Webgiisoo Framework V3.2

使用开发指南

宗旨：**做一个 Web Java 的快速开发框架，让开发人员能够容易上手，编写高性能应用系统**，而不需要去了解一堆与业务无关的“计算机技术”；我们喜欢很容易找到别人写的代码在哪里；我们不喜欢花半天时间都没有理清代码和配置文件的关系。

1 概述

Webgiisoo 框架是运行在 Tomcat 下的一个 Web 应用框架，包含了基础 web 应用 api 和在其基础上开发的模块、和一组开发规范和运行期的管理和调度。适合与具有 Web 界面的应用开发。

提供了适合 Java Web 开发的模块管理和主题管理。

提供基础 API： 配置管理、数据连接池、线程池管理、缓存、文件仓库、MDC 通信框架。

1）封装的基础 API，为应用开发提供基础资源的管理和更加简便的调用机制。

2）提供标准简便的模块管理和模型开发，只需要实现相应的业务模型和业务模型的界面。更大的重用了代码。

2 安装配置

2.1 安装与配置

2.1.1 软硬件环境

- Linux （Centos， Ubuntu， …）
- Nginx 1.2+， HTTP 反向代理服务器，web 访问提速和并行处理，虚拟域服务；
- JDK 1.7+, Java 虚拟机；
- Python 2.6+, 应用守护狗；
- Mysql 5.0+/postgresql 8.4+/Oracle 10g+, 数据库服务器；
- Mongo 2.4+, NON-SQL 数据库服务器；
- Memcached 1.0+, 内存缓存 （可选）；
- Webgiisoo, Web 应用服务器。

2.1.2 软件安装与配置

2.1.2.1 CentOS

1）参见 CentOS 安装手册， Base Server;

2）常见设置如下：

设置网络地址：

1. 修改操作系统的网络

cd /etc/sysconfig/network-scripts/

vi ifcfg-eth0

#修改以下内容

ONBOOT=yes

BOOTPROTO=static

IPADDR=

NETMASK=

GATEWAY=

DNS1=

#重启网络

/etc/init.d/network restart

设置 SELinux

vi /etc/selinux

#修改以下内容

SELINUX=disabled

关闭 IPv6

vi /etc/sysconfig/network

#修改以下内容

IPV6INIT=no

#复制本地文件~/d/jiahao/centos/etc/modprobe.d 到对应目录下

scp disable-ipv6.conf root@192.168.1.110:/etc/modprobe.d/

scp dist.conf root@192.168.1.110:/etc/modprobe.d/

#重启

init 6

设置防火墙

vi /etc/sysconfig/iptables

#修改以下内容

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT

#重启防火墙

/etc/init.d/iptables restart

设置开机自启动

```
#查看是否已经自启动
chkconfig --list ...
#添加到自启动
chkconfig --add ...
#开启或关闭自启动
chkconfig ... on/off


chkconfig postgresql-9.3
chkconfig --list nginx
chkconfig --list smb
chkconfig --list mysqld
...
```

2.1.2.2 Nginx

1）参见 nginx 安装方法，配置 apt-get/yum repo 源，直接使用 apt-get install nginx 或 yum install nginx 方法安装。对无法 Internet 环境，下载 nginx rpm 包和相应的依赖包，使用 rpm –ivh 安装；

2）配置 nginx，参见：

```
upstream demo {
        server joe-t:9302;
}
server {
        listen              80;
        server_name     s1;
        location / {
                proxy_set_header Port 80;
                proxy_set_header X-Real-IP $remote_addr;
                if ( $request_uri ~ /(images|js|css)/.*) {
                                proxy_pass http://demo/s1$request_uri;
                                expires 365d;
                                break;
                }
               if ( $request_uri ~ /(repo)/.*) {
                                proxy_pass http://demo/s1$request_uri;
                                expires 1d;
                                break;
               }
               proxy_pass http://demo/s1/;
        }
}
```

2.1.2.3 JDK1.7+

下载 JDK 1.7+，直接解压到/opt/jdk…目录；

设置/etc/profile 包含：

export JAVA_HOME=/opt/jdk…

export PATH=JAVA_HOME/bin:$PATH

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="9301" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
  <Service name="Catalina">
    <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="1000" minSpareThreads="4"/>
    <Connector executor="tomcatThreadPool"
                  port="9302" protocol="HTTP/1.1"
                  connectionTimeout="20000"
                 URIEncoding="GBK"
                  redirectPort="8443" />
    <Engine name="Catalina" defaultHost="joe-t">
      <Host name="joe-t"    appBase="webapps"
            unpackWARs="true" autoDeploy="true">
     <Context path="/s1" docBase="/opt/d/joe/www/s1" reloadable="false"/>
      </Host>
    </Engine>
  </Service>
</Server>
```

2.1.2.4 Python 2.6+

参见 python 2.6+安装方法。

2.1.2.5 Postgresql

参见 mysql/postgresql/oracle 安装方法；

创建相应的数据库和可使用该数据库的用户。以下以 postgresql 9.3 为例：

```
//initdb and start
/etc/init.d/postgresql-9.3 initdb
/etc/init.d/postgresql-9.3 start

//create db and role
sudo -u postgres psql
create database demo;
sudo -u postgres psql demo
create role giisoo with password 'demo123456';
alter role giisoo with login;
grant all on database demo to giisoo;
```

确保数据库用户认证为 MD5

/etc/postgresql/9.3/main/pg_hda.conf

```
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
host    all             all             192.168.1.1/24          md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
```

2.1.2.6 Memcached

参见 memcached 安装方法。

2.1.2.7 Appdog

应用守护狗

#复制/etc/init.d/appdog 到服务器对应的目录中

scp appdog root@192.168.1.100:/etc/init.d/

#复制/etc/appdog/apps.conf 到服务器对应的目录中

scp apps.conf root@192.168.1.100:/etc/appdog/

vi /etc/appdog/apps.conf

[app:mongo]

start=/opt/mongodb/bin/m1.sh

pattern=/opt/mongodb

path=/opt/mongodb

user=

check=0.5

notify=0

enabled=1

2.1.2.8 Webgiisoo安装配置

获取安装包： giisoo_3.2_[buildno].tar.gz, http://www.giisoo.com 下载

1）解压至你的目录：

tar xzf giisoo_3.2_[buildnno].tar.gz

修改配置文件：

vi giisoo/webgiisoo/conf/giisoo.propertiees

2）修改 giisoo/webgiisoo/conf/giisoo.properties

a) 数据库驱动 db.driver=配置为合适的数据库 mysql/postgresql/oracle;

b) 数据库连接 db.url=安装的数据库和相应的用户， 比如：

db.url=jdbc:mysql://127.0.0.1:3306/s1?user=demo&password=demo123&useUnicode=true&characterEncoding=GBK

c) 配置数据连接池的连接数 db.number= （缺省 10 个）

3）修改 giisoo/webgiisoo/conf/log4j.properties，使日志输出到合适的位置；

4）启动 webgiisoo 会根据数据库配置自动创建必须的数据表，运行 giisoo/bin/startup.sh；

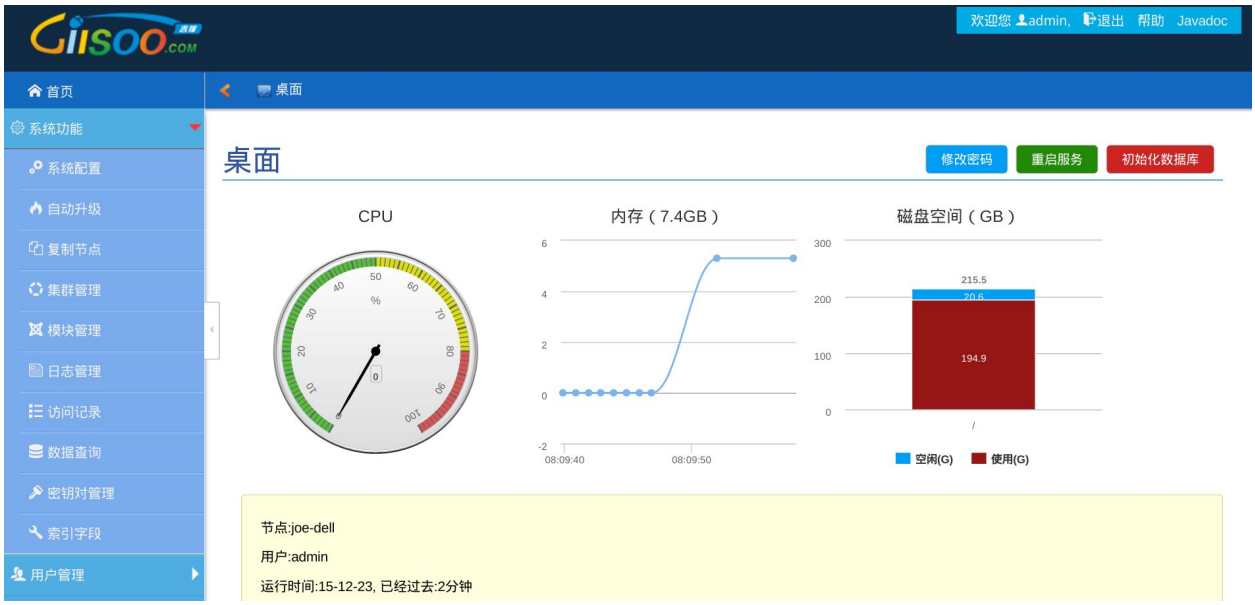
5）Webgiisoo 框架安装配置好后，仅提供后台管理，不提供用户前端的 Web 的页面。

2.2 应用配置

1）登录后台

缺省用户名：admin/admin，注意：密码错误 3 次会被锁定 1 小时， 账户是按照 IP 地址锁止的， 如果需要立即解锁， 需要清除 mongo 数据表中：gi_userlock 中的数据。

2）登录后台后，可以进行系统设置，用户管理，模块管理和日志查询等。



2.3 模块管理

Webgiisoo 框架使用模块化管理，所有“应用开发”在 webgiisoo 中被作为一个模块，模块之间可以复用和“override”，参见《快速开发，模块开发》。

进入后台管理，模块管理>>模块。

模块管理可以上传一个新模块或更新一个模块，下载、关闭模块，关闭后的模块可以删除。

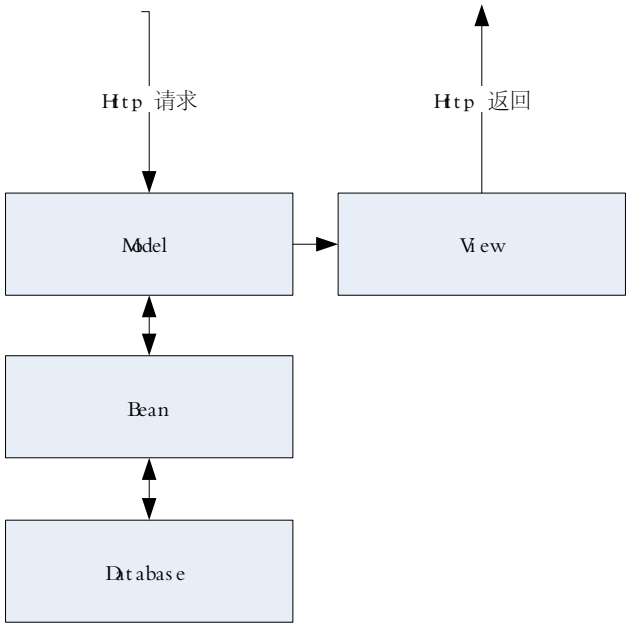
3 快速开发

Webgiisoo 提供 Web 开发框架和模块管理机制，开发用户需要根据自己的业务需求，开发自己的应用模块或复用别的模块，以向最终用户提供自己的业务服务。

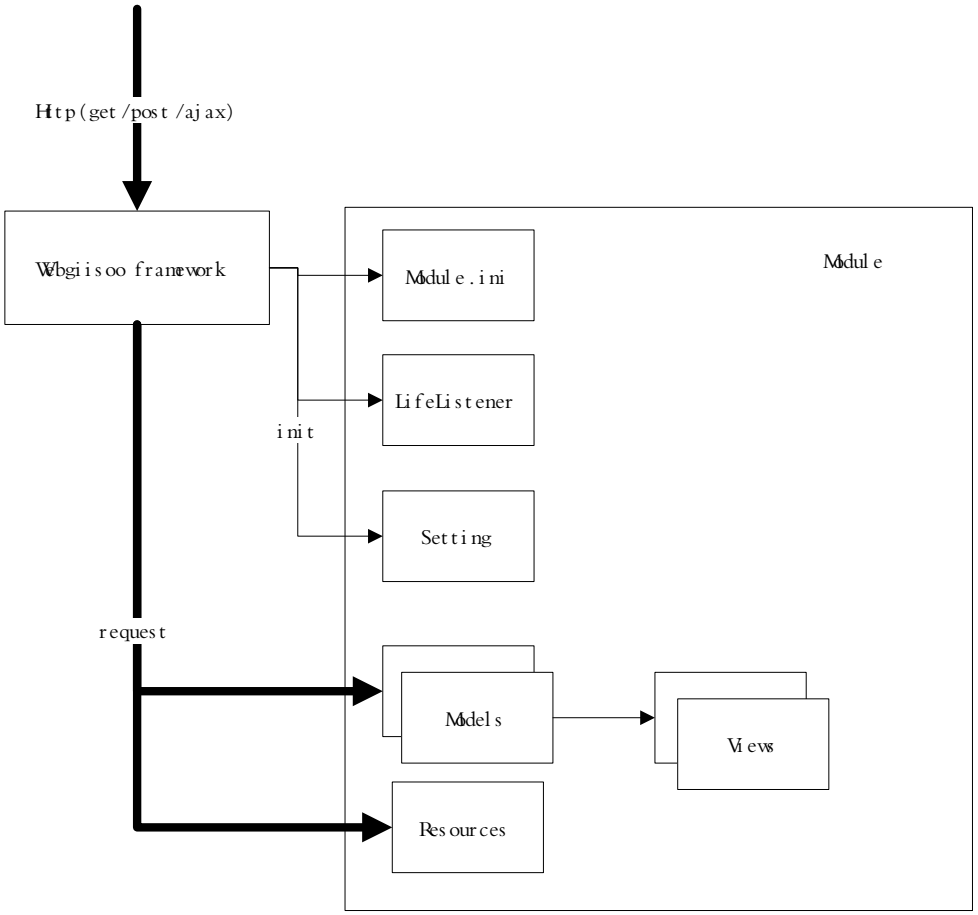
在开发自己的应用模块之前，需要了解 webgiisoo 的框架模块。

3.1 框架模型

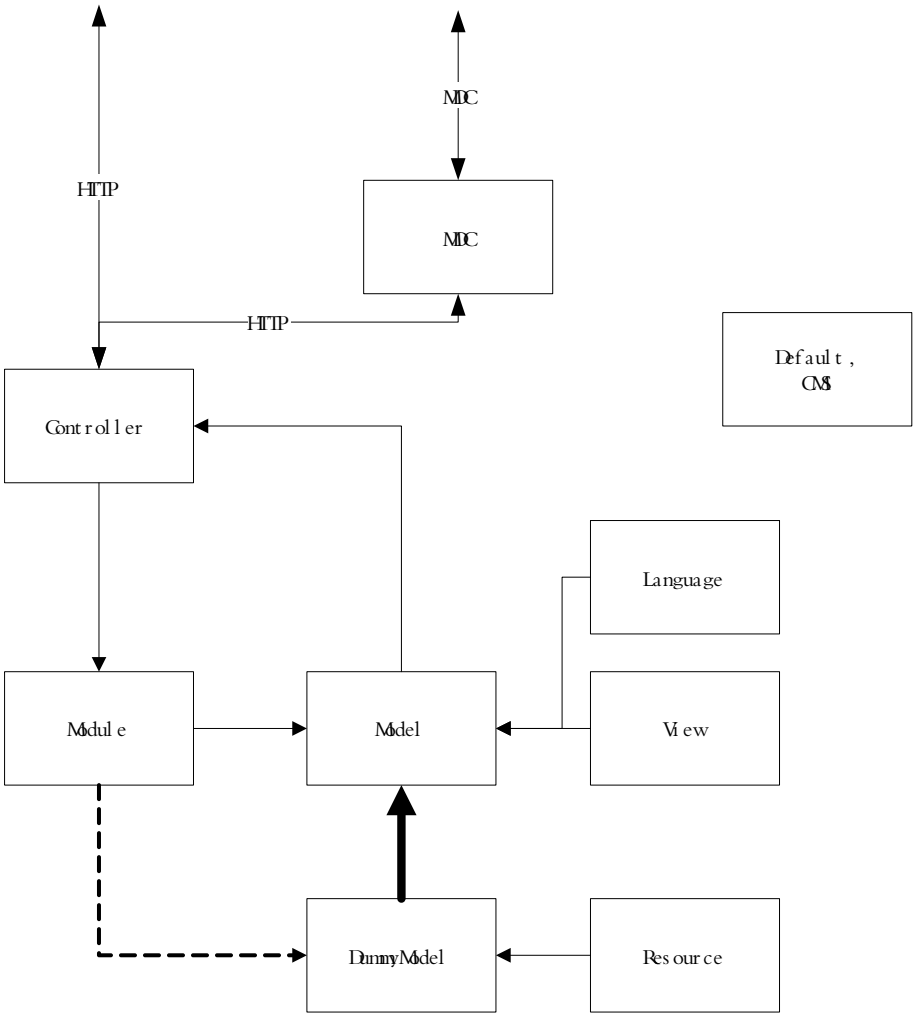
首先 Webgiisoo 框架简化了 MVC 模型， 我们只有 MV, M: 模型，负责业务逻辑处理和数据获取和设置；V：模板，采用 velocity 模板语言的 html 页面。



基本框架图



Webgiisoo Framework 框架模型图



图（2）Webgiisoo 框架内部流程图

◆Controller: 负责装载 module， 分发请求。

◆Module: 管理模块的生命周期，装载 Model，并缓冲 model 和 view。

◆Model：负责业务逻辑处理，并返回用户 View。

◆View: Template,一组 HTML 和 JS 页面，负责界面展示。

◆MDC: MDC 通信模块，支持 UDP、断点续传、加密、压缩，提供一种适合移动网络传输的高效通信协议组件。

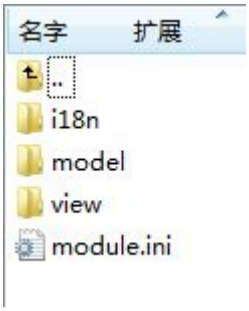
◆CMS: 内容管理，基于 Web Giisoo 框架开发的缺省内容管理，包括后台管理、用户管理、内容发布等。

3.2 命名规则

Web Giisoo 框架是直接利用 URI 与 java 类映射来完成请求分发的，简化了配置项目，但同时也需要遵循相应的规则。

3.2.1 模块命名规则

一个模块的目录结构如下图：



图（3）模块目录结构

1） module.ini 文件，模块的根目录必须包含该文件， 该文件包含了对模块的基本说明，参见《模块管理，module.ini》。

2） /view 目录，视图、资源文件存放的“根目录”。

3） /model 目录，模块和模块的依赖的 jar 文件目录，Webgiisoo 框架会自动加载 /model 下所有 jar 文件。

4） /i18n 目录，国际化语言文件的“根目录”。

3.2.2 模型命名规则

URI 直接对应模型的名称，URI 中的 path 对应模型的“子包”名。根包名由 module.ini 指定， 比如：根包名为 com.giisoo.web， 模型的全名 com.giisoo.web.user.class，则对应的 URI 为/user；如果模型的全名为 com.giisoo.web.user.my.class，则对应的 URI 为 /user/my。

模块序号大的 URI 会覆盖序号小的模块中相同的 URI。

3.2.3 View命名规则

View 必须存放在/view 目录下，否则无法访问。对实际命名没有特殊要求。View 的选择是依靠 model 在处理单元中，根据实际处理逻辑指定。

模块序号大的会覆盖序号小的模块中相同“路径”的 view 文件。

3.2.4 模型装载策略

Controller 收到一个 HTTP 请求后，从序号最高模块中装载与 URI 对应的模型，URI 与模样匹配策略根据模型的命名规则确定。如果不能发现与 URI 匹配的模型， 则依次从序号高的模块中装载，直到由模块装载成功。装载次序：

1）从序号最大的模块中开始解析 URI 成对于的 class。

2）如果找到对应的 class, 则调用该模型对应的 onGet、onPost 或 onMDC， 并预检查对应方法需要的的权限和预装载的国际化语言。

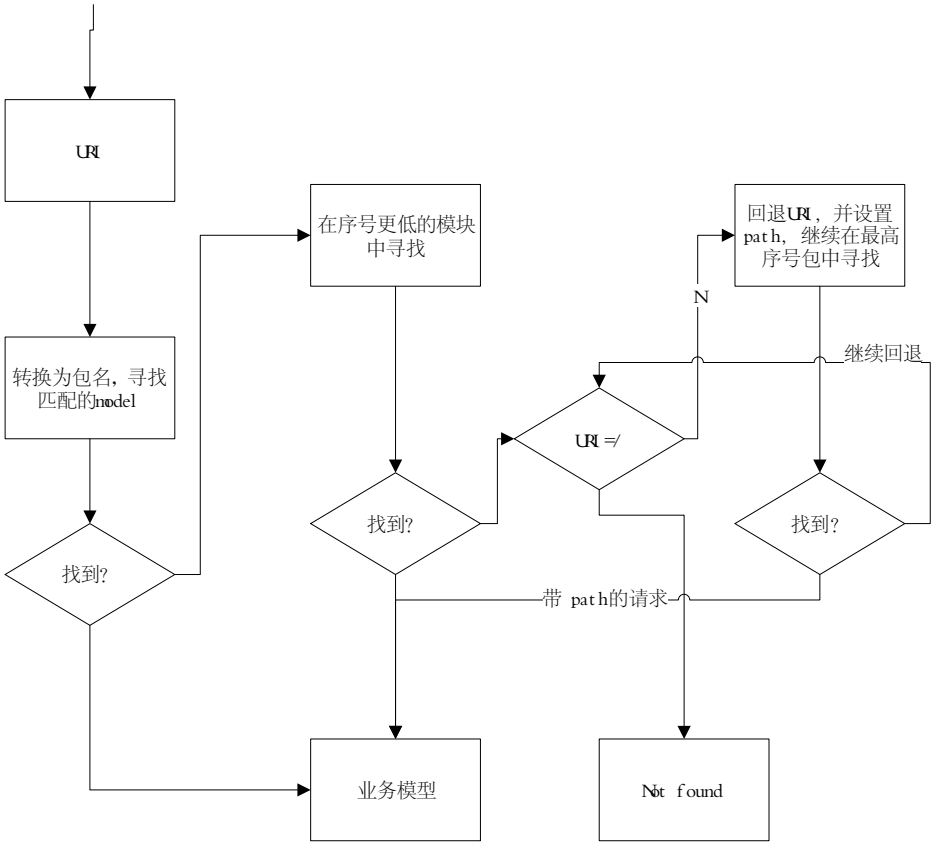
3）如果装载失败，则一次从序号低的模块中装载，重复 1)。

4）如果所有模块都遍历，并且装载失败，则尝试 rewrite url， 并重复 1), rewrite 的策略和次序如下：

a) 检查系统中的 urlmapping 规则， 如果匹配上预制的 url 规则， 则 rewrite。

b) 回退 uri，回退 URI 即 URI 的上层“目录”， 比如：/user/my, 回退 URI 为/user，并把 my 作为 path 传入/user 中，目的是实现 URI 的目录式访问。

c) 如果所有回退都无法处理，则 Controller 试图作为资源文件直接访问 module 的/view 下的资源文件， 如果找到资源文件， 则响应 HTTP 请求，否则响应”not found”错误信息。如下图：



图（5）模型装载策略图

3.3 开发环境

Linux， JDK 1.8+， Eclipse 3.4+， Ant 1.8+。

3.4 调试环境

环境同运行环境;

JDK 1.8+, Tomcat 7.0+, Mongo 2.4+, Postgresql 9.3+;

使用 Tomcat 的远程调试功能，jpda;Tomcat 脚本或 Appdog 需要调用 jpda start 参数。

3.5 生成第一个模块

登录已经部署的 webgiisoo，进入/admin，系统功能->模块管理，点击新建模块， 输入：

ID：模块的编号， 编号越大， 越先被加载;

Name：模块的名称， 相同名称的模块会被认为同一个模块;

Package：模块中 web api 的 package 名称;

LifeListener：模块启/停事件监听类名称，前缀 package 名称保持与 Package 名相同;

Setting：模块本身设置处理类，框架会自动加载到框架的系统设置页卡中;

Readme：说明

点击“创建”会自动生成 Eclipse 的工程文件包。

3.6 模块 Demo 学习

Webgiisoo 中的模块开发即是 eclipse 中的一个工程。模块开发分为模型开发和模板开发。

以 Demo 项目为例，快速开发和部署模块，包括显示 demo 页面，提交数据，保存数据，查询数据，显示 demo 数据列表。

3.6.1 开发环境

Eclipse 3.4+, Ant 1.8+，添加 Demo 项目。

3.6.2 Module.ini

```
name = demo
package = com.giisoo.demo.web
screenshot=/images/demo_screenshot.png
readme=Demo 模块
version=1.0
build=0
```

id = 100

enabled = true

- Name

  标识模块的名称，相同名称的模块会认为是同一个模块。

- Package

  模块中“模型”的根包名，框架会访问的模型自动加上包名后查询类名。比如：/demo，快加会自动装载 com.giisoo.demo.web.demo.class 以处理请求。

- Screenshot

  简要屏幕截图或 logo 之类的。

- Readme

  模块说明。

- Version

  模块版本号。

- Build

  模块 build 编号，自动生成， 不用填写。

- Id

  模块编号，决定模块中模型在框架中查找的顺序，编号大的优先找到，为系统提供“override”机制。

- Enabled

  模块打开或关闭。

3.6.3 Build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project name="demo" default="build" basedir=".">

    <!-- Project Properties -->
    <property file="src/module.ini" />

    <property name="DEPLOYDIR" value="/opt/d/joe/www/demo" />
    <property name="REPOSITORY" value="http://w1.giisoo.com/repository" />
    <property name="DEPENDS" value="default" />
    ......
```

- REPOSITORY

  依赖代码远，用于开发模型时，获取对框架的依赖包。

3.6.4 Model

3.6.4.1 Index.java

```
package com.giisoo.demo.web;

import com.giisoo.framework.web.Model;
import com.giisoo.framework.web.Path;

/**
 * demo  模块缺省首页
 *
 * @author joe
```

```
 *
 */
public class index extends Model {

    /**
     * Path 声明，这里不需要任何权限， 都缺省。
     */
    @Path()
    public void onGet() {
        /**
         * 显示 view/index.html 模板，模板的根目录为模块下 view/
         */
        this.show("/index.html");
    }
}
```

3.6.4.2 Demo.java

```
package com.giisoo.demo.web;

import com.giisoo.bean.Beans;
import com.giisoo.demo.bean.Demo;
import com.giisoo.framework.mdc.X;
import com.giisoo.framework.web.Model;
import com.giisoo.framework.web.Path;
```

```
/**
 * demo， 必须继承 Model 类， url: /demo
 *
 * @author joe
 *
 */
public class demo extends Model {

    // 缺省 path，不需要 login
    @Path()
    public void onGet() {
        /**
         * 从 http request 中获取参数
         */
        int s = this.getInt("s");
        int n = this.getInt("n");

        /**
         * 从数据库中查询 Demo
         */
        Beans<Demo> bs = Demo.load(null, s, n);

        /**
         * 设置回 Model，供模板使用
```

```
         */
        this.set(bs, s, n);

        /**
         * 显示模板
         */
        this.show("/demo/demo.index.html");
    }

    // url=demo/add, 需要用户登录，需要用户有 access.config.admin 权限令牌， 并且记录 POST 日志。
    @Path(path = "add", login = true, access = "access.config.admin", log = Model.METHOD_POST)
    public void add() {
        /**
         * 如果是 POST 方法...
         */
        if (method.isPost()) {
            /**
             * 从 http request 中获取参数
             */

            // 获取 string 类型的参数， 会自动转义"<", ">"等 html 符号
            String name = this.getString("name", 20);
```

```
            // 获取 html 惨素， 不会转义 html tag
            String description = this.getHtml("description", 4096);

            // 存入数据库
            String id = Demo.create(login.getId(), name, description);

            // 从数据库获取数据
            Demo d = Demo.load(id);

            // 设置到 Model 中， 供模板使用
            this.set("d", d);

            //设置消息提示信息
            this.set(X.MESSAGE,  lang.get("demo.add.success"));

            // 显示模板
            this.show("/demo/demo.detail.html");
            return;
        }

        // 显示模板
        this.show("/demo/demo.add.html");
    }
```

```
}
```

3.6.5 数据库访问

3.6.5.1 Demo实体类

```
package com.giisoo.demo.bean;

import java.sql.ResultSet;
import java.sql.SQLException;

import com.giisoo.bean.Bean;
import com.giisoo.bean.Beans;
import com.giisoo.bean.DBMapping;
import com.giisoo.bean.UID;
import com.giisoo.bean.X;

/**
 * Demo 数据表访问类
 *
 * @author joe
 *
 */
// 声明与 tbldemo 对应
@DBMapping(table = "tbldemo")
public class Demo extends Bean {
```

```
// 类变量
int uid;
String id;
String name;
String description;
long created;

/**
 * 创建一个 Demo 数据记录，并返回数据项的 ID
 *
 * @param uid
 * @param name
 * @param description
 * @return String
 */
public static String create(int uid, String name, String description) {
    // name 是否为 null 或 ‘’
    if (X.isEmpty(name)) {
        return null;
    }

    /**
     * 根据 uid, name, time 生成唯一 ID
     */
    String id = UID.id(uid, name, System.currentTimeMillis());
```

```
        /**
         * 使用 V 对象来构造 values 子句， 而不用去数 ( ?, ?, ? ) 多少个?， 是否对齐
         */
        V v = V.create("name", name).set("uid", uid)
                .set("description", description).set("id", id)
                .set("created", System.currentTimeMillis());

        /**
         * 插入数据
         */
        if (Bean.insert(v, Demo.class) > 0) {
            return id;
        } else {
            return null;
        }
    }

    /**
     * 更新数据
     *
     * @param id
     * @param v
     * @return int
     */
```

```
    public static int update(String id, V v) {
        /**
         * 更新数据，并返回更新的条数
         */
        return Bean.update("id=?", new Object[] { id }, v, Demo.class);
    }

    /**
     * 查询数据，查询条件在 W 中，W 封装了 where 子句和参数， 不用再数? 个数，也不用管是否对齐了。
     *
     * @param w
     * @param s
     * @param n
     * @return Beans<Demo>
     */
    public static Beans<Demo> load(W w, int s, int n) {
        return Bean.load(w == null ? null : w.where(),
                w == null ? null : w.args(),
                w == null || X.isEmpty(w.orderby()) ? "order by created desc"
                        : w.orderby(), s, n, Demo.class);
    }

    /**
     * 查询 ID 的 demo 数据项。
```

```
     *
     * @param id
     * @return Demo
     */
    public static Demo load(String id) {
        return Bean.load("id=?", new Object[] { id }, Demo.class);
    }

    /**
     * 根据数据结果集合填充类变量, Bean 类在查询的时候自动调用
     */
    @Override
    protected void load(ResultSet r) throws SQLException {
        id = r.getString("id");
        name = r.getString("name");
        description = r.getString("description");
        created = r.getLong("created");
        uid = r.getInt("uid");
    }

    public String getId() {
        return id;
    }

    public String getName() {
```

```
        return name;
    }

    public String getDescription() {
        return description;
    }

    public long getCreated() {
        return created;
    }

    public int getUid() {
        return uid;
    }

}
```

3.6.5.2 数据库脚本

框架会自动根据模块的数据库脚本创建数据表。数据表本存储在：/view/install/mysql, /view/install/postgresql，或/view/install/oracle，对应 mysql, postgresql, oracle 数据库。

Initial.sql

```
create table tbldemo(
    id varchar(20),
    uid int,
    name varchar(20),
```

```
    description varchar(4096),
    created bigint
);
create index tbldemo_index_id on tbldemo(id);
```

3.6.6 View

Html 模板和所有资源的跟目录。

3.6.7 I18n

国际化语言的包文件。根据用户浏览器和模块的缺省语言设置自动装载。

Zh_cn.lang 中文

En_us.lang 英文

3.6.8 编译发布

3.6.8.1 Ant update

进入模块的跟目录，运行 ant update,根据 build.xml 中配置的代码源，获取对框架的依赖包。

3.6.8.2 Ant

打包，进入模块的更目录，运行 ant，自动把模块打包为框架能识别的 zip 文件包， zip 文件存在模块目录在 target 下。包括所有的设置、语言、view 模板文件和模型等。

3.6.8.3 上传模块

进入已经配置好的框架后台管理>>模块管理， 上传 zip 文件包。重启应用服务器。

3.6.9 测试demo

http://[domain]/，将转到 index.onGet()方法。

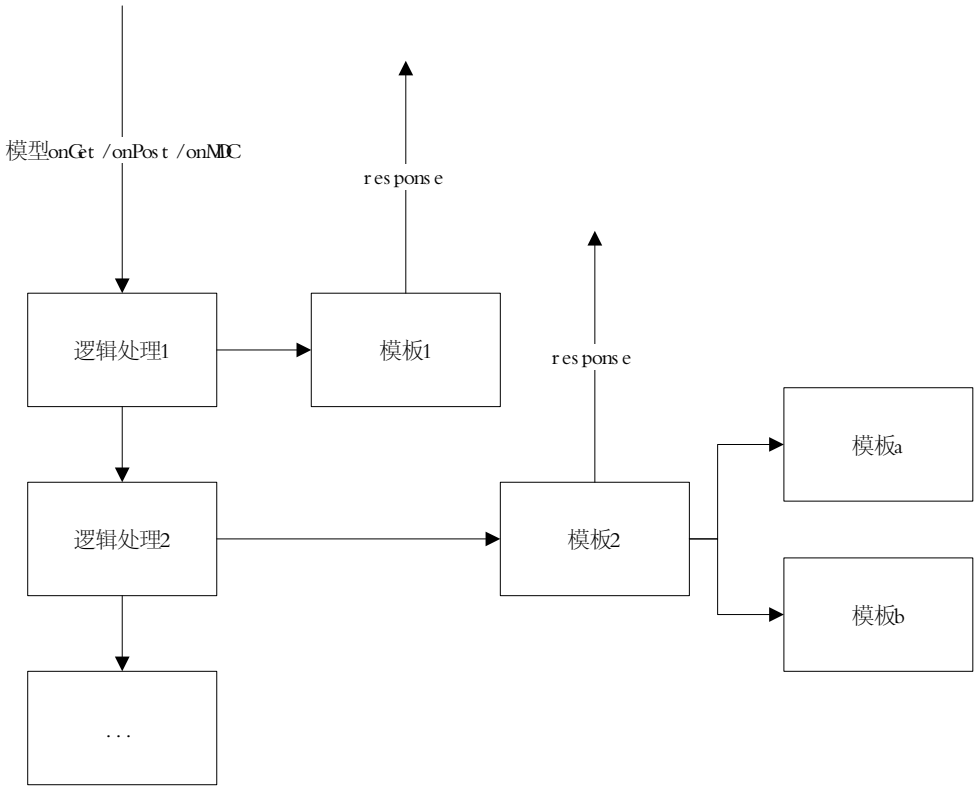
http://[domain]/demo，将转到 demo.onGet()方法。

http://[domain]/demo/add，将转到 demo.add()方法。

3.6.10 更多

3.6.10.1 模板开发

模板就是一组 html 文件，该文件中包含 velocity 宏语言，（该语言类似可读取更高，明显优于 html 中 jsp 语言。模板的装载需要在模型的逻辑处理中明确调用。请参见《Apache Velocity》了解更多。



图（6）模板与模型处理流程图

3.6.10.2 模板中的全局变量

- $module，对应 com.giisoo.framework.web.Module，可以获取模块的配置参数等。
- $request，对应 javax.servlet.http.HttpServletRequest
- $response，对应 javax.servlet.http.HttpServletResponse
- $lang， 对应 com.giisoo.framework.web.Lanugage，获取国际化语言， 框架会自动

根据用户浏览器支持的语言和用户浏览时设置的 Cookie(“lang”) 或 Query(“lang”)来初始化调取的国际化语言包。

- $uri， 请求的 uri 字符串。
- $path，请求的 uri 后的字符串。

3.6.10.3 权限令牌

Webgiisoo 使用权限令牌的方式来实现权限的分配和鉴权。

权限令牌不需要额外维护，只需要在模型方法中，使用@Path 指定的权限令牌 access 和 user.hasAccess(“???”)鉴权的权限令牌会自动加入到数据库的权限令牌库中，供后台管理>>用户管理>>角色管理分配指定。比如：

```
@Path(path=”add”, login=true, access=”access.user.add”)
Public void add{
    //TODO
}
等同于：
@Path(path=”add”, login=true)
Public void add{
    If(login.hasAccess(“access.user.add”){
        //TODO
    } else {
        This.redirect(“/user/login”);
    }
}
```

约定：

1）权限令牌令牌的命令必须是：access.[group].[op]方式，并需要在 i18n 中对权限

进行翻译。比如：access.user.add=添加用户， group.access.user=用户管理

2）access.[group].admin 包含所有[op]的权限令牌。

3.6.10.4 日志记录

日志记录分自动记录和 API 调用两种。

1) 自动记录应用在@Path 中指定 log=Model.METHOD_POST|Model.METHOD_GET 中，webgiisoo 框架会自动记录 post 和 get 方法的输入参数和输出结果。(password, pwd 等标签不会记录)。如果出现发那个发调用异常， webgiisoo 会自动记录 warn 日志。
2) 或在应用中使用 OpLog.log (….) 记录操作日志。

3.6.10.5 模块设置

模板设置的都存储在 module.ini 文件中，除 Webgiisoo 框架需要的设置外，模块的“设置”必须以“setting.”开头，否则不可以读取。设置值可以预配置 module.ini 文件，或模块开发自己开发设置页面进行管理。在

模块设置值的获取和设置：在模块的生命周期中和模型基类中，通过 module.get(setting) ， 或在模板中使用$module.user_add == ‘true’的方式获取。

4 开发 API

4.1 配置管理

Webgiisoo 的配置分三类：

4.1.1 giisoo.properties 中的配置

在模型中可以使用 conf.get 方法和 SystemConfig.[i/s/l]方法获取， 不可以修改，但是可以被运行期配置 override。比如：conf.getString(“node”)==SystemConfig.s(“node”)。

4.1.2 运行期可以修改的配置

保存运行期用户修改的配置和获取。Java 代码中，使用 SystemConfig.s[i|l]获取，使

用 SystemConfig.setConfig(name, object)保存。

在 html 模板中，可以使用$system.s[i|l]获取。

SystemConfig.s(name)，获取字符串类型；

SystemConfig.i(name)，获取整数类型；

SystemConfig.l(name)，获取长整类型。

4.1.3 模块开发期指定的配置

开发期配置存储与 module.ini 中，运行期间不可以修改。在 Java 中获取方法 module.get(name)；在 html 模板中使用$module.name 方法获取。

4.2 最重要的抽象类

4.2.1 Model

所有处理 Web/MDC 请求的模型类必须继承该类，其提供了丰富的 API 处理请求参数/参数文件等，和返回结果或显示页面等。参见 Model。

4.2.2 Bean

所有数据库访问实体类必须继承该类，其提供了丰富的 API 操作数据库。参见 Bean。

4.2.3 WorkerTask

多线程的任务类，提供统一线程池管理和任务管理。参见 WorkerTask。

4.3 数据库

所有数据实体类都需要继承 Bean，Bean 提供了数据库访问的所有 API。

4.4 线程池

系统提供同意的线程池管理，多线程任务只需要继承 WorkerTask，就可以直接 schedule。

4.5 缓存

所有实体类（继承于 Bean）都支持被缓存，参见 Cache.set/Cache.get。

4.6 分布式通信

框架提供 ActiveMQ 分布式通信支持。参见 MQ。

4.7 MDC 编程

MDC 为类似 https 的协议， 不同之处在于 MDC 提供服务器向客户端推送的功能。服务端没有特殊设置，仅仅在 web api 接口说明中，@Path(method=Model.METHOD_MDC)。

参加 TConn, TConnCenter。

4.8 文件仓库

系统提供文件仓库存储，并可以被 web 访问下载，文件仓库为永久存储，可以被分布式中的多台服务器访问。参见 Repo.load()/Repo.store()。

文件仓库中文件资源访问 Web 接口: /repo/[id][/name], 其中[id]为文件 ID 号,[/name]不是必须的，仅为了浏览器自动保存文件时的一个缺省名称。

4.9 临时文件

系统提供创建临时文件， 并可以被 web 访问下载，临时文件为 1 天，过期的临时文件会被自动删除。参见 Temp.get()。

除应用模块生成的临时文件，框架会自动清除临时文件，包括：Temp 生成的、Cache 文件的、Tomcat logs 目录和 Tomcat work 目录。需要在其中脚本中添加 CATALINA_HOME 环境变量。

临时文件 Web 访问接口：/temp/[id][/name], 其中[id]为临时文件 ID 号，[/name]不是必须的，仅为了浏览器自动保存文件时的一个缺省名称。

4.10 Web 接口

框架提供了部分重要的 Web 接口。

/upload，上传文件到文件仓库，支持断点续传；

/repo/[id]，下载/访问文件仓库中的文件；

/temp/[id]，下载/访问临时文件；

/mydata，支持读取/写入个人数据；

更多 API 请参见 Javadoc