**Jsock framework**

Simple java framework and server.

1. Multithreading client server (TCP/UDP)

2. Easy configuration

3. MVC like controllers (tasks)

4. Simle databese mysql usage

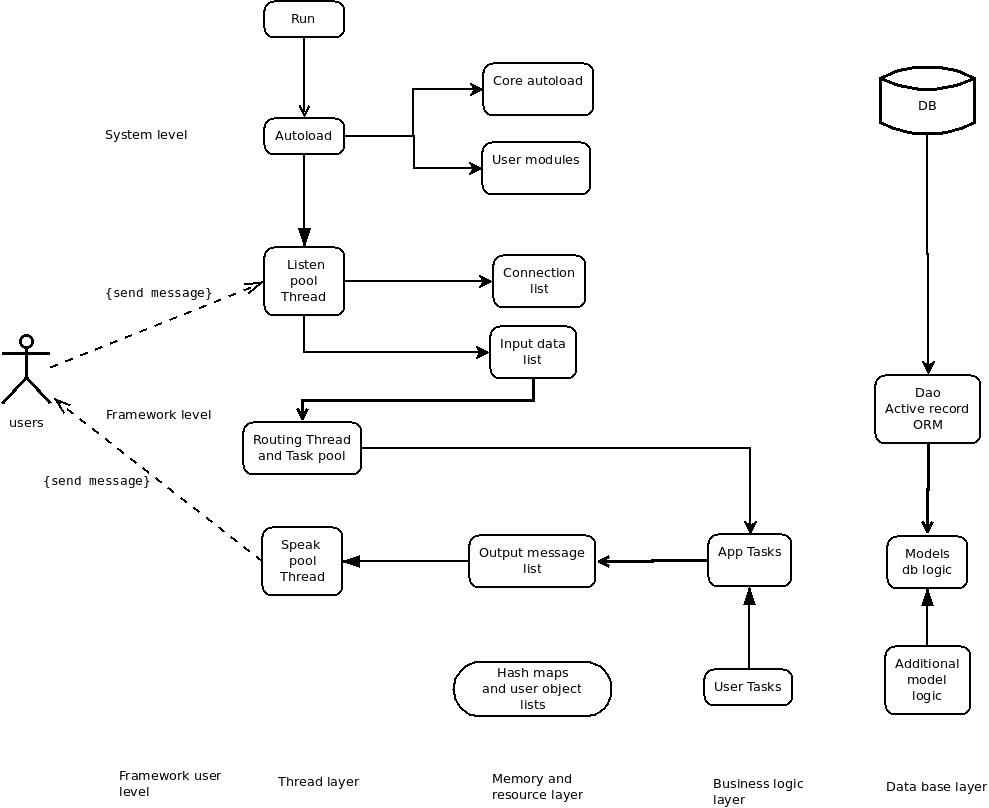
5. Web User rights and session

6. Modules

7. Data storage

8. Errors.

Framework diagram



**Basic working principle**

**Scenario**

1. Startup framework and webserver daemon

2. Read configuration

3. Load modules

4. Run pool threads

5. Receive message from client

6. Run task and compute result

7. Send message to user

**Thread pools**

1. Receive pool

2. Sender pool

3. Routing pool

4. Connection pool

Application send and receive json messages.

byte[] buffer = "{\"task\":\"JtestTask\",\"message\":\"mymessage\",}".getBytes();

**Tasks** – executable java code (like controllers in our frameworks Controllers)

**Models** – presentation of data from mysql, execute query.

**Run and Install**

**1. Install**

Download framework and copy source files to project directory.

Install musql server.

**2. Configrure**

Open conf/Jconfig.java and change settings

modules – modules list

sender\_pool – size of sender pool

resiver\_pool – size of resiver pool

task\_pool – size of resiver pool

protocol – tpc or udp sender

connection\_life\_time – connection time out

client\_port – client port

server\_port – server port

socket\_buffer\_size – size of socket buffer (length of message)

**Database settings**

mysql\_user – mysql user name

mysql\_password – mysql user password

mysql\_url – mysql connection string for jdbc

mysql\_db\_name – mysql dabase name

**3. Create database and tables**

Load dump from dump.sql (in root directory)

**5. Run application**

execute main file conf/Jsock.java for test

**6. Add external libraries if not exists**

mysql-connector-java-5.1.39-bin.jar

json-simple-1.1.1.jar

**Example Task: send message**

**1. Create Task**

Copy file JtestTask.java from /example/JtestTask.java

Copy file JtestTask.java in directory from source file /example/JtestTask.java

**2. Open lines into test file**

Open jsock.tests.JclientUDPTest.java

Add lines in UDPSendThread subclass run method.

byte[] buffer = "{\"task\":\"JtestTask\",\"message\":\"mymessage\",}".getBytes();

task – executable java class (task name).

message – free data format .

3. Run application conf/Jsock.java

4. Run jsock.tests.JclientUDPTest.java

out: {"message":"mymessage"}

**Task basics**

**1. All task extend** **JclientTask.**

**2. Rules method.**

Method contains rules set, for example user message must contain field message.

@Override

public String[][] rules(){

String[][] rules = {

{"require","message"}

};

return rules;

}

Validation code located in jsock.validators and include in JclientTask.

**3. User rights**

Method contains string of user grands, who have access to the task. Default user privilege guest.

For example: guest,user,moderator,manager,admin

@Override

public String rights() {

//String rigths = "user,admin";

String rigths = "guest";

return rigths;

}

**4. Before action**

Code that runs before action. Method defined in JTask and override in JclientTask.

@Override

public void beforeAction(){

super();

//user code here

}

**5. After action**

Code that runs after action. Method defined in JTask and override in JclientTask.

@Override

public void afterAction(){

super();

//user code here

}

**6. Action**

The main method in the task

@Override

public void action(){

//System.out.println(“The task action complite”):

}

**7.** **Receive** **message from client**

In action methodwrite

String message = this.message.json.get("message").toString();

Get field message from user data string.

**8. Send message to client.**

In the end of action method write

String outString = "{\"message\":\"some out string"}";

JOutMessages outMessage = new JOutMessages(this.message.ip,outString);

outMessage.insert();

When on the client side can receive a message.

**Messages collection**

jsock.message.JInMessages - Contains incoming client messages.

jsock.message.JOutMessages - Contains outgoing client messages.

**Database**

**1.** **jsock.db.****DBConnection**

Contains jdbc connect to mysql.

**2. jsock.db.DBQuery**

Contains methods for execution queries to database.

delete – delete field by id

execute – execute sql string

deleteById – delete field by id

findById – find field by id

insert – insert field by id

update – update field by id

unescapeMySQLString – escape sql string

**Models**

Models is a user class which must be created in models folder and contains database logic.

All models must be extended from DBQuery, use models in tasks.

Examples:

models.Session.java

models.Users.java

**Sutdownhook**

Located in jsock.java and triggered when system shutdown.

class ShutdownHook extends Thread {

public void run() {

System.out.println("System halt");

}

}

**Modules**

Modules is a standalone piece of code.

Modules must be located in folder jsock.modules example TestModule

For plug in modules need add module class name to config section

public static String[] modules = { "TestModule" };

**Authorization example**

1. Run the jsock.tests.JClientUDPTest with command

byte[] buffer = "{\"task\":\"JLoginTask\",\"email\":\"jetananas@yandex.ru\",\"password\":\"test\"}".getBytes();

Copy receive string

7ebc5d1781c51c50c864629299e6a5d91467830206

And run next command

//byte[] buffer = "{\"task\":\"JtestTask\",\"auth\_token\":\"7ebc5d1781c51c50c864629299e6a5d91476670763\",\"message\":\"authorized\"}".getBytes();

Server authorized user by token with admin rights.

**Cron like commands**

JCommandExecutor run user commands from directory commands.

List of first run stored in jConfig in variable executor\_timeout 19 seconds by default

In variable executor\_tasks stored array of executable commands

Open class commands.JsystemGarbage and you will see how it works:

In main method run you will see code:

if(condition()){

//your code here

}

condition – custom user code, which must return boolean (true/false)

You can create custom command.

1. Create class in commands directory

2. add class name to executor\_tasks