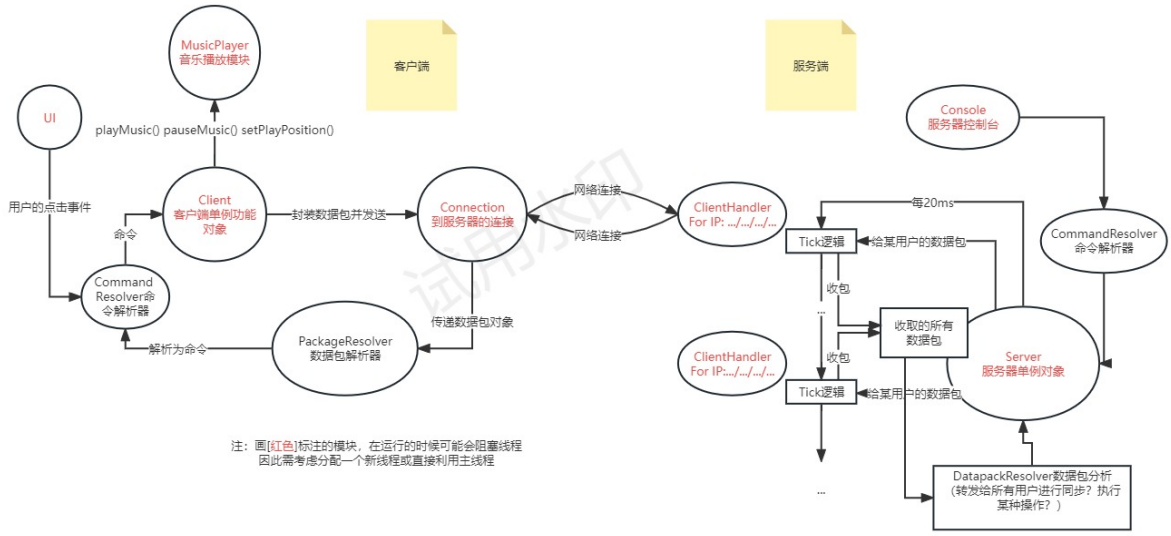
开发标准约定

- 2023/11/1 日 23:33 创建版
- Java JDK版本：`1.8.0_392`
- 构建工具`Maven 3.9.2`
- 推荐IDE:`IntelliJ IDEA`

- 分工
  音乐获取与播放 卫佳欣
  UI 郭立杰
  系统 王玮宁
  架构 朱玉林

基本软件架构



命名标准

- 变量一律使用**全名**，（除**世界公认**的缩写以外）

```
Thread serverThread; //√
Thread serThr; //X

String num; //√
String number; //√

User admin;//√
User administrator;//√
```

- 禁止**单一字母**，**字母加数字变量**（除短声明周期变量，遍历序号（某些情况下也不能使用字母））

```
User u1; //X
User administrator; //√

try{
    //...
}catch(Exception e){
    //...
}
//√
```

- 禁止**拼音命名。**

```
String yonghuming;
//X
String userName;
//√
```

- **禁止**`Util`类，`Tool`类。类名必须见名知意。

```
public class Utils{
    public static void playMusic(){
        //...
    }

    public static void receiveMessage(){
        //...
    }
}
//X
```

```
public class Music{
    public static void playMusic(){
        //...
    }
}

public class ChatRoom{
    public static void receiveMessage(){
        //...
    }
}
//√
```

- 变量命名必须是有**明确意义的名词**，方法命名必须是有**明确意义的动词。**

```
class Test{
    public void help(){
        //X
    }

    public void vertifyUser(){
        //√
    }
}
```

- 变量命名，方法命名均使用**驼峰命名法**，即小写字母开头，其他单词开头字母为大写。
- 常量命名，枚举类型命名**全大写。**
- 类名**帕斯卡命名法**，即单词首字母均大写。
- 包名**全小写。**
- 接口类在类名前加`I`标明接口。`IMusicPlayer`，`IDataSaver`。

编写规范

- **以方法最外层缩进为参考，缩进不得超过五层**

```
public void func(){
    //此处为0层缩进
    System.out.println("Running");
    if(){
        //1
        if(){
            //2
            if(){
                //3
                if(){
                    //4
                    if(){
                        //此处为5层缩进，再缩进即违反规则。
                    }
                }
            }
        }
    }
}
```

- 验证过程使用**提前退出**的方法，不要使用无意义的`else`分支。尽量减少缩进。

```
public Socket connect(String ip, int port){
    if(!Vertifier.isValidIP(ip)){
        System.out.println("Invalid IP");
        return null;
    }else{
        if(!Vertifier.isValidPort(port)){
            System.out.println("Invalid Port");
            return null;
```

```
        }else{
            //...
        }
    }
}
//X

public Socket connect(String ip, int port){
    if(!Vertifier.isValidIP(ip)){
        System.out.println("Invalid IP");
        return null;
    }
    if(!Vertifier.isValidPort(port)){
        System.out.println("Invalid Port");
        return null;
    }

    //...
}
//√
```

- 涉及到**大量字符串加减**运算，一律使用`StringBuilder`而非`String`。

```
String a = "";
for(var c : str.toCharArray()){
    a += c;
}
//X

StringBuilder a = new StringBuilder();
for(var c : str.toCharArray()){
    a.append(c);
}
//√
```

- 存在可以用`switch`语句判断的方法就一定要用`switch`，不要使用`if-else`

```
switch(type){
    case A:
    case B:
        //...
}
//√

if(type == E.A){
    //...
}else if(type == E.B){
    //...
}else if(type == E.C){
    //...
}
//X
```

- 如果存在`ErrorLog`等错误处理，错误记录，服务器信息类，必须使用相应方法而不是`System.out.println()`。
- `ErrorLog`，`Log`，`Warn`等必须输出信息**类型**，**线程类型，信息时间**，必要情况下需要输出信息源ip，用户名。如果遇到严重的`Exception`异常，需要打印调用堆栈信息。

```
[ERROR/SeverThread][00:00:00] User GLJ is trying to kill administrator, which is not allowed.
```

```
public static void main(String[] args) {
    try{
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

- **禁止**忽略`Exception`，不进行处理或者记录就继续运行。
- **禁止**方法根据传入的`boolean`，`enum`确定基本运行模式，而是使用重载或者命名不同。

```
public void save(SaveType type){
    if(type == SaveType.JSON){

    }else{

    }
}
//X

public void jsonSave(){

}

public void otherSave(){
    //...
}

//√
```

- 不允许类中出现过多与类毫不相关的辅助方法，做到类功能的单一化。

```
public class Server{
    public void saveFile(){
        //...
    }

    public Music getMusic(){
        //...
    }
}
//X
```

```
public class Files{
    public void saveFile(){}
}

public class Music{
    public Music getMusic(){
    }
}
//√
```

- **提交修改时除特殊情况外禁止保留大段注释掉的代码。**

```
public void playMusic(){
    Music mus = Music.getMusic();
    mus.play();

    // Music mus = getMusic();
    // MusicPlayer.playMusic(mus);
    // 不知道为什么这个实现不了，烦死了
}
// X
```

- 不要保留**不可到达的代码**

```
public void function(){
    if(){
        //...
    }else{
        //...
    }
    return;

    for(int i = 0; i < 5; i++){
        //...
    }
    //X
}
```

- 方法**不得过长**，尽量拆分功能。

- 如果重写`equals()`方法，必须重写`hashCode()`方法。

- 禁止滥用单例模式，**除逻辑上理应只能有一个存在的类。**

```
public class User{
    private static User instance;
}
//X

public class Server{
    private static Server instance;
}
//√
```

- 只需要使用静态方法的类必须私有化构造方法。

```
public class Vertifier{
    private Vertifier(){}
}
//√
```

PS

- 不要关掉我的MC服务器