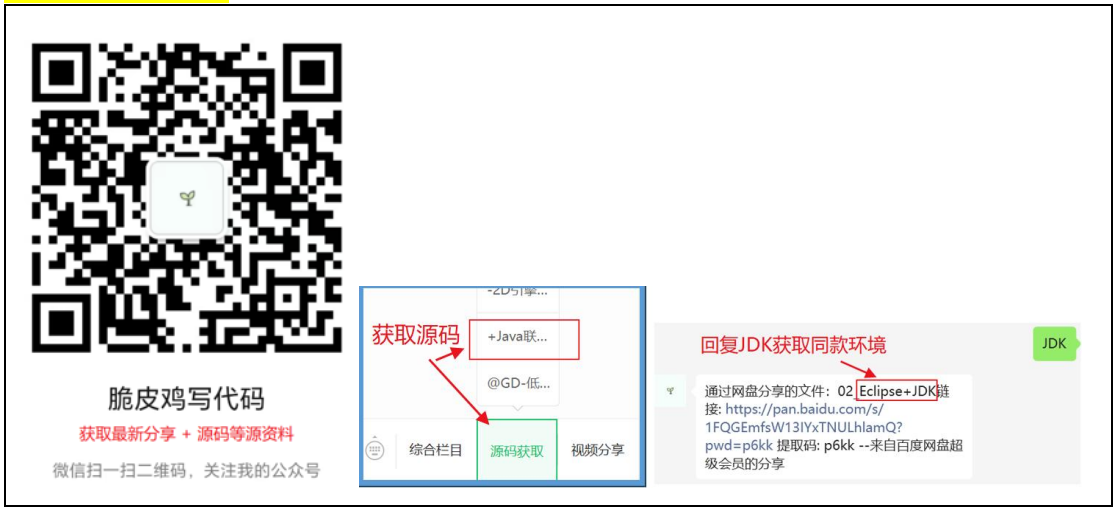
API 及应用框架说明

关注获取最新发布



1. 框架类作用简要说明

1.1. 客户端

1.1.1.com.frm.base（常用 base 类）

Singleton	单例基类，方便需要单例的类继承实现使用
GlobalClock	全局时钟单例
IGlobalClockListener	使用全局时钟需要实现的接口
EGlobalClockKey	注册全局时钟时用的 key
EGlobalClockTime	全局时钟的计时间隔常量
IClearable	实现此接口，意味着具有清理方法

1.1.2.com.frm.cfg（配置包）

INetCfg	配置连接服务器的 ip 和端口
Char	专门用来放中文字符串常量的

1.1.3.com.frm.global_observer（全局观察者模块）

GlobalObserverCtrl	主要目的是后台监听（观察）一些事件（PID）并给出处理
GlobalObserverModel	暂时用不上

GlobalObserverView	暂时用不上

1.1.4.com.frm.mvc（mvc 框架包）

BaseCtrl	Controller 类，管理着 View 和 Model 类的初始化、清理、更新
BaseModel	Model 基类，一些生命周期方法
BaseView	View 基类，一些生命周期方法
CCenter	控制器管理中心，引用 MVC 模块实例，负责打开关闭模块
IModelArgsExt	模块打开时传递的参数接口类，对初始化进行一些控制
IObserver	观察者接口，定义了观察者需要具备的能力
ModelArgsExt	IModelArgsExt 接口的实现类，数据及行为实现
MVC.java	一些 MVC 模块需要用到的能力，静态全局可以访问
MVCBaseComm	MVC 模块都需要具备的能力，对 MVC.java 类的封装
UILocType	枚举类，一个值代表一个 UI 坐标的计算方法

1.1.5.com.frm.netc（客户端网络实现包）

CSocketImp	一些网络能力的具体实现，被 CSocketMgr 类持有
CSocketMgr	一个统筹管理包内其它类的类
KeepAliveGuard	维持和后端心跳连接的线程
ReceiveGuard	接收来自后端消息的线程

1.1.6.com.frm.proto（协议包）

PDTO_Common	通用的结构体，可以配合 PID，组合使用
PDTO_Empty	消息/协议包的无扩展结构
PDTO_TICKER	与后端保持连接的心跳包结构
PDTO	消息/协议包的基类
PdtoDispatch	消息/协议的注册与派发
PID	消息/协议 id，通过 id 注册以便能通知界面、数据更新

1.1.7.协议衍生包（双端都有这两个包）

包名	作用
com.frm.z_proto.pdto	前后端协议的数据结构对象，前后端一致
com.frm.z_proto.ui	前后端本地的 UI 消息所需要的结构体，放在这个包里面

1.1.8.com.frm.thread（线程包）

AbsThread	线程基类
AutoThread	构造时就启动的线程（没用到）

1.1.9.com.frm.util_res（资源使用工具包）

ImgUtils	ImageIcon 的构建工具
*.png	资源可以放这个包里，方便使用
ImgPath	存放路径常量

1.1.10. com.frm.utils（工具包）

DialogTipUtils	提示小弹窗工具
PrintUtils	打印工具
ScreenUtils	屏幕操作工具
StringUtils	字符串操作工具
TimeUtils	时间操作工具
LogUtils	打印日志工具，方便统一管理
MathUtil	数值相关的工具
UIHelper	UI 通用操作辅助类

1.1.11. com.frm.z_common_entity（复用实体包）

com.frm.z_common_entity
专门用于放复用的模块实体包。
其它的实体包建议命名：com.系统名.模块名_entity。

1.1.12. com.frm.zserver_connect（连接服务器模块）

ServerConnectCtrl	MVC 中的 Controller 模块
ServerConnectModel	MVC 中的 Model 模块
ServerConnectView	MVC 中的 View 模块，用于给用户输入服务器 ip 和端口进行连接

1.1.13. com.z.main（程序入口类）

Main.java	作为程序入口

1.1.14. com.z.ztest（练习测试包）

这个包是专门用来放练习测试代码的（因为过程中肯定会有需要测试的代码）

1.2. 服务端

为了方便，后端直接使用了前端大部分框架内容，所以只列出差异包与类：

1.2.1.com.frm.cfg（配置包）

IDbCfg	JDBC 数据库连接配置——暂时不用数据库
INetCfg	开放端口配置

1.2.2.com.frm.db（数据库包）

DbMgr	数据库的管理类
DbUtils	数据库工具类

1.2.3.com.frm.nets（服务端网络包）

3、ClientSocketImpl	客户端 socket 的实现，持有 GuardReceive 线程
1、GuardConnection	客户端连接守卫，负责接待客户端连接
4、GuardReceive	具备收发客户端协议能力
IDestroyable	实现此接口，需实现 destroy 方法（ClientSocketImpl、GuardReceive）
2、SSocketMgr	启动 ServerSocket 及 GuardConnection，维护所有客户端 socket 连接

1.2.4.com.frm.utils（工具包）

SocketUtil	Socket 相关信息工具类

1.2.5.com.frm.z_game_hall（游戏大厅）

GameHallCtrl	MVC 中的 Controller 模块
GameHallModel	MVC 中的 Model 模块
GameHallView	MVC 中的 View 模块，通过昵称进入后就会展示出来

1.2.6.com.frm.z_game_match（游戏匹配）

GameMatchCtrl	MVC 中的 Controller 模块，直接响应操作可以放这里
GameMatchModel	MVC 中的 Model 模块
GameMatchView	MVC 中的 View 模块，匹配中的成员会显示出来

1.2.7.com.frm.z_game_room_mgr（游戏房间管理）

GameRoomMgrCtrl	MVC 中的 Controller 模块
GameRoomMgrModel	MVC 中的 Model 模块
GameRoomMgrView	MVC 中的 View 模块，不使用时创建出来占位

1.2.8.com.frm.z_game_room_mgr.sub（游戏房间相关实现）

GamePlayerRobot	游戏玩家机器人，代替不在线玩家操作，代打
GameRoomSvc	各个独立房间，两两匹配形成一个房间

1.2.9.com.frm.zserver_start（启动服务器模块）

ServerStartCtrl	MVC 中的 Controller 模块
ServerStartModel	MVC 中的 Model 模块
ServerStartView	MVC 中的 View 模块，特意通过 ui 启动服务器监听客户端连接的功能

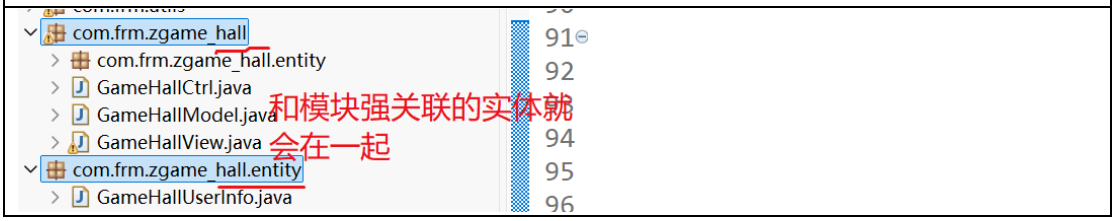
2. 创建一个模块的流程

2.1. 双端通用流程

2.1.1.模块包定义

模块包定义【推荐】
模板：sys.系统名.模块名
例子：sys.test.login

2.1.2.实体包定义

实体包定义【推荐】
模板：sys.系统名.模块名.entity
例子：sys.test.login.entity


2.1.3.模块类定义-M

模块类定义【推荐方式】
模板：模块名+Model
例子：LoginModel
必须继承 BaseModel，并实现，如下图：

LoginModel.java

```
1  package sys.test.login;
2
3  import com.frm.mvc.BaseModel;
5
6  public class LoginModel extends BaseModel{
7
8      @Override
9      protected void init() {
10         // TODO Auto-generated method stub
11
12     }
13
14     @Override
15     public void updateDt(PDTO pdto) {
16         // TODO Auto-generated method stub
17
18     }
19
20     @Override
21     protected void mClear() {
22         // TODO Auto-generated method stub
23
24     }
25
26 }
27
```

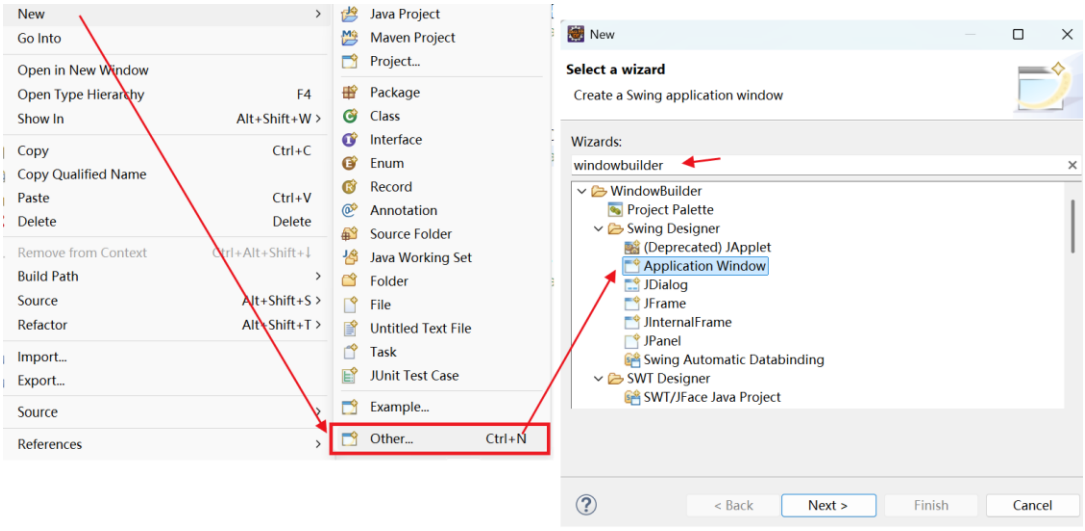
2.1.4.模块类定义-V

模块类定义【推荐方式】
模板：模块名+View

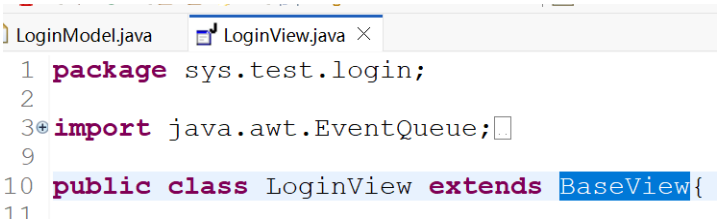
例子：LoginView

必须创建可视化界面，然后继承 BaseView，并实现，如下图：

1、创建可视化界面



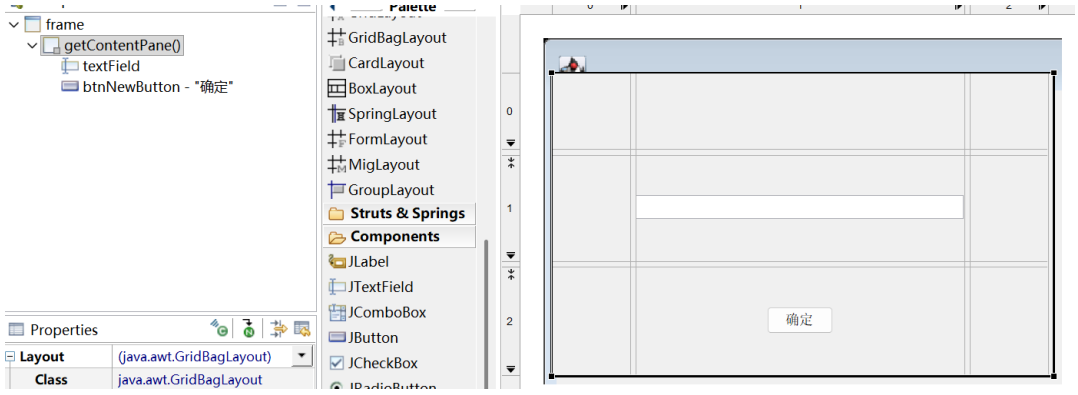
2、继承、实现、填写必要代码：



LoginModel.java LoginView.java

```
35      }
36
37      /**
38       * Initialize the contents of the frame.
39       */
40      private void initialize() {
44      }
45
46      @Override
47      protected void setVisible(boolean b) {
48          // TODO Auto-generated method stub
49          frame.setVisible(b);
50      }
51
52      @Override
53      public void updateView(PDTO pdto) {
54          // TODO Auto-generated method stub
55              //后续消息/协议更新派发的这里进行处理
56      }
57
58      @Override
59      protected void mClose() {
60          // TODO Auto-generated method stub
61          frame.dispose();
62      }
63
64  }
65
```

3、可视化编辑 UI 界面



2.1.5.模块类定义-C

模块类定义【推荐方式】
模板：模块名+Ctrl
例子：LoginCtrl
必须继承 BaseCtrl，并实现，如下图：

*LoginCtrl.java | OpenCloseSh... | OpenCloseSh... | OpenCloseSh...

```java
package sys.test.login;

import com.frm.mvc.BaseCtrl;

public class LoginCtrl extends BaseCtrl{
    @Override
    public void init() {
        // TODO Auto-generated method stub
        this.view = newView(LoginView.class);
        this.model = newModel(LoginModel.class);
    }

    @Override
    public PID[] registerPIDList() {
        // TODO Auto-generated method stub
        PID[] list = {
        };
        return list;
    }

    @Override
    public void updateModel(PDTO pdto) {
        // TODO Auto-generated method stub
    }

    @Override
    public void updateView(PDTO pdto) {
        // TODO Auto-generated method stub
    }

}

```

2.1.6.打开/关闭模块

打开模块（mvc 三部分都创建，其中 m 和 v 是在 init 时按照你的要求创建的）：
`MVC.openModule(LoginCtrl.class);`
关闭模块（mvc 三部分存在的都关闭）：

`btnNewButton_1 = new JButton("关闭界面");`
`btnNewButton_1.addActionListener(new ActionListener() {`
`    public void actionPerformed(ActionEvent e) {`
`        MVC.closeModule(LoginCtrl.class);`
`    }`
`});`

2.1.7.关闭/重新打开界面（保留 ctrl+model）

关闭界面（销毁界面，保留 controller + model，以便保存的数据被别处使用）：

```
MVC.closeView(LoginCtrl.class);
```

重新打开界面，依旧是用 openModule（controller + model 本来就保留着的）：

```
MVC.openModule(LoginCtrl.class);
```

2.1.8.隐藏/显示界面

隐藏界面（仅仅是简单地隐藏，不销毁界面）：

```
MVC.hideView(OpenCloseShowHideCtrl.class);
```

显示界面（仅仅是简单地将被隐藏的界面显示出来）：

```
MVC.showView(OpenCloseShowHideCtrl.class);
```

备注：调用 openModule 也可以将隐藏的界面显示出来：

```
MVC.openModule(LoginCtrl.class);
```

2.1.9.界面扩展参数

界面扩展参数：

使用示例（窗体可缩放+x 方向居中）：

```
ModelArgsExt args = new ModelArgsExt();
args.setResizable(true);
args.setLocPoint(ScreenUtils.screenWidth()/2, 0);
args.setLocationValType(UILocType.setX_frmW22___calcY);
MVC.openModule(LoginCtrl.class, ar
```

窗体大小可以鼠标拖拽缩放

com.frm.mvc.UILocType.setX_frmW22___calcY

主动设置的x - 窗体的宽/2 和 默认计算出来的y坐标 【x方向居中】

特别注意：

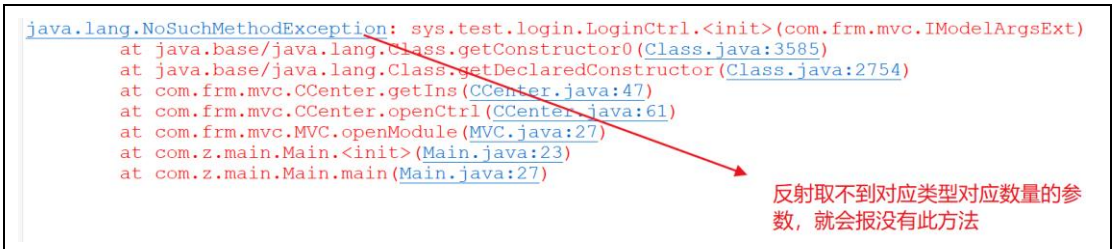
使用了 ModelArgsExt 参数后，对应的 Ctrl 类必须实现带参的构造方法，不然反射创建时会报错：

```
public class LoginCtrl extends BaseCtrl{
    public LoginCtrl(IModelArgsExt args) {
        super(args);
    }
    @Override
    public void init() {
```

必须实现带参构造函数

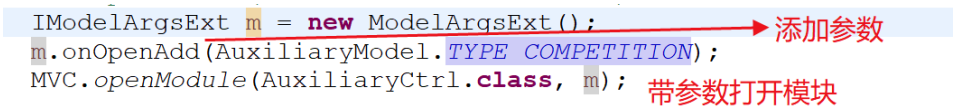
备注：构造函数快捷实现方式：Alt+Shift+S，选择 Generate Constructors From Superc……

未实现带参构造函数报错如下：



==========

onOpen 使用示例（携带参数到 onOpen 方法的参数列表）：



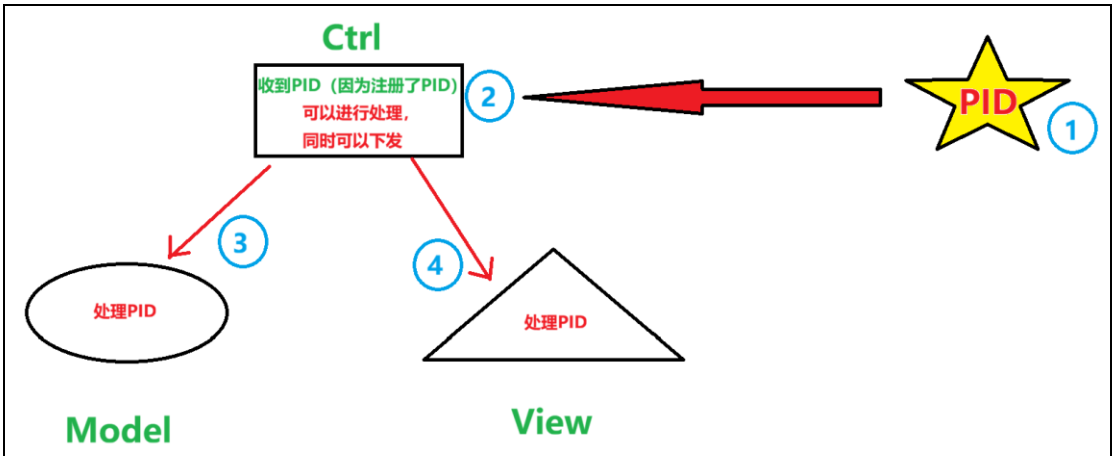
界面（View）的打开处（onOpen）使用：

2.1.10. 本地通过 PID 进行消息数据传递

2.1.10.1. 必须知道的

在本框架中，消息总是意味着一个对象【对象携带：唯一 PID+可选数据】
也就是说，虽然下方有各种重载变体方法，但是本质都一样，最终发出去时都会是一个带有 PID 的对象（数据可带可不带，即按需）。—— 重载那么多个方法只是为了方便。

PID 派发流程图：



代码体现：

GameMatchCtrl.java | BaseCtrl.java

```
132            }
133        }
134
135    //  /**
136    //   * enhance for sequence.
137    //   */
138    //  protected abstract void init();
139
140        public void updateSignal(PDTO pdto) {
141            this.updateCtrl(pdto); 1
142            if (isModelInit()) {
143                this.updateModel(pdto); 2
144            }
145            if (isViewInit()) {
146                this.updateView(pdto); 3
147            }
148        }
149        /**
150         * 必要时子类重写
151         * @param pdto
152         */
153        protected void updateCtrl(PDTO pdto) {
154
155        }
156
157    }
```

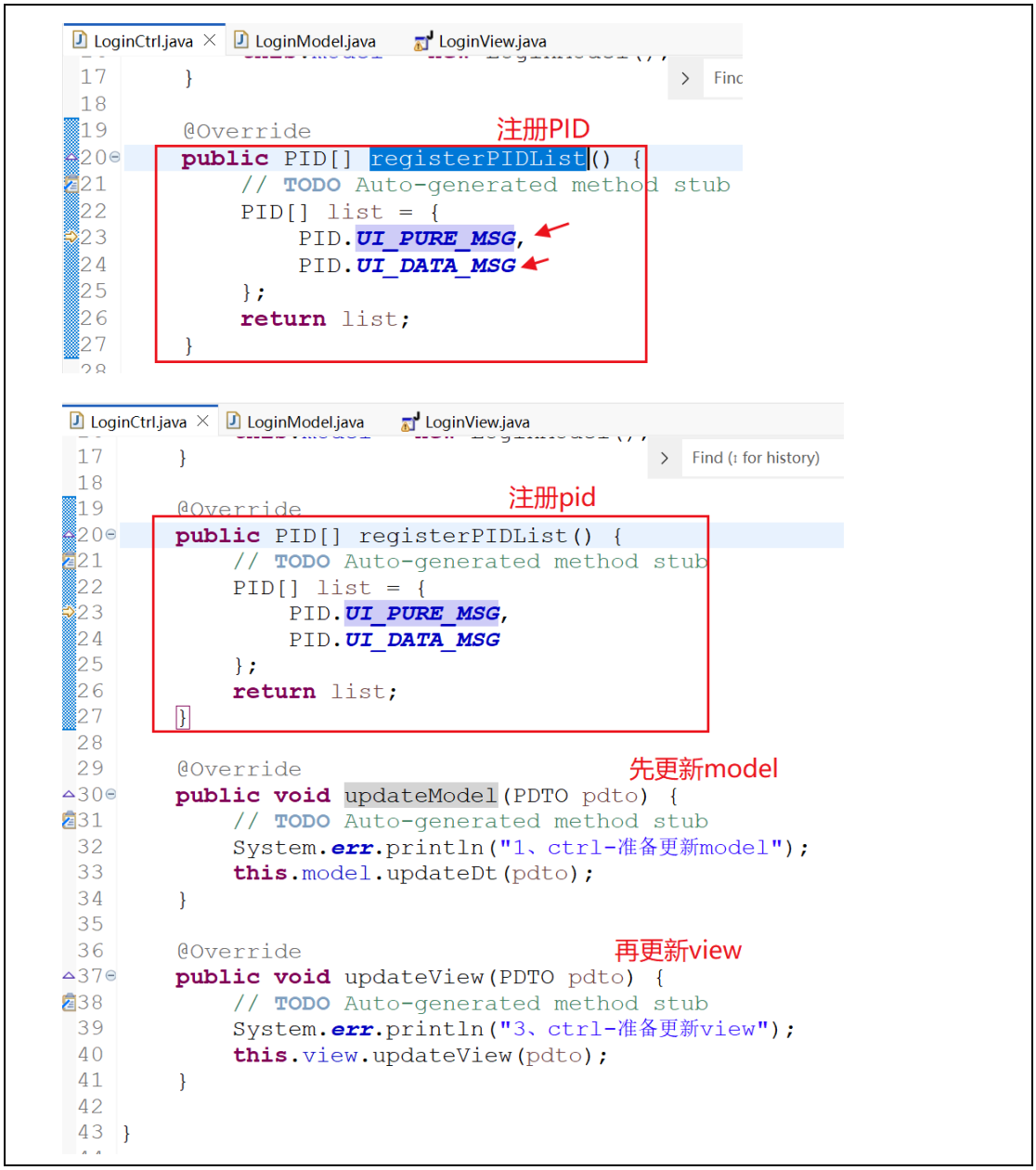
GameMatchCtrl.java × BaseCtrl.java

```
34      /**
35       * 必要时子类重写
36       *
37       * @param pdto
38       */                          1、处理需要立即响应的内容
39      protected void updateCtrl(PDTO pdto) {
40          switch (pdto.getPId()) {
41          case PID_REQ_GAME_MATCH:
42              response(PID.PID_REQ_GAME_MATCH, pdto);
43              break;
44          }
45      }
46
47      @Override                    2、更新数据
48      public void updateModel(PDTO pdto) {
49          // TODO Auto-generated method stub
50          this.model.updateDt(pdto);
51      }
52
53      @Override          3、更新界面-可能依赖model数据
54      public void updateView(PDTO pdto) {
55          // TODO Auto-generated method stub
56          this.view.updateView(pdto);
57      }
58
59  }
```

1、 增加 PID（可以理解为定义一个事件 id）

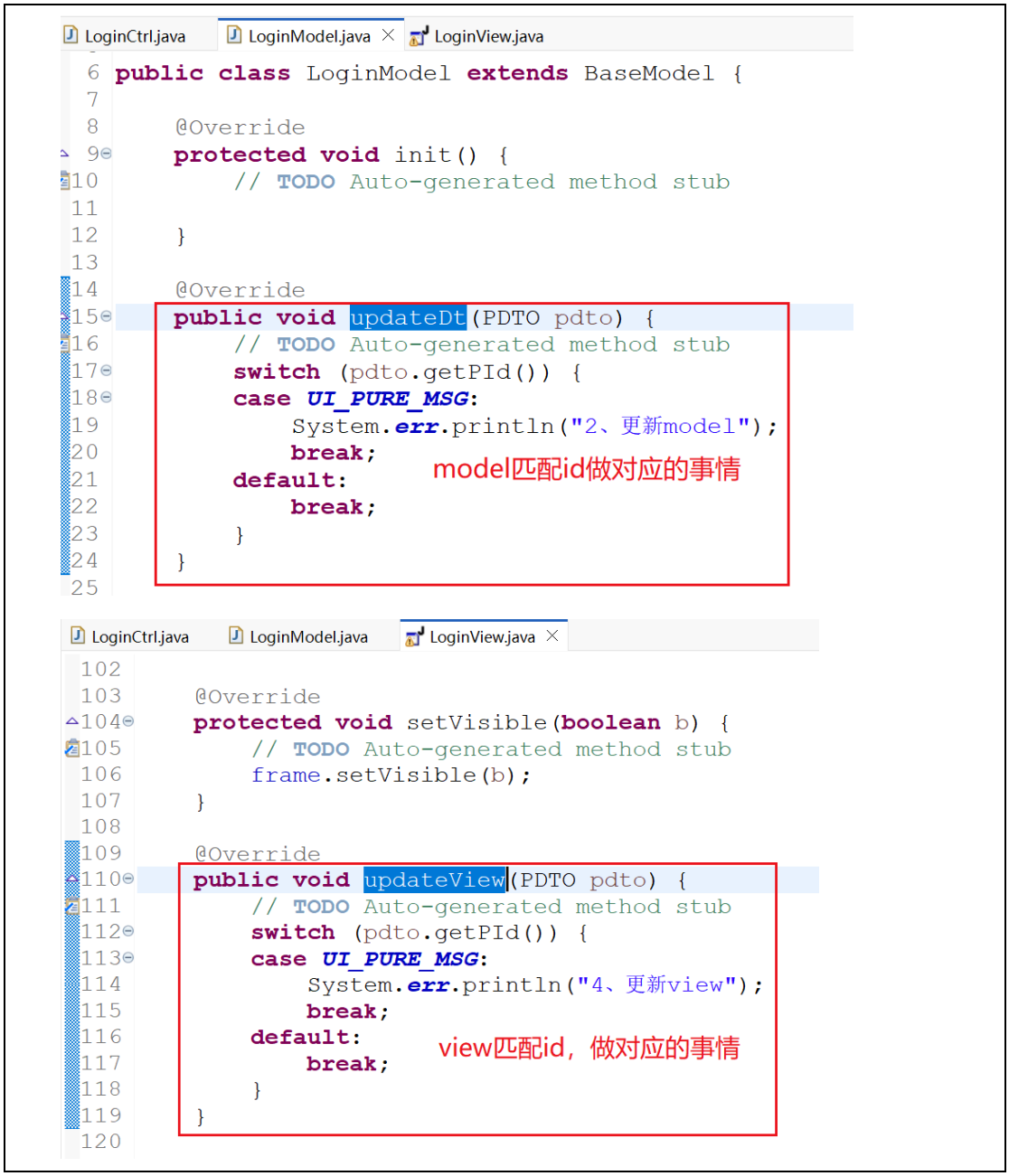
```
 9  public enum PID {
10      PID_ZERO, //作为默认值使用，无意义
11      PID_TICKER,
12      MAX_REMAIN_PID, //保留字，不要在上面加
13
14  //  ----------------------后续为自定义
15      UI_PURE_MSG, //ui-纯消息
16      UI_DATA_MSG, //ui-带数据消息
17  }                                定义2条
18
19
```

2、 需要监听的人就要监听事件 id，并给出事件 id 发生时，要怎么做




```
17      }
18
19      @Override
20      public PID[] registerPIDList() {
21          // TODO Auto-generated method stub
22          PID[] list = {
23              PID.UI_PURE_MSG,
24              PID.UI_DATA_MSG
25          };
26          return list;
27      }
```

```
17      }
18
19      @Override
20      public PID[] registerPIDList() {
21          // TODO Auto-generated method stub
22          PID[] list = {
23              PID.UI_PURE_MSG,
24              PID.UI_DATA_MSG
25          };
26          return list;
27      }
28
29      @Override
30      public void updateModel(PDTO pdto) {
31          // TODO Auto-generated method stub
32          System.err.println("1、ctrl-准备更新model");
33          this.model.updateDt(pdto);
34      }
35
36      @Override
37      public void updateView(PDTO pdto) {
38          // TODO Auto-generated method stub
39          System.err.println("3、ctrl-准备更新view");
40          this.view.updateView(pdto);
41      }
42
43  }
```



2.1.10.2. 纯消息——仅 PID（无需数据）

基类实现的发送 UI 消息方法：

```
 * 仅要id
 * @param emptyStruct
 */
protected void sendUIMsg(PID emptyStruct) {
    MVC.sendUIMsg(emptyStruct);
}
```

子类需要的时候调用：

LoginModel.java　LoginView.java

```
JButton btnNewButton = new JButton("确定");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //TODO - 在此添加点击事件处理
        sendUIMsg(PID.UI_PURE_MSG);
    }
});
```

纯id消息，不用带数据

调用后效果:

Problems　Javadoc　Declaration　Search　Console　Progress　Call Hierarchy

Main [Java Application] D:\S_Software\E_EclipseJEE\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x8

```
msgUI=> (UI_PURE_MSG)
1、ctrl-准备更新model
2、更新model
3、ctrl-准备更新view
4、更新view
```

2.1.10.3. 带数据的消息——唯一（PID+数据）

带数据的话，需要继承扩展一个结构体类:

```
1 package sys.test.aproto;
2
3 import com.frm.proto.PDTO;
4
5 public class PDTO_Content extends PDTO{
6     public String content = "";
7 }
8
```

这个变量用于存储数据

当然还可以按需要定义更多

基类实现的发送 UI 消息方法:

```
/**
 * 独一无二的，id和结构都不同
 * @param uniStruct
 */
protected static void sendUIMsg(PDTO uniStruct) {
    MVC.sendUIMsg(uniStruct);
}
```

子类使用示例:

va LoginModel.java LoginView.java MVCBaseComm.java PDTO_Content.java

```
JButton btnNewButton = new JButton("确定");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //TODO - 在此添加点击事件处理
        sendUIMsg(PID.UI_PURE_MSG);

        PDTO_Content dto = new PDTO_Content();
        dto.content = "我是带了数据的";
        dto.setPId(PID.UI_DATA_MSG);
        sendUIMsg(dto);
    }
});
```

数据 PID 发送

收到消息后的处理:

CtrlJava LoginModel.java LoginView.java MVCBaseComm.java PDTO_Content.java

```
}

@Override
public void updateDt(PDTO pdto) {
    // TODO Auto-generated method stub
    switch (pdto.getPId()) {
    case UI_PURE_MSG:
        System.err.println("2、更新model");
        break;
    case UI_DATA_MSG:
        System.err.println("2、更新model: " + ((PDTO_Content)pdto).content);
        break;
    default:
        break;
    }
}
```

收到PID 做对应的操作 能取到数据

view同样可以做这样的操作（前提是ctrl注册了PID不要忘了）

控制台结果:

```
msgUI=> (UI_DATA_MSG)
1、ctrl-准备更新model
2、更新model: 我是带了数据的
3、ctrl-准备更新view
4、更新view: 我是带了数据的
```

2.1.10.4. 带数据的消息——组合（PID+复用对象）

适合能够**复用**的 PDTO 类型，特点是会在内部设置 pid（不理解看看源码就懂了）
使用例子:

LoginCtrl.java | LoginModel.java | LoginView.java | MVCBaseComm.java | PDTO_Content.java

```
77          JButton btnNewButton = new JButton("确定");
78          btnNewButton.addActionListener(new ActionListener() {
79              public void actionPerformed(ActionEvent e) {
80                  //TODO - 在此添加点击事件处理
81  //              ================纯消息
82  //              sendUIMsg(PID.UI_PURE_MSG);
83  //              ================带数据消息——唯一==========
84                  PDTO_Content dto = new PDTO_Content();
85                  dto.content = "我是带了数据的";
86  //              dto.setPId(PID.UI_DATA_MSG);
87  //              sendUIMsg(dto);
88  //              ================带数据消息——组合复用==========
89                  sendUIMsg(PID.UI_DATA_MSG, dto);
90              }
91          });
```

2.1.10.5. 剩余的变体

sendUIMsg（发送了会有一点点延迟执行）：

```
/**
 * 同构，id不同
 * @param pId
 * @param str
 */
protected void sendUIMsg(PID pId, String str) {
    MVC.sendUIMsg(pId, str);
}
/**
 * 同构，id不同
 * @param pId
 * @param num
 */
protected void sendUIMsg(PID pId, int num) {
    MVC.sendUIMsg(pId, num);
}
```

都是内部使用了通用的结构体，直接传具体的类型

sendUIMsgQuick（发送了立刻执行）：

```java
/**
 * 独一无二的，id和结构都不同
 * @param uniStruct
 */
protected void sendUIMsgQuick(PDTO uniStruct) {
    MVC.sendUIMsgQuick(uniStruct);
}
/**
 * 仅要id
 * @param emptyStruct
 */
protected void sendUIMsgQuick(PID emptyStruct) {
    MVC.sendUIMsgQuick(emptyStruct);
}

/**
 * 同构，id不同
 * @param pId
 * @param commStruct
 */
protected void sendUIMsgQuick(PID pId, PDTO commStruct) {
    MVC.sendUIMsgQuick(pId, commStruct);
}


/**
 * 同构，id不同
 * @param pId
 * @param str
 */
protected void sendUIMsgQuick(PID pId, String str) {
    MVC.sendUIMsgQuick(pId, str);
}
/**
 * 同构，id不同
 * @param pId
 * @param num
 */
protected void sendUIMsgQuick(PID pId, int num) {
    MVC.sendUIMsgQuick(pId, num);
}
```

内部通用结构体，直接传具体的值

2.1.11. Model 侧重数据逻辑

模块 M\V\C 三部分都能取到，可以简单地理解为都是单例，本处重点讲 Model。

C 部分

这一部分主要职责是调控作用，尽量不写逻辑。
【能取，但是建议少取来用】

V 部分

这一部分主要是专注于界面处理。
【能取，但是不建议取】

M 部分

这一部分，专注于数据存储 + 数据有关的操作
【建议用，建议一些自身数据 + 数据相关的逻辑可以写在这里】
详细使用查看《获取模块数据》

2.2. 获取模块数据

数据一般存储在 Model 里面，我们可以在任何模块任何地方方便地取到里面的数据。下面举个例子：

需求：想要在【GameMatchModel】取到《GameHallModel》里面保存的昵称。 1、 首先《GameHallModel》要提供公开接口。 2、 然后通过 MVC.getModel 方法，在【GameMatchModel】中取到【GameHallModel】实例。 3、 通过【GameHallModel】实例，调用公开的接口，最终获得想要的数据。
首先《GameHallModel》要提供公开接口：

GameMatchModel.java | GameHallModel.java | GameMatchView.java | PID.java

```
70
71      public String getUserNickName(int hsCode) {
72          GameHallUserInfo v = _userMap.get(hsCode);
73          if(v == null) {
74              return "unknown";
75          }
76          return v.getNickName();
77      }
78
```

然后通过 MVC.getModel 方法，在【GameMatchModel】中取到【GameHallModel】实例。

GameMatchModel.java | GameHallModel.java | GameMatchView.java | PID.java | GameMatchCtrl.java | GameHallCtrl.j

```
43      private void updateUserMap(PDTO pdto){
44          GameHallUserInfo v = this._userMap.get(pdto.getsHsCode());
45          if(v != null) {
46  //          do sth.
47  //          清理旧的逻辑
48          }else {
49              v = new GameHallUserInfo();
50          }
51          int key = pdto.getsHsCode();
52          GameHallModel model = MVC.getModel(GameHallCtrl.class, GameHallModel.class);
53          String nickName = model.getUserNickName(key);
54          v.setNickName(nickName);
55          v.setHashcode(key);
56          this._userMap.put(key, v);
57      }
```

通过【GameHallModel】实例，调用公开的接口，最终获得想要的数据。

GameMatchModel.java | GameHallModel.java | GameMatchView.java | PID.java | GameMatchCtrl.java | GameHallCtrl.j

```
43      private void updateUserMap(PDTO pdto){
44          GameHallUserInfo v = this._userMap.get(pdto.getsHsCode());
45          if(v != null) {
46  //          do sth.
47  //          清理旧的逻辑
48          }else {
49              v = new GameHallUserInfo();
50          }
51          int key = pdto.getsHsCode();
52          GameHallModel model = MVC.getModel(GameHallCtrl.class, GameHallModel.class);
53          String nickName = model.getUserNickName(key);
54          v.setNickName(nickName);
55          v.setHashcode(key);
```

2.3. 客户端发协议

2.3.1.sendPackMsg

在按钮事件中发送一条协议，例子如下：

2.3.1.1.定义 PID

```
public enum PID {
    PID_ZERO, //作为默认值使用，无意义
    PID_TICKER,
    MAX_REMAIN_PID, //保留字，不要在上面加
//  ================================新增协议，
    PID_NICK_NAME, //设置昵称
//  ================================新增协议，
```

2.3.1.2.发送协议包

```
btnEnter.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String nick = tfNick.getText();
        if(nick.isEmpty() || nick.isBlank()) {
            DialogTipUtils.simpleWarnTip(Char._20);
            return;
        }
        发送昵称
        MVC.sendPackMsg(PID.PID_NICK_NAME, nick);
        MVC.openModule(LoginCtrl.class);
        MVC.closeModule(ServerConnectCtrl.class);
    }
});
```

2.3.1.3.发送完毕

后端会收到想要的协议包，然后后端会进行响应的处理，会按需返回给前端。

2.3.1.4.MVC 类中所有重载方法/变体

```
/**
 * 独一无二的，id和结构都不同
 * @param uniStruct
 */
public static void sendPackMsg(PDTO uniStruct) {
    CSocketMgr.cs.send(uniStruct);
}
/**
 * 仅要id
 * @param emptyStruct
 */
public static void sendPackMsg(PID emptyStruct) {
    PDTO_Empty pdto = new PDTO_Empty();
    pdto.setPId(emptyStruct);
    CSocketMgr.cs.send(pdto);
}
```

```
/**
 * 同构，id不同
 * @param pId
 * @param commStruct
 */
public static void sendPackMsg(PID pId, PDTO commStruct) {
    commStruct.setPId(pId);
    CSocketMgr.cs.send(commStruct);
}
```

```java
    /**
     * 同构，id不同
     * @param pId
     * @param str
     */
    public static void sendPackMsg(PID pId, String str) {
        PDTO_Common commStruct = new PDTO_Common();
        commStruct.str = str;
        commStruct.setPId(pId);
        CSocketMgr.cs.send(commStruct);
    }
    /**
     * 同构，id不同
     * @param pId
     * @param num
     */
    public static void sendPackMsg(PID pId, int num) {
        PDTO_Common commStruct = new PDTO_Common();
        commStruct.num = num;
        commStruct.setPId(pId);
        CSocketMgr.cs.send(commStruct);
    }
```

2.3.1.5.MVCBaseComm 中同样有以上变体封装

模块的 M、V、C 部分都能更方便地调用，详见《MVCBaseComm》，部分例子如下：

LoginView.java

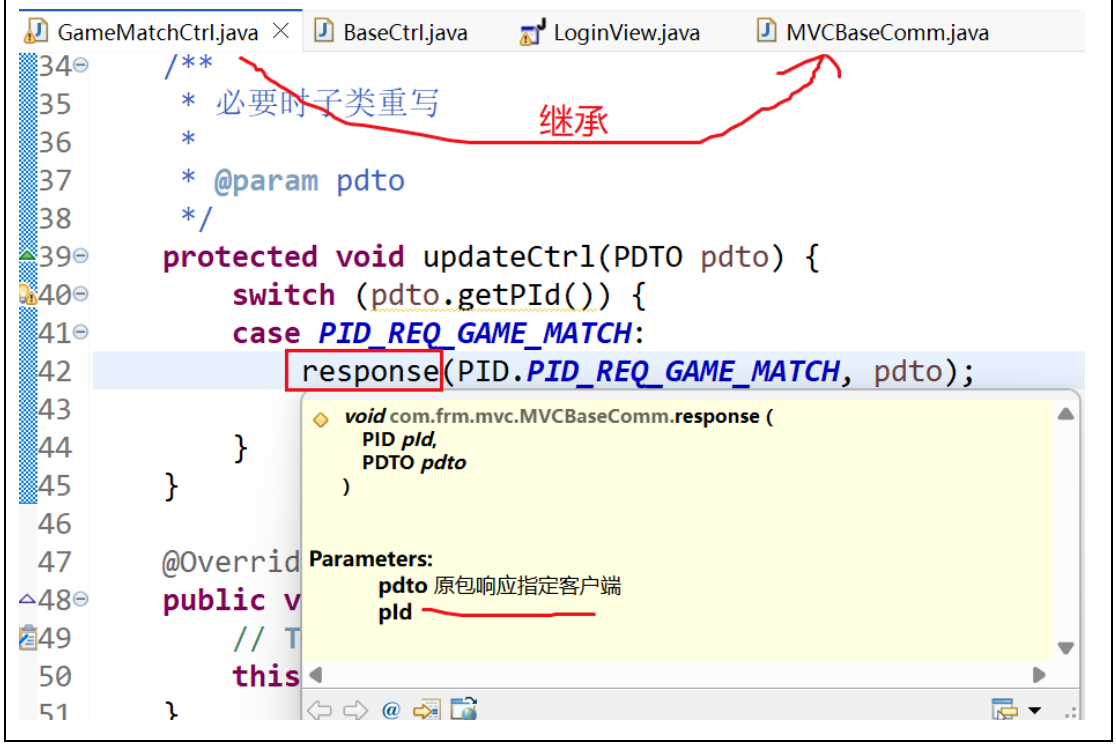
```java
tnNewButton_2 = new JButton(Char._21);
tnNewButton_2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        sendPackMsg(PID.PID_REQ_GAME_MATCH);
    }
);
ridBagC
bc_btnN
bc_btnN
bc_btnN
rame.ge
```

```
void com.frm.mvc.MVCBaseComm.sendPackMsg (
    PID emptyStruct
)

仅要id

Parameters:
    emptyStruct
```

2.4. 服务器端响应协议

2.4.1.原包响应指定 - response

response 可以指定 PID 然后原包内容返回。
PS：继承了 MVCBaseComm 的可以直接调用 response().



2.4.2.响应指定用户 - responseWithSpecifiedUser

responseWithSpecifiedUser

有多个变体，详细看源码

说明：

```
void com.frm.mvc.MVC.responseWithSpecifiedUser (
    PID pId,
    String str,
    int sHsCode
)

Parameters:
        pId
        pdto - 通用字符串
        sHsCode 响应指定用户
        str
```

例子：

GameRoomMgrModel.java GameMatchModel.java GameRoomSvc.java

```java
package com.frm.z_game_room_mgr.sub;

import com.frm.mvc.MVC;
import com.frm.proto.PID;

public class GameRoomSvc {
    /**
     * 房间大小
     */
    public static int ROOM_SIZE = 2;
    private String _roomKey;
    private int[] roomMemb = new int[GameRoomSvc.ROOM_SIZE];
    public void init(String roomKey, int hsCode1, int hsCode2) {
        // TODO Auto-generated method stub
        _roomKey = roomKey;
        roomMemb[0] = hsCode1;
        roomMemb[1] = hsCode2;
        MVC.responseWithSpecifiedUser(PID.PID_ENTER_ROOM, _roomKey, hsCode1);
        MVC.responseWithSpecifiedUser(PID.PID_ENTER_ROOM, _roomKey, hsCode2);
    }

}
```

2.4.3.响应指定用户数组 - responseWithSpecifiedUsers

responseWithSpecifiedUsers

3 个变体

说明：

```
void com.frm.mvc.MVC.responseWithSpecifiedUsers (
    PID pId,
    PDTO pdto,
    int[] sHsCodes
)

Parameters:
    pId
    pdto
    sHsCode 响应指定用户们
    sHsCodes
```

例子:

GameRoomMgrCtrl.java | GameRoomSvc.java

```
0
1     private void timeHandler(long sec) {
2         PDTO_RoomGaming p = new PDTO_RoomGaming();
3         p.roomTimeSec = sec;
4         if(sec == 0) {
5 //            说明开始
6             p.stage = 0;
7         }
8         else if(sec == GameRoomSvc.RESULT_TIME) {
9 //            结算
0             p.stage = 1;
1             Map<Integer, Integer> resultMap = _resultMap;
2             for(int id: roomMemb) {
3                 if(!resultMap.containsKey(id)) { //随机帮玩家取一个，如果玩家没有选
4                     resultMap.put(id, GamePlayerRobot.getIns().getRandomVal());
5                 }
6             }
7             p.dict = resultMap;
8         }else {
9 //            同步计时
0             p.stage = 2;
1         }
2         MVC.responseWithSpecifiedUsers(PID.PID_S2C_ROOM_GAMING, p, roomMemb);
3     }
```

2.4.4.广播给所有用户

相关响应函数都在服务端的 MVC 类可以看到

MVC.java

```
218
219     /**
220      * @param pdto 广播给所有客户端
221      */
222     public static void broadcast(PID pId, PDTO pdto) {
223         pdto.setPId(pId);
224         SSocketMgr.ss.broadcast(pdto);
225     }
226     /**
227      * @param pdto 广播给所有客户端 - 通用字符串
228      */
229     public static void broadcast(PID pId, String str) {
230         PDTO_Common pdto = new PDTO_Common();
231         pdto.str = str;
232         pdto.setPId(pId);
233         SSocketMgr.ss.broadcast(pdto);
234     }
235     /**
236      * @param pdto 广播给所有客户端 - 通用数值
237      */
238     public static void broadcast(PID pId, int num) {
239         PDTO_Common pdto = new PDTO_Common();
240         pdto.num = num;
241         pdto.setPId(pId);
242         SSocketMgr.ss.broadcast(pdto);
243     }
244
```

3. 开发一个模块的流程

一、需求文档+UI 设计
二、定协议/定数据包结构
三、后端根据协议编写
四、客户端根据协议编写
五、双端联调
六、联调完毕，测试模块，模块完成。
详细如下：

10_开发一个模块的流程.txt

文件 编辑 查看 H1 B

零、需求文档+UI设计

一、定协议/定数据包结构：
游戏开始
s2c —— 后端通知
复用【*】
游戏进行
s2c —— 下发过程数据【*】
int stage;
long roomTimeSec;
Map dict; ——用户数据
c2s —— 选择数值
int —— 复用通用结构
游戏结算
s2c —— 下发结算数据
复用【*】

一-1、代码增加协议。
协议前后端要一致，一端定好，另一端复制。
一-2、代码增加所需结构体。
结构体前后端要一致，一端定好，另一端复制。

二、后端根据协议编写
后端维护处理数据，向前端发数据。

++++++continue：

三、客户端根据协议编写
根据后端数据，进行相关前端处理展示。

四、双端联调
如果有需要修改再修改。

五、联调完毕，测试模块，模块完成。

4. 常用介绍

4.1. JDK 常用

4.1.1.GridBagLayout

https://blog.csdn.net/Huanzhi_Lin/article/details/156099652

容器设置 gridbag 布局。

容器内组件使用 gridbag 约束。

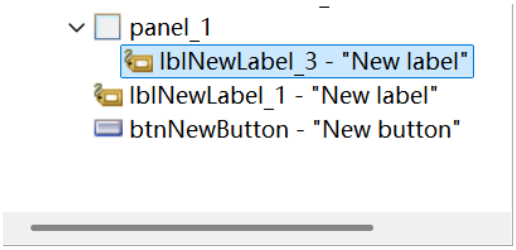
容器设置：

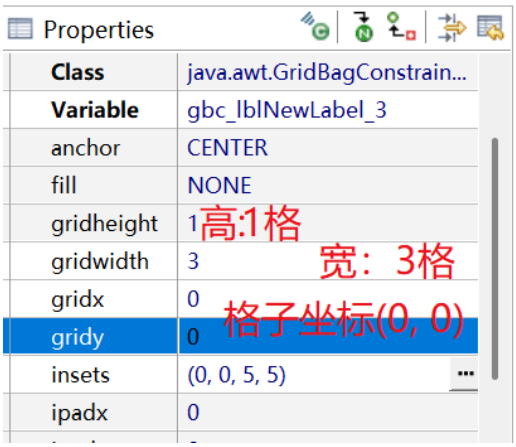
```
gbl_panel_1.columnWidths = new int[]{0, 0, 0};
gbl_panel_1.rowHeights = new int[]{0, 0, 0};
gbl_panel_1.columnWeights = new double[]{1,1,1};
gbl_panel_1.rowWeights = new double[]{1,1,1};
```

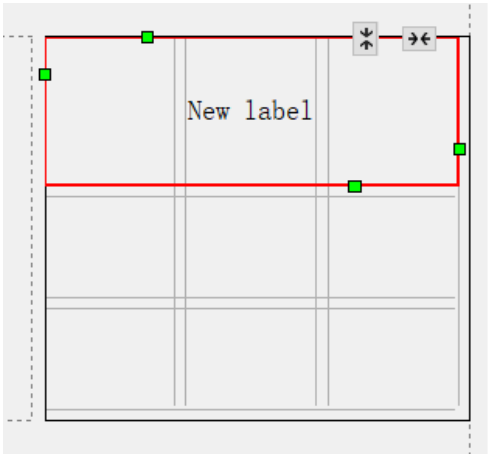
3列3行

每行每列占比为1/3

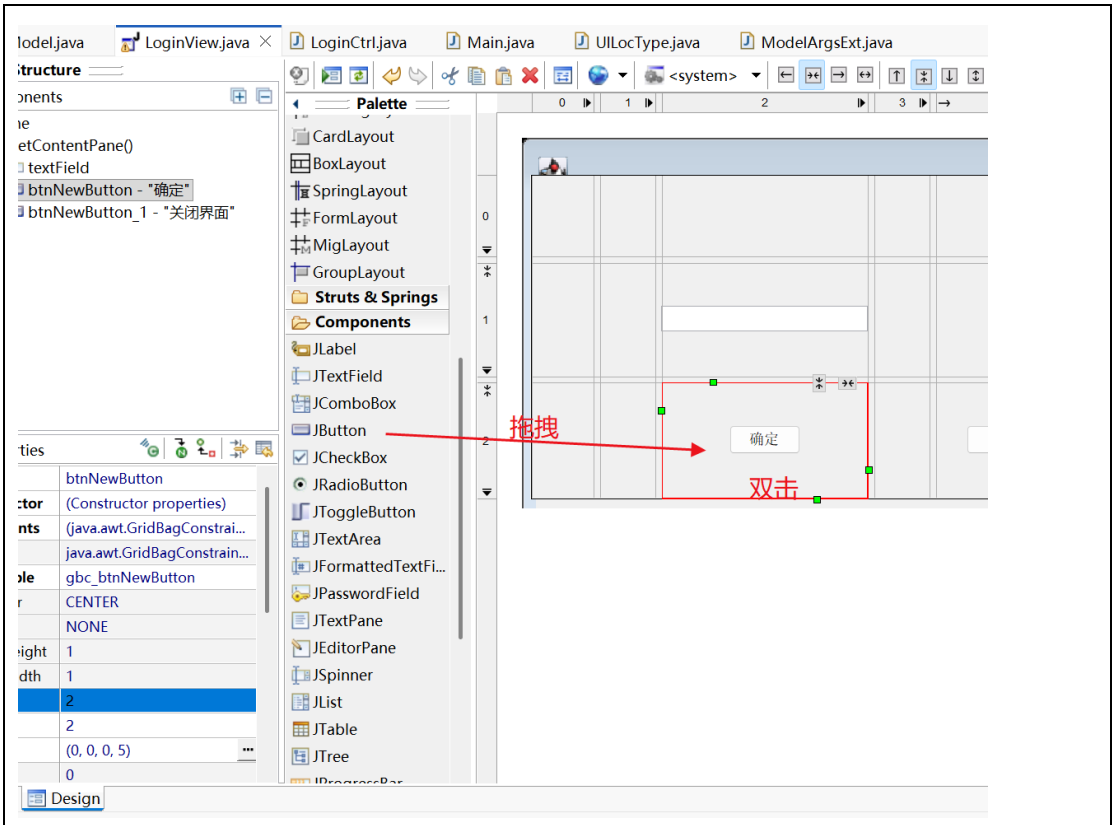
容器中组件设置（右图为效果）：







4.1.2.添加按钮事件




```
JButton btnNewButton = new JButton("确定");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //TODO - 在此添加点击事件处理
    }
});
```

4.1.3.窗体各个时机监听接口

实现接口:

```
import java.awt.event.ActionEvent;

public class LoginView extends BaseView implements WindowListener{

```

可以在各个时机做一些处理:

```java
@Override
public void windowOpened(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowClosing(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowClosed(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowIconified(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowDeiconified(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowActivated(WindowEvent e) {
    // TODO Auto-generated method stub
}
@Override
public void windowDeactivated(WindowEvent e) {
    // TODO Auto-generated method stub
}
```

记得加这一行，不然监听不生效!!!:

```java
frame.addWindowListener(this);
```

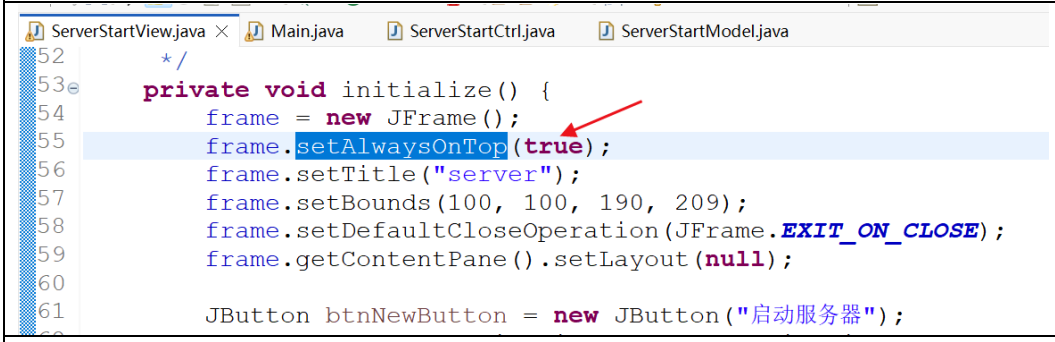
或者还可以使用 WindowAdapter 实现等价效果:

```java
// 使用WindowAdapter，只需重写需要的方法
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        handleWindowClosing();
    }

    @Override
    public void windowClosed(WindowEvent e) {
        System.out.println("窗口已完全关闭");
    }
});
```

4.1.4.界面始终在最顶层

始终在最顶层的界面不会被别的界面挡住：setAlwaysOnTop()；方式 1 和 2 用一种即可。

方式 1：



方式 2（推荐）：

```
IModelArgsExt args = new ModelArgsExt();
args.setLocationValType(UILocType.R_TOP);
args.setOnTop(true);
MVC.openModule(ServerStartCtrl.class, args);
```

4.1.5.设置界面关闭的触发类型

4.1.5.1.什么都不做（Nothing）——不推荐

1、【内存不自动释放、程序不退出】设置此选项后，窗口关闭操作不会执行任何默认动作。窗口会保持打开状态，允许开发者通过窗口监听器（如 WindowAdapter）自定义关闭逻辑，例如显示确认对话框。

```
/**
 * The do-nothing default window close operation.
 */
public static final int DO_NOTHING_ON_CLOSE = 0;
```

适用于自定义关闭逻辑情景：

```java
JFrame frame = new JFrame("自定义关闭");
frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        // 自定义关闭逻辑
        int option = JOptionPane.showConfirmDialog(frame, "确定要退出吗? ", "确认",
            JOptionPane.YES_NO_OPTION);
        if (option == JOptionPane.YES_OPTION) {
            frame.dispose();
            System.exit(0);
        }
    }
});
```

不推荐，除非有特殊的需求。使用这个记得在 windowClosing 里面按照需求选择调用《隐藏界面》or《销毁/关闭界面》

4.1.5.2.隐藏（Hide）——不释放资源(推荐 2

2、【不释放资源、不退出程序】当用户关闭窗口时，窗口会隐藏（即调用 setVisible(false)），但窗口对象仍存在。再次显示时，窗口内容不会重置，适用于需要保留状态的场景。

```java
/**
 * The hide-window default window close operation
 */
public static final int HIDE_ON_CLOSE = 1;
```

适用于反复显示隐藏的窗口：

```java
// 工具窗口示例
public class ToolWindow extends JFrame {
    public ToolWindow() {
        setTitle("工具窗口");
        setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
        // 可以重复显示隐藏
    }
}

// 使用
ToolWindow tool = new ToolWindow();
tool.setVisible(true);
// 用户关闭窗口后，可以再次调用 tool.setVisible(true) 显示
```

使用这种方式，只需要设置为 HIDE_ON_CLOSE 即可。需要再次打开时，只需要调用《隐藏/显示界面》即可将隐藏的界面显示出来。

4.1.5.3.销毁（Dispose）——释放资源

3、【释放资源、不退出程序】关闭窗口时，不仅隐藏窗口，还会释放其占用的系统资源（如 AWT 资源）。窗口对象被销毁，无法直接重新显示，适用于一次性使用的对话框。

```
/**
 * The dispose-window default window close operation.
 * <p>
 * <b>Note</b>: When the last displayable window
 * within the Java virtual machine (VM) is disposed of, the VM may
 * terminate.  See <a href="../../java/awt/doc-files/AWTThreadIssues.html">
 * AWT Threading Issues</a> for more information.
 * @see java.awt.Window#dispose()
 * @see JInternalFrame#dispose()
 */
public static final int DISPOSE_ON_CLOSE = 2;
```

适用于多（子）窗口界面：

```
// 多文档界面示例
public class MDIApplication {
    private JDesktopPane desktop;

    public void createDocumentWindow() {
        JInternalFrame docFrame = new JInternalFrame("文档", true, true, true, true);
        // 内部窗口的默认关闭操作
        docFrame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        desktop.add(docFrame);
        docFrame.setVisible(true);
    }
}
```

注意-本框架中使用这种方式，只要设置 DISPOSE_ON_CLOSE 即可，但是记得通过《窗体各个时机监听接口》清理掉引用，不然没办法按预期销毁释放界面（因为 CCenter 有界面引用），使用例子如下：

GameHallVie... | Main.java | GameHallCtr... | ServerStart... | MVC.java | CCenter.java

```
22 public class GameHallView extends BaseView implements WindowListener{
23
```

GameHallVie... | Main.java | GameHallCtr... | ServerStart... | MVC.java | CCenter.java

```
55     private void initialize() {
56         frame = new JFrame();
57         frame.setBounds(100, 100, 450, 300);
58 //      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
59         frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
60         frame.addWindowListener(this);
```

4.1.5.4.关闭程序（Exit）——推荐 1

4、【释放所有资源、退出程序】关闭窗口时，终止当前 Java 应用程序(即调用 System.exit)。通常用于独立应用程序的主窗口，关闭主窗口即退出整个程序。

```
/**
 * The exit application default window close operation. Attempting
 * to set this on Windows that support this, such as
 * <code>JFrame</code>, may throw a <code>SecurityException</code> based
 * on the <code>SecurityManager</code>.
 * It is recommended you only use this in an application.
 *
 * @since 1.4
 * @see JFrame#setDefaultCloseOperation
 */
public static final int EXIT_ON_CLOSE = 3;
```

```
// 主应用程序窗口
public class MainApplication extends JFrame {
    public MainApplication() {
        setTitle("主程序");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(800, 600);

        // 主窗口关闭时退出整个程序
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new MainApplication().setVisible(true);
        });
    }
}
```

这个最方便，不需要做任何额外处理，创建 View 默认就是设置了这种方式。

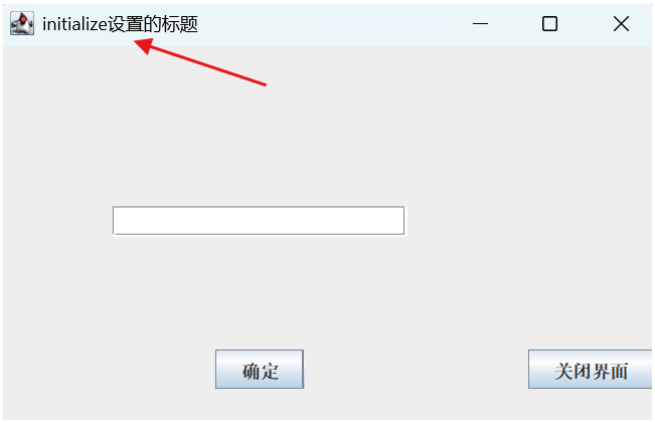
4.1.6.设置标题

通过 windowbuilder 创建后，在自动生成 initialize 方法那设置标题

```
LoginModel.java  LoginView.java  MVCBaseComm.java  PDTO_Content.java  BaseView.jav
 */
private void initialize() {
    frame = new JFrame();
    frame.setTitle("initialize设置的标题");
    frame.setBounds(100, 100, 450, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    GridBagLayout gridBagLayout = new GridBagLayout();
```

这一行是手动加的

运行效果:



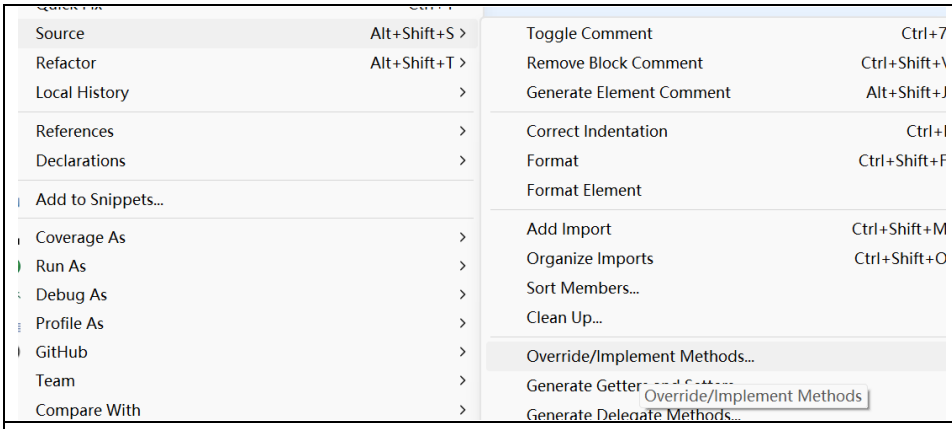
4.2. 框架生命周期

4.2.1.View 生命周期函数

4.2.1.1.打开-onOpen()

```
oginCtrl.java  LoginModel.java  LoginView.java  BaseView.java  MVCBaseComm.java
    /**
     * @param params
     * @description 界面打开调用方法，用于子类重写
     */
    protected void onOpen(ArrayList<Object> params) {

    }
```

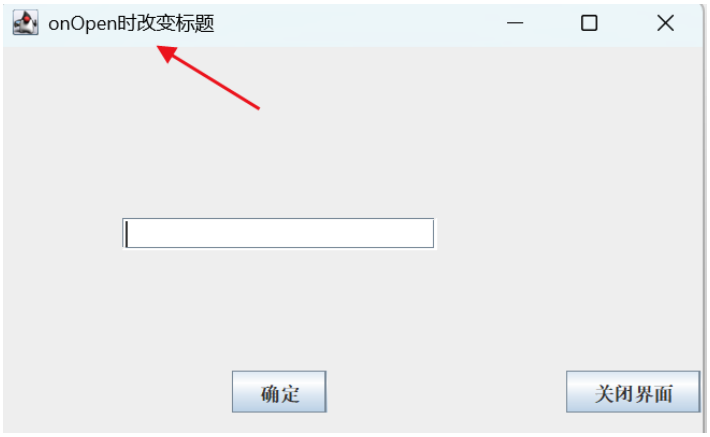
重写/实现基类方法快捷方式:



例子:

```
141
142     @Override
143     protected void onOpen(ArrayList<Object> params) {
144         // TODO Auto-generated method stub
145         super.onOpen(params);
146         frame.setTitle("onOpen时改变标题");
147     }
148
```

运行效果:



4.2.1.2.关闭-mClose()

作用是关闭界面时清理需要清理的东西（如：xxx = null; 之类的）

```
136     @Override
137     protected void mClose() {
138         // TODO Auto-generated method stub
139         frame.dispose();
140     }
141
```

4.2.1.3.frame 可见设置-setVisible()

决定界面是否可见，默认是可见的，这里需要写这个关联起来

```
114
115     @Override
116     protected void setVisible(boolean b) {
117         // TODO Auto-generated method stub
118         frame.setVisible(b);
119     }
```

4.2.1.4.界面更新-updateView()

注册的 PID 触发后会到这里，决定对应 PID 如何处理。

```
120
121     @Override
122     public void updateView(PDTO pdto) {
123         // TODO Auto-generated method stub
124         switch (pdto.getPId()) {
125         case UI_PURE_MSG:
126             System.err.println("4、更新view");
127             break;
128         case UI_DATA_MSG:
129             System.err.println("4、更新view: " + ((PD
130             break;
131         default:
132             break;
133         }
134     }
```

4.2.2.Model 的生命周期函数

```
1  public abstract class BaseModel extends MVCBaseComm{
2
3      public BaseModel() {
4          // TODO Auto-generated constructor stub
5          this.init();
6      }
7
8      protected abstract void init();
9
10     /**
11      * @param pdto
12      * @return
13      * @description 更新数据
14      */
15     public abstract void updateDt(PDTO pdto);
16     protected abstract void mClear();
17
18 }
```

必要的初始化

更新model数据

清理必要东西

4.2.3.Controller 的生命周期函数

ClientSocketImpl.java | GuardReceive.java | LoginCtrl.java

```
7
8  public class LoginCtrl extends BaseCtrl{
9      public LoginCtrl(IModelArgsExt args) {
0          super(args);
1      }
2      public LoginCtrl() {
3          super();
4      }
5      @Override
6      public void init() {  必要初始化
7          // TODO Auto-generated method stub
8          this.view = newView(LoginView.class);
9          this.model = newModel(LoginModel.class);
0      }
1
2      @Override
3      public PID[] registerPIDList() {注册PID
4          // TODO Auto-generated method stub
5          PID[] list = {
6              PID.UI_PURE_MSG,
7              PID.UI_DATA_MSG
8          };
9          return list;
0      }
1
2      @Override                          更新引用的model
3      public void updateModel(PDTO pdto) {
4          // TODO Auto-generated method stub
5          System.err.println("1、ctrl-准备更新model");
6          this.model.updateDt(pdto);
7      }
8
9      @Override                                  更新引用的view
0      public void updateView(PDTO pdto) {
1          // TODO Auto-generated method stub
2          System.err.println("3、ctrl-准备更新view");
3          this.view.updateView(pdto);
4      }
5
6  }
```

4.3. 框架自定义布局

4.3.1.默认是居中随机偏移

设计目的，避免多个界面打开后全部重合，界面被盖住。

BaseCtrl.java | IModelArgsExt.java | ModelArgsExt.java

```
            int h = jf.getHeight();
            double viewLocScaleX = 0.5;
            double viewLocScaleY = 0.3;
            int screenW = ScreenUtils.screenWidth();
            int screenH = ScreenUtils.screenHeight();
            int x = (int) ((screenW - w) * viewLocScaleX + offset);
            int y = (int) ((screenH - h) * viewLocScaleY + offset);
            boolean r = false;
            if(args != null) {
                r = args.isCanResizable();
                jf.setResizable(r);
                jf.setLocation(args.getLocPoint(x, y, w, h, screenW, screenH));
                jf.setAlwaysOnTop(args.getOnTop());
            }else { //默认
                jf.setResizable(r);
                jf.setLocation(x, y);
            }
        }
    }
```

4.3.2.让界面靠在四个角

优先级比较高，设置了四个角的就忽略其它布局设置了（可自行修改）

枚举类型:

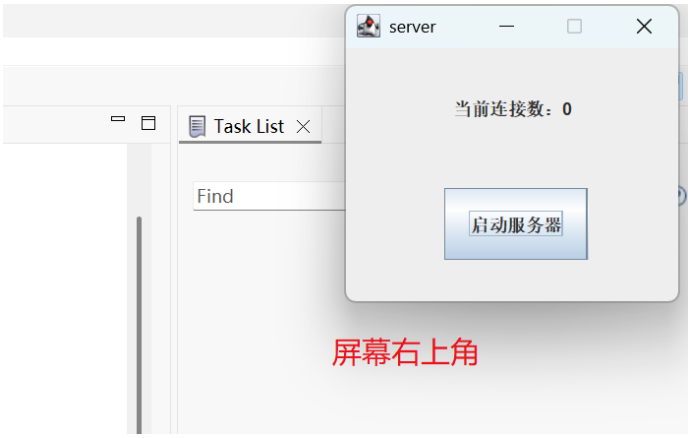
```
public enum UILocType {
    /**
     * 右top
     */
    R_TOP,
    /**
     * 右bottom
     */
    R_BOTTOM,
    /**
     * 左top
     */
    L_TOP,
    /**
     * 左bottom
     */
    L_BOTTOM,
//  ==================上面高优先级================
```

例子：

```
IModelArgsExt args = new ModelArgsExt();
args.setLocationValType(UILocType.R_TOP);
args.setOnTop(true);
MVC.openModule(ServerStartCtrl.class, args);
```

右上角

效果：



4.3.3.进行自定义操作

有点不太好理解，没理解可以自己设计修改：

具备类型：

```
    setX_frmW___setY,
    setX___setY,
    /**
     * 主动设置的x + 窗体宽   和   主动设置的Y
     */
    setX__frmW___setY,
    /**
     * 主动设置的x - 窗体的宽/2   和 默认计算出来的y坐标
     * 【x方向居中】
     */
    setX_frmW22___calcY,
```

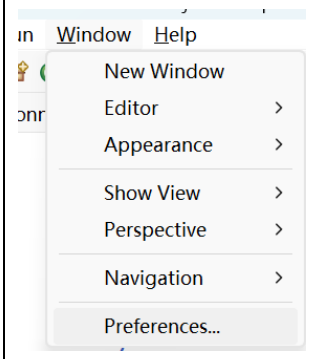
使用例子（配合 setLocPoint 使用）：

```
args.setLocPoint(ScreenUtils.screenWidth()/2, 0);
args.setLocationValType(UILocType.setX_frmW22___calcY);
MVC.openModule(GlobalObserverCtrl.class);
MVC.openModule(ServerConnectCtrl.class, args);
```

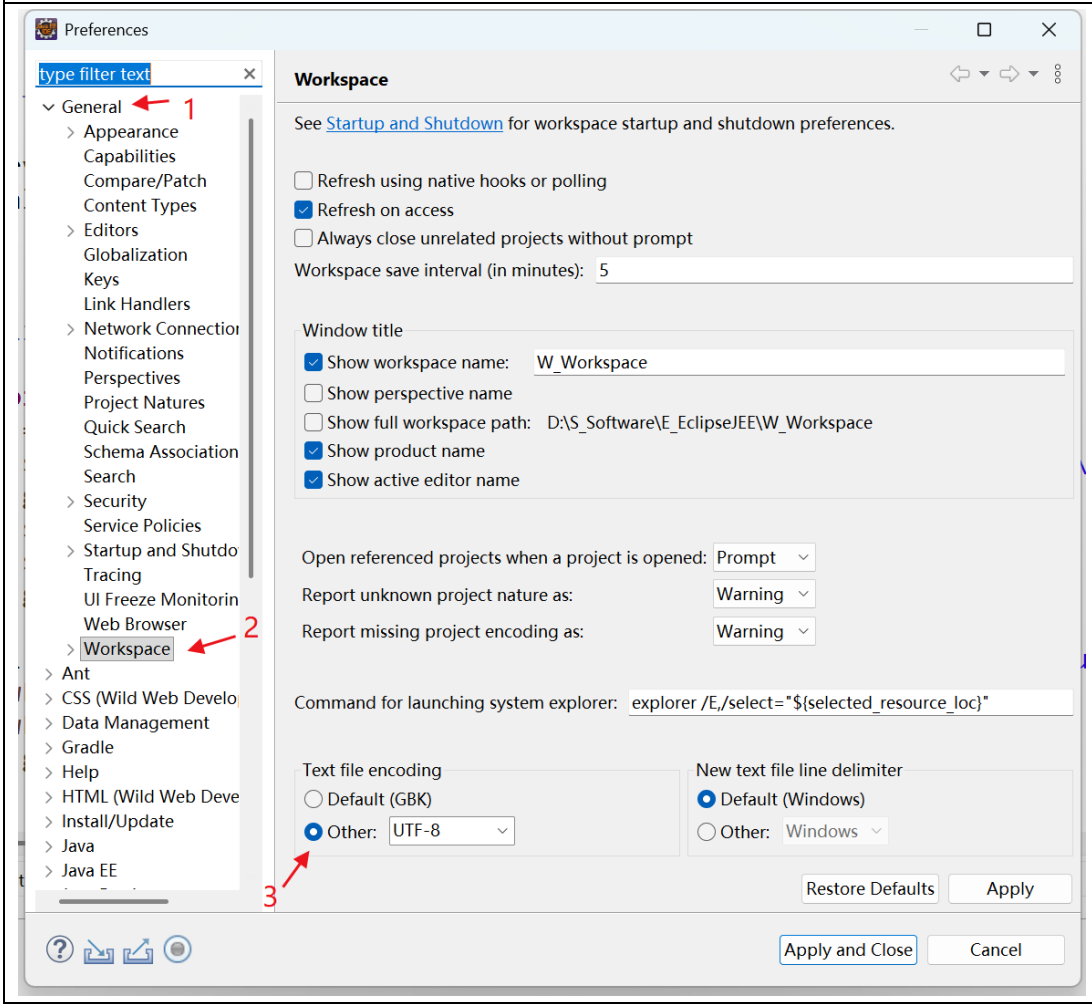
5. 注意事项

5.1. IDE 的编码——UTF-8

（如果出现乱码，请设置编码）



选择 Preferences



6. 考虑扩展

6.1. BaseAloneView

可以做，但是暂时用不到先不做的：
BaseAloneView： 1、 通过静态导入方案得到一个常量对应一个类。 2、 需要一个 ViewMgr 类进行管理，就不要和 CCenter 混合在一起了。 3、 可以独立进行监听消息（不依赖于 Ctrl）。 4、 使用组合代替继承，实现独立监听消息等额外的功能（看情况是继承 BaseView 称为 BaseAloneView 还是直接 BaseView 里面通过组合方式拓展）。 5、 可以一个 View 类多次实例化。 6、 可以配合 MVC + BaseAloneView 使用，达到一些特殊需求效果。

```java
public final class ViewClasses {
    private ViewClasses() {} // 防止实例化

    public static final Class<LoginView> LoginView = LoginView.class;
    public static final Class<MainView> MainView = MainView.class;
    public static final Class<SettingsView> SettingsView = SettingsView.class;
    // 添加更多视图...
}

// Step 2: 静态导入
import static com.yourpackage.ViewClasses.*;
```

7. 获取最新发布