

**try** {

KeyPairGenerator keyPairGen = KeyPairGenerator.*getInstance*("RSA");

keyPairGen.initialize(2048);

KeyPair keyPair = keyPairGen.generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

String publicKeyString = *getKeyString*(publicKey);

String privateKeyString = *getKeyString*(privateKey);

Map<String, String> map = **new** HashMap<String, String>();

map.put("publicKey", publicKeyString);

map.put("privateKey", privateKeyString);

**return** map;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** Map<String, String> createGenerateKeyPair(String filePath) {

**try** {

KeyPairGenerator keyPairGen = KeyPairGenerator.*getInstance*("RSA");

keyPairGen.initialize(2048);

KeyPair keyPair = keyPairGen.generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

String publicKeyString = *getKeyString*(publicKey);

String privateKeyString = *getKeyString*(privateKey);

FileWriter pubfw = **new** FileWriter(filePath + "/key.pub");

FileWriter prifw = **new** FileWriter(filePath + "/key.pri");

BufferedWriter pubbw = **new** BufferedWriter(pubfw);

BufferedWriter pribw = **new** BufferedWriter(prifw);

pubbw.write(publicKeyString);

pribw.write(privateKeyString);

pubbw.flush();

pubbw.close();

pubfw.close();

pribw.flush();

pribw.close();

prifw.close();

Map<String, String> map = **new** HashMap<String, String>();

map.put("publicKey", publicKeyString);

map.put("privateKey", privateKeyString);

**return** map;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**private** **static** PublicKey getPublicKey(String key) **throws** Exception {

**byte**[] keyBytes;

keyBytes = (**new** BASE64Decoder()).decodeBuffer(key);

X509EncodedKeySpec keySpec = **new** X509EncodedKeySpec(keyBytes);

KeyFactory keyFactory = KeyFactory.*getInstance*("RSA");

PublicKey publicKey = keyFactory.generatePublic(keySpec);

**return** publicKey;

}

**private** **static** PrivateKey getPrivateKey(String key) **throws** Exception {

**byte**[] keyBytes;

keyBytes = (**new** BASE64Decoder()).decodeBuffer(key);

PKCS8EncodedKeySpec keySpec = **new** PKCS8EncodedKeySpec(keyBytes);

KeyFactory keyFactory = KeyFactory.*getInstance*("RSA");

PrivateKey privateKey = keyFactory.generatePrivate(keySpec);

**return** privateKey;

}

**private** **static** String getKeyString(Key key) **throws** Exception {

**byte**[] keyBytes = key.getEncoded();

String s = (**new** BASE64Encoder()).encode(keyBytes);

**return** s;

}

**public** **static** String encrypt(PublicKey publicKey, String plainText) {

**try** {

*cipher*.init(Cipher.***ENCRYPT\_MODE***, publicKey);

**byte**[] enBytes = *cipher*.doFinal(plainText.getBytes());

**return** (**new** BASE64Encoder()).encode(enBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** String publicKeyFileEncrypt(String publicKeyFilePath, String plainText) {

**try** {

FileReader fr = **new** FileReader(publicKeyFilePath);

BufferedReader br = **new** BufferedReader(fr);

String publicKeyString = "";

String str;

**while** ((str = br.readLine()) != **null**) {

publicKeyString += str;

}

br.close();

fr.close();

*cipher*.init(Cipher.***ENCRYPT\_MODE***, *getPublicKey*(publicKeyString));

**byte**[] enBytes = *cipher*.doFinal(plainText.getBytes());

**return** (**new** BASE64Encoder()).encode(enBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** String encrypt(String publicKey, String plainText) {

**try** {

*cipher*.init(Cipher.***ENCRYPT\_MODE***, *getPublicKey*(publicKey));

**byte**[] enBytes = *cipher*.doFinal(plainText.getBytes());

**return** (**new** BASE64Encoder()).encode(enBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** String decrypt(PrivateKey privateKey, String enStr) {

**try** {

*cipher*.init(Cipher.***DECRYPT\_MODE***, privateKey);

**byte**[] deBytes = *cipher*.doFinal((**new** BASE64Decoder()).decodeBuffer(enStr));

**return** **new** String(deBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** String decrypt(String privateKey, String enStr) {

**try** {

*cipher*.init(Cipher.***DECRYPT\_MODE***, *getPrivateKey*(privateKey));

**byte**[] deBytes = *cipher*.doFinal((**new** BASE64Decoder()).decodeBuffer(enStr));

**return** **new** String(deBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

**public** **static** String privateKeyFileDecrypt(String privateKeyFilePath, String enStr) {

**try** {

FileReader fr = **new** FileReader(privateKeyFilePath);

BufferedReader br = **new** BufferedReader(fr);

String privateKeyString = "";

String str;

**while** ((str = br.readLine()) != **null**) {

privateKeyString += str;

}

br.close();

fr.close();

*cipher*.init(Cipher.***DECRYPT\_MODE***, *getPrivateKey*(privateKeyString));

**byte**[] deBytes = *cipher*.doFinal((**new** BASE64Decoder()).decodeBuffer(enStr));

**return** **new** String(deBytes);

} **catch** (InvalidKeyException e) {

e.printStackTrace();

} **catch** (IllegalBlockSizeException e) {

e.printStackTrace();

} **catch** (BadPaddingException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

}