## HR服务平台需求分析文档

# 系统介绍

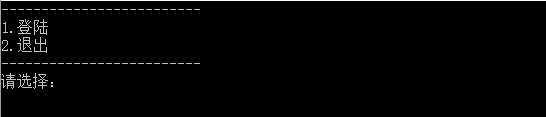
## 系统背景

说到找工作，我们自然想到去跑人才市场，求职者需要人头攒动的摊位前递上自己的简历，用工单位则要租用场地来设点，无论对于求职者，还是对用工单位来说，都是耗时耗力，消耗成本事情。那么，HR服务平台则让它变的简单，只需要一个服务平台，即可以实现用工单位发布职位需求信息。

## 系统介绍

1. 用户打开HR服务平台系统，首先进行身份信息验证即登录操作，只有登录成功之后才能进行一系列的功能操作。(HR登陆用户名与密码:HR，123)

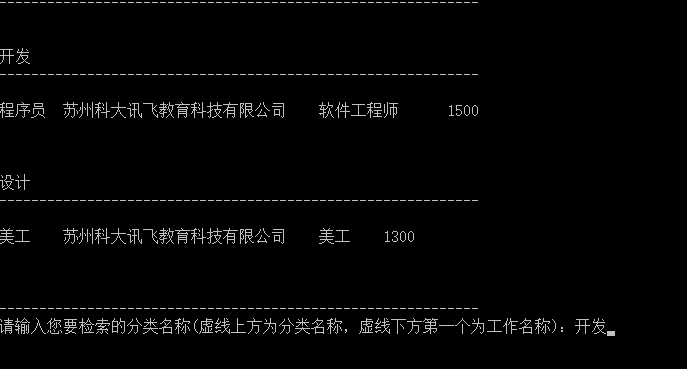
如图所示：

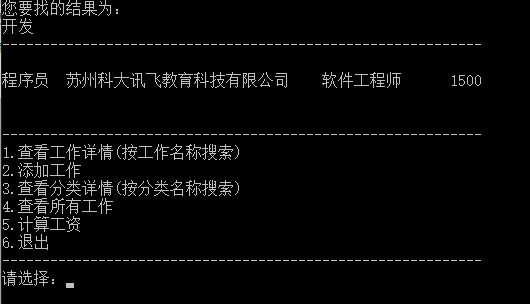


当登录成功后则显示菜单下的所有功能模块：

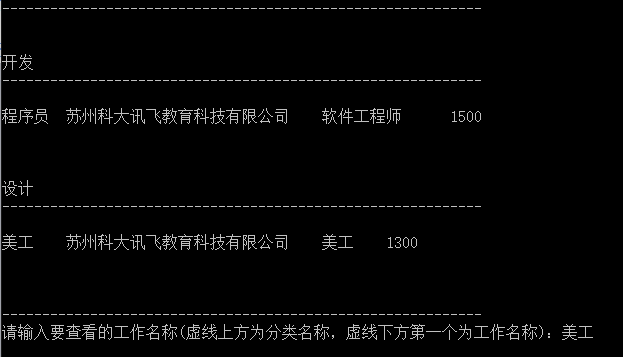


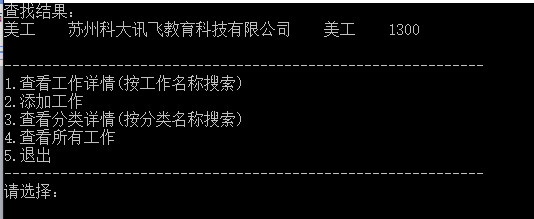
1. 对于每个工作分类，HR能够选择分类名字来查看这个分类下面的所有的工作。如图所示：





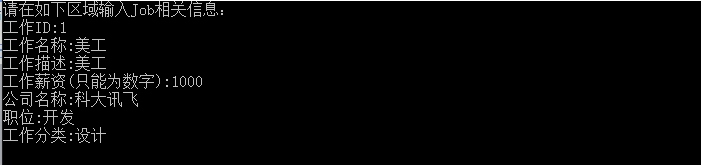
1. HR可以有搜索已发布工作的功能，系统则显示工作详细信息，包括职位、薪资、公司名称等信息。如图所示：





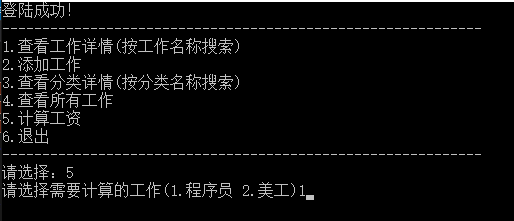
1. HR登录到系统，发布工作，需要填写工作相关信息，包括工作ID、工作名称、公司的名字、职位、分类、工作描述以及如何申请等信息。

如图所示：



1. HR登录到系统，可以选择计算工资模块功能，即通过工作类别输出不同工作类别下的工作工资。

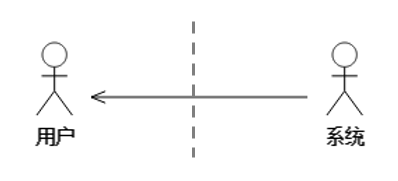
如图所示：





# 用户登录到系统角色分析

从上面的系统的简单分析来看，本系统相对简单，涉及到的角色只有系统角色（HR），简单起见，并做归一化处理，把系统卷角色（HR）定义成一种称谓，就是用户，在使用这个系统。在过程中，用户和系统之间的关系用简单的图形表现如下：

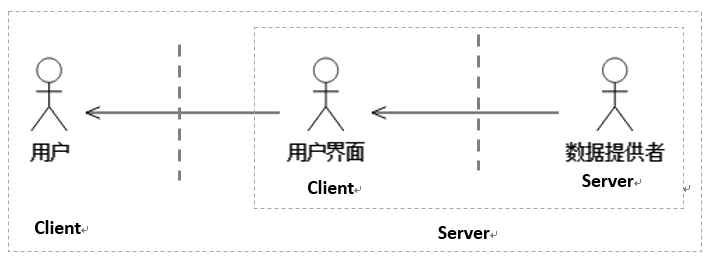


在这个只有一类角色的关系中，我们可以把系统看作是系统服务提供方，英文表达叫做Server，用户看作是使用服务的对象，也叫客户端，英文表达叫做Client，所以用户使用系统这个例子，告诉我们在这个场景中可以归纳出Client和Server这样的关系。

现在我们撇开“用户”这个角色，单看“系统”这个角色，系统需要提供用户界面给用户，比如首页上的按照分类方式组织的工作列表或者某个工作的全部详细内容等用户界面。那么从这个意义上来说，“系统”这个角色其实可以拆分成用户界面显示角色，称为UI，也就是UserInterface的简称。我们知道，界面上内容的展示需要数据，界面只是负责把数据以某种比较优雅和优好的方式来展示出来，界面本身是没有数据的或者说它本身不持有数据，它在展示数据的时候，需要有一个角色来为它提供数据。这样，我们需要一个角色来完成数据的提供，姑且称为数据提供者，英文表达DataProvider。当然，数据提供者持有数据，这些数据可以是简单的数据或是复杂的结构性数据，也可以是来自网络等数据。所以，对于界面角色(UI)来说，它只关心的是如何把数据渲染出来形成界面，它并不关心数据真正从哪儿来。而数据从哪儿来，是数据提供者才需要关心的事情。那么，UI和DataProvider的关系可以用下面的图来简单的表达。同样，我们把用户界面称为Client，而数据提供者称为Server。



这样，把用户这个角色加入进来，则整个关系用下面的图来表达。



确定出系统的各个组成部分和各个组成部分的角色（Server/Client），还要分析出随着时间的变化，系统中角色的具体的信息是哪些，那么，我们借助于一种UML工具将这个过程表达出来作为后续业务实现的依据，那就是时序图。

# 角色功能分析

|  |  |  |
| --- | --- | --- |
| 对象 | 功能 |  |
| HR用户 | 登录系统 | √ |
| 发布工作 | √ |
| 查看所有工作信息 | √ |
| 查看工作分类详情(根据分类名称查找) | √ |
| 查看工作详情(根据工作名称查找) | √ |
| 计算工资 | √ |

在前面我们理清了用户和系统的关系，也就是确定了系统的边界，也就是说，使用系统的用户是处于边界外面。那么现在来看看用户到底能干什么呢？

HR用户能做的事情是（功能列表）如下：

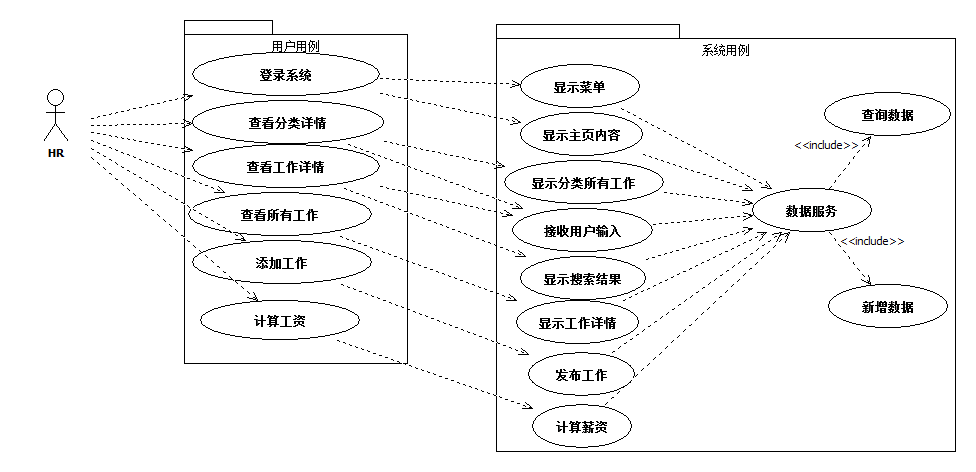
1. 登录到系统
2. 发布工作
3. 查看所有工作
4. 查看工作分类详情(根据分类名称查找)
5. 查看工作详情(根据工作名称查找)
6. 计算工资

# 系统UML设计

根据上面的角色分析和角色的功能分析，我们借助比较专业的建模工具UML来表征和表达，对于角色的功能，可以使用用例图来表达，对于角色功能交互，可以使用时序图来表达。

## 用例图

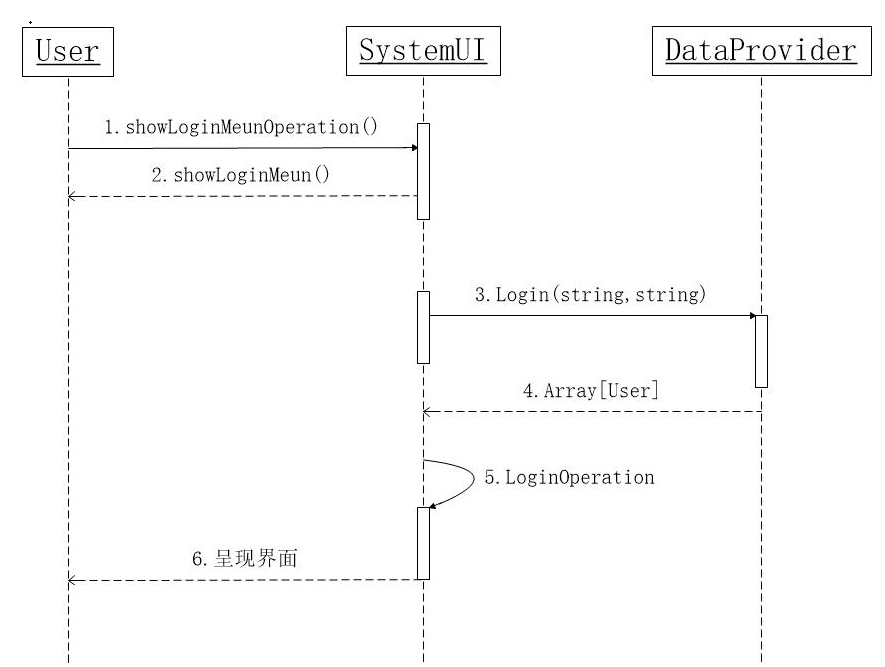
用例图主要用来表达什么人在什么地方干什么事情，这里的什么人，在用例图中指参与者(Actor)，什么地方指系统边界，干什么事情指用例(Use Case)，其主要作用就是用来对系统进行、子系统进行功能行为分析并建模。结合系统角色功能分析，借助于StarUML工具，用户使用系统的用例图如下：



## 系统时序图

根据系统的业务流程，结合用例图，我们将为每个用例和业务创建时序图，以帮助理解系统事件的发生过程。

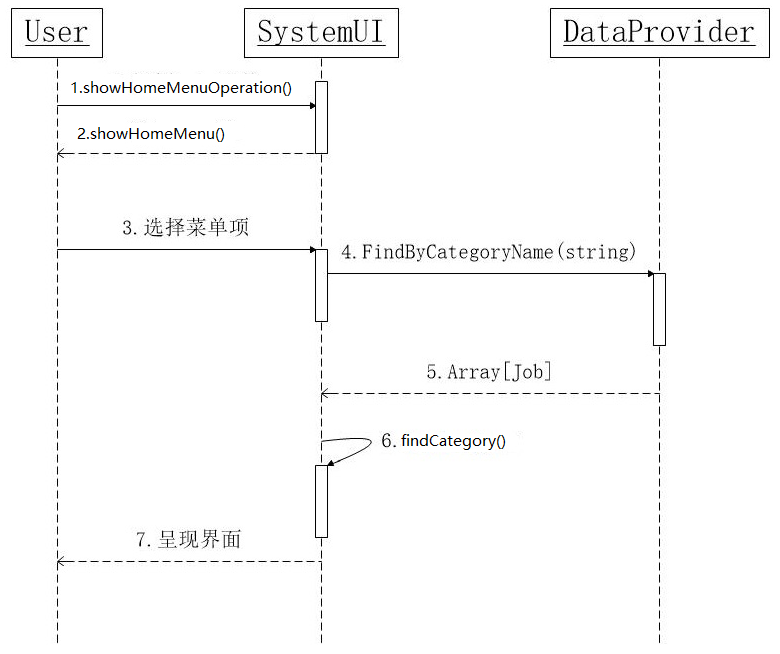
### 用户登录时序图



从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showLoginMeun() | 显示登录菜单 |
| LoginOperation() | 用户登录操作 |
| DataProvider | Login(string,string) | 登录（账号，密码） |

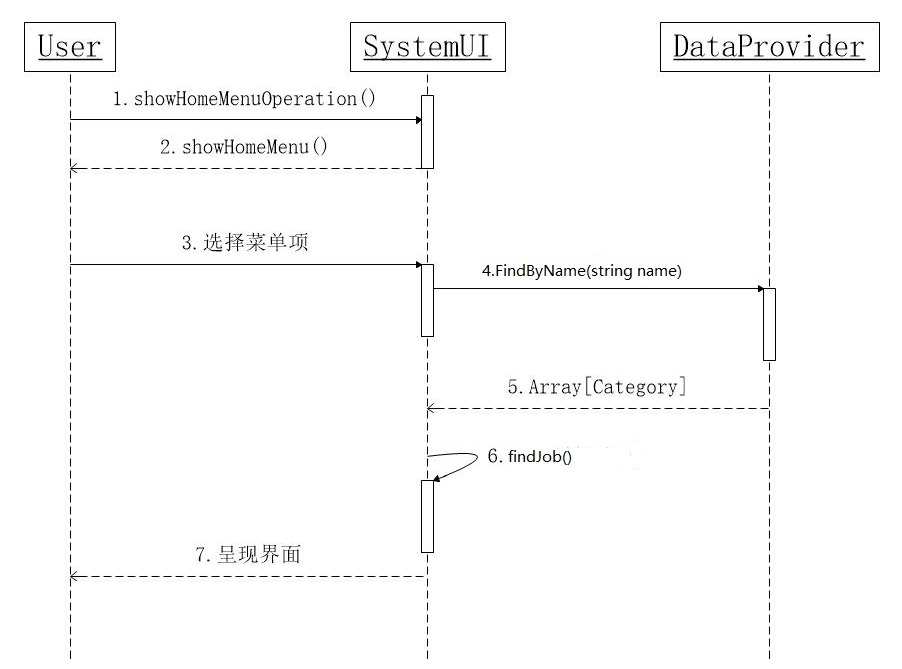
### 用户查看某分类下所有工作时序图



从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showHomeMenu() | 显示主菜单供用户选择 |
| findCategory() | 查找分类及分类下的工作 |
| DataProvider | FindByCategoryName(string) | 根据分类名称获取所有工作信息 |

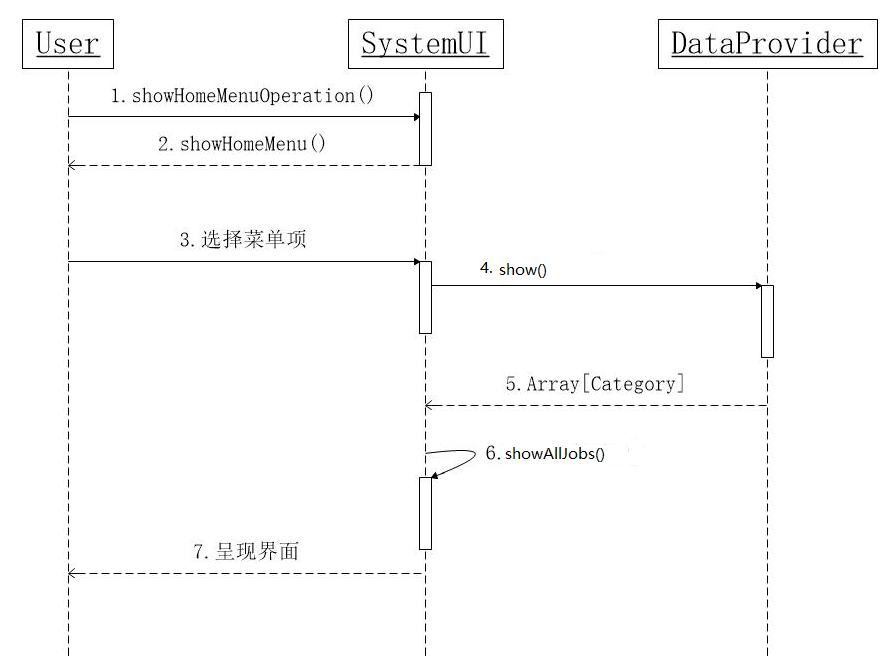
### 根据工作名称搜索工作时序图



从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showHomeMenu | 显示菜单供用户选择 |
| findJob() | 查找工作 |
| DataProvider | FindByName(string) | 根据工作名称获取所有职位信息 |

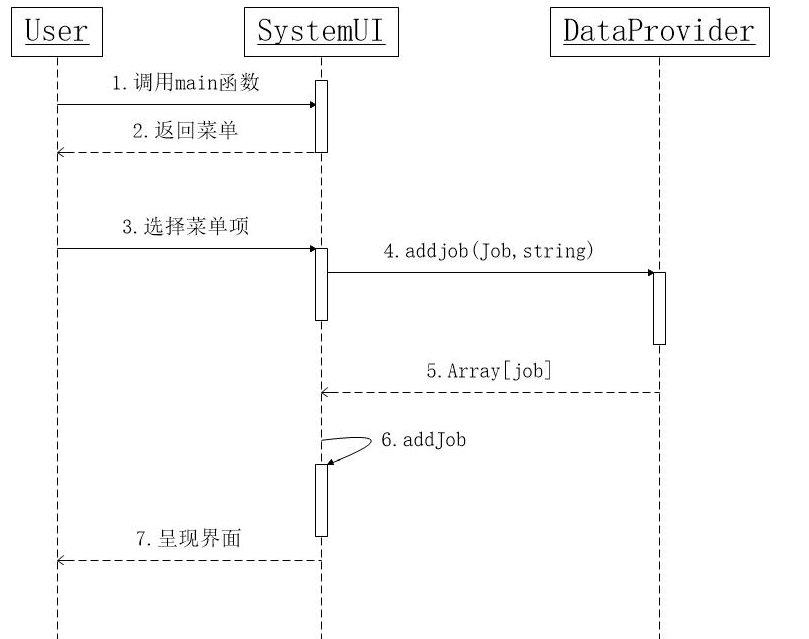
### 显示所有工作时序图



从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showHomeMenu | 显示菜单供用户选择 |
| showAllJobs | 展示所有工作 |
| DataProvider | show | 显示所有分类下职位信息 |

### 用户添加工作时序图

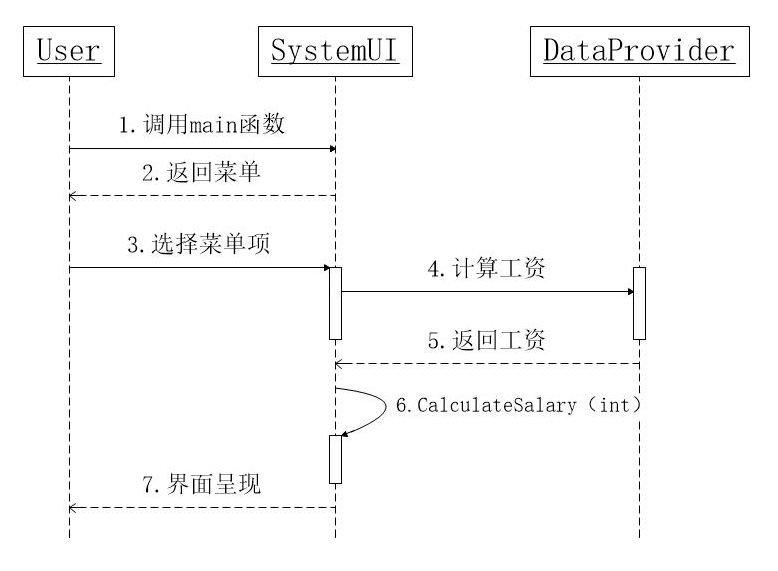


从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showHomeMenu | 显示菜单供用户选择 |
| addJob | 添加工作信息 |
| DataProvider | addJob(Job,string) | 添加工作（工作类，工作类别） |

### 用户计算工资时序图

从图上可以看出，用户查看系统首页时，涉及的对象有SystemUI和DataProvider对象，他们的接口如下表所示。

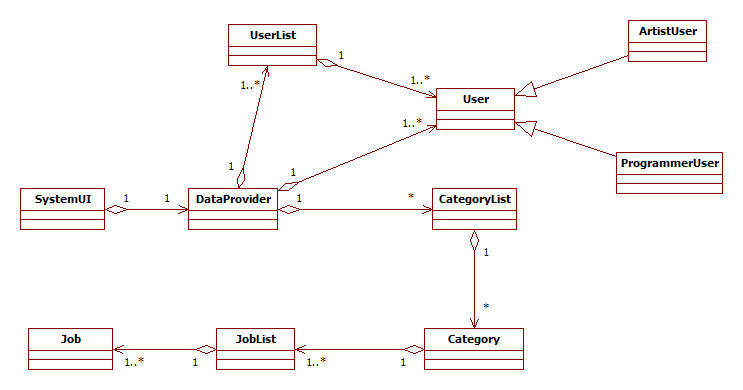


|  |  |  |
| --- | --- | --- |
| 对象 | 接口名 | 接口说明 |
| SystemUI | showHomeMenu | 显示菜单供用户选择 |
| CalulateSalary(int) | 计算工资 |

## 类图及说明

类图实际上实在对系统进行整体设计之后，系统详细设计时所做的工作，通过类图我们把用例和时序进行整理，使系统的实现跃然纸上。

### 整体类图



这时有同学可能会觉得比较茫然，之前角色分析中，我们明明只分析出了3个角色：用户角色、系统界面角色以及数据提供者角色。假设每一个角色是一个类，那应该有3个类，怎么整体类图里却涉及了8个类呢？

实质上这个我们之前设计的并不矛盾，我们只不过是将每一个大的角色再次进行了细分。例如数据提供者DataProvider 角色，我们知道数据提供者的作用是提供数据，那么它应该提供哪些数据？怎么提供呢？这时我们就引入一个思路，即将数据按照一定的分类进行提供，那么数据类别就成为了一个单独的类，即Category类，同时数据提供者能够提供的数据类别应该不止一种，因此我们再设计一个类似于类别集合的角色，用于保存所有的数据类别，因此就需要一个CategoryList类。

以此类推，我们**将每一个基础角色（****User、Category、Job）单独设置成一个类，其中存在两个继承类（ArtistUser、ProgrammerUser）表示不同角色类别继承于角色User类，然后为每一个基础角色添加一个集合，并将每一个集合（UserList、CategoryList、JobList）单独设置成一个类，再加上之前所设计的UI类和数据提供者DataProvider 类，共10个类。**

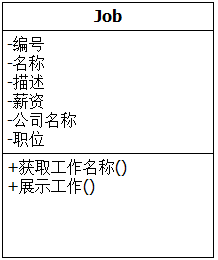
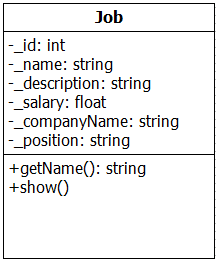
以下是针对每一个类的简单描述。

* **Job** : 工作类，用来表示系统中的工作。
* **JobList**: 工作集合类，是系统中工作的集合类，主要将工作保存在工作列表类的集合中，关联系统中的Job工作类。
* **Category** : 工作类别类，用来表示系统中工作的类别，将系统中的工作进行分类，关联工作列表类。
* **CategoryList** : 工作类别集合类，是系统中工作类别的集合类，用来将类别保存在集合中，关联系统中的Category类别类。
* **User**：HR用户类。
* **ArtistUser**: 美工工作类别类，继承于用户类，用来表示工作中工作类别为美工的工作。
* **ProgrammerUser**: 程序员工作类别类，继承于用户类，用来表示工作中工作类别为程序员的工作。
* **UserLIst**：作为系统中用户类的集合类，关联系统中的User用户类，主要的功能作用就是将用户类保存在集合中。
* **DataProvider** : 系统中提供数据服务的单元，系统中涉及到数据输入、输出的操作完全由DataProvider完成.其中包括三个类分别为：User(用户类)、UserList(用户集合类)和CategoryList(类别集合类).
* **SystemUI** : 界面上的请求处理器对象，用来获取用户的请求的ID，并协同DataProvider完成处理，返回需要的数据给请求者。

上图是项目相关的整体类图，从整体类图来看，系统涉及到的类比较多，如果结合用例图来看，实际上这些类并不复杂，各自之间有着千丝万缕的联系，实际上我们在思考系统类图时可以先把每个类看作是一个与其他类毫无关系的单一个体，提取出系统中可能涉及到的类，最后再结合系统边界捋清各自的职责，构建相互之间的联系，而类之间的联系无非是继承、聚合、组合这几种关系，下面的篇幅我们会对系统中的类图作个体说明。

### Job

Job是系统中用来表示工作的一个类，Job的类图及其成员说明如下图所示。

伪类图转成实际类图

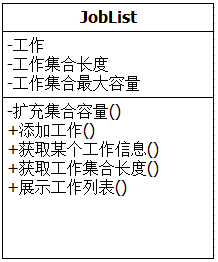
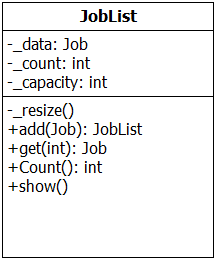
**注：属性或方法前面的加号“+”表示****该属性或方法为共有的（public）权限，而减号“-”则表示该属性或方法为私有的（private）权限，以下类图同理，不再赘述。**

为了方便大家对类中各成员的了解，特别制作了如下的表格来对job作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_id | 工作编号 |  |
| \_name | 工作名称 |  |
| \_salary | 工作薪资 |  |
| \_position | 职位名称 |  |
| \_description | 工作内容描述 |  |
| \_companyName | 工作所属公司名称 |  |
| void show()； |  | 展示当前工作对象信息 |
| string getName()； |  | 获取当前工作对象名称 |

### JobList

JobList主要是系统中工作列表的一个集合类，JobList的类图及其成员说明如下图所示。

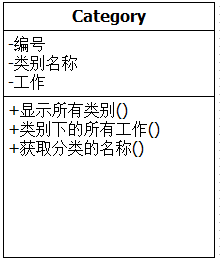
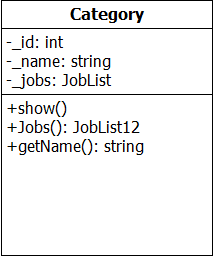
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对jobList作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_data | 工作对象 |  |
| \_count | 工作集合长度 |  |
| \_capacity | 工作集合最大容量 |  |
| void resize()； |  | 扩充工作集合容量 |
| JobList\* add(Job& job)； |  | 给工作集合中添加一项工作 |
| Job\* get(int)； |  | 根据工作编号获取某项工作 |
| int Count()； |  | 获取当前工作集合长度 |
| void show()； |  | 展示当前工作列表下所有工作的信息 |

### Category

Category主要为系统中的工作分类，Category的类图及其成员说明如下图所示。

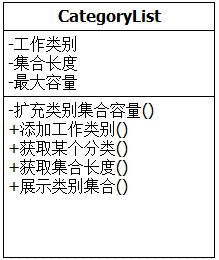
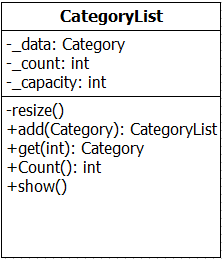
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对Category作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_id | 工作分类的编号 |  |
| \_name | 工作分类的名称 |  |
| \_jobs | 当前分类下的工作信息列表 |  |
| void show()； |  | 展示当前分类下所有工作信息 |
| JobList\* Jobs()； |  | 获取当前分类下的工作列表 |
| string getName()； |  | 获取当前工作分类的名称 |

### CategoryList

CategoryList主要为工作分类的一个集合类，CategoryList的类图及其成员说明如下图所示。

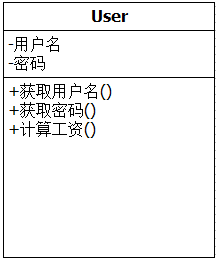
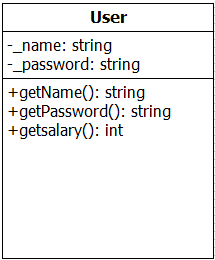
伪类图转成实际类图

为方便大家对类中各成员的了解，特别制作了如下的表格来对CategoryList作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_data | 工作类别 |  |
| \_count | 类别集合长度 |  |
| \_ capacity | 工作类别集合最大容量 |  |
| int Count()； |  | 获取当前工作类别的总数 |
| void show()； |  | 展示所有分类下的工作信息 |
| Category\* get(int)； |  | 根据工作编号获取工作类别 |
| void resize()； |  | 扩充分类集合容量 |
| CategoryList\* add(Category&)； |  | 给工作类别添加一个分类 |

### User

User即为HR用户， User的类图及其成员说明如下图所示。

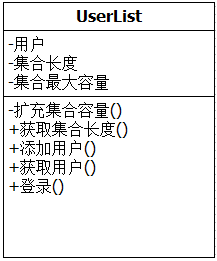
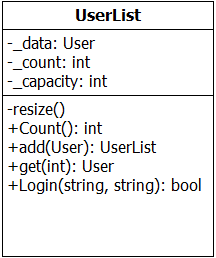
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对User作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_name | 用户账号 |  |
| \_password | 用户密码 |  |
| string getName()； |  | 获取用户账号 |
| string getPassword()； |  | 获取用户密码 |
| Int getsalary(); |  | 计算工资 |

### UserList

UserList的是系统中用户列表的一个集合类, UserList的类图及其成员说明如下图所示。

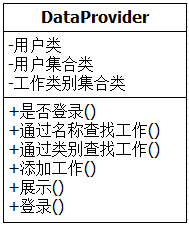
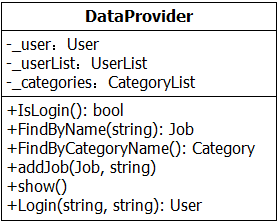
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对UserList作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_data | 用户 |  |
| \_count | 用户集合长度 |  |
| \_capacity | 用户集合最大容量 |  |
| int Count()； |  | 获取当前用户  集合长度 |
| void resize()； |  | 扩展用户集合容量 |
| User\* get(int i)； |  | 根据编号获取  用户信息 |
| bool Login(string \_name，string \_password)； |  | 用户登录操作，通过用户名和密码进行登录 |
| UserList\* add(User& \_user)； |  | 给用户集合添加  一名用户 |

### DataProvider

DataProvider是系统中提供数据服务的单元，该类图及其成员说明如下图所示。

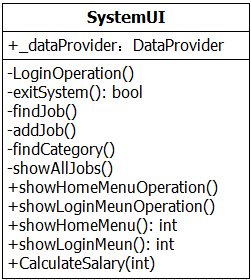
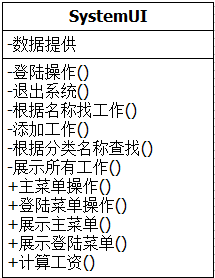
伪类图转成实际类图

为方便大家对类中各成员的了解，特别制作了如下表格来对DataProvider作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_user | 存储登录用户信心 |  |
| \_userList | 存储登录用户信息列表 |  |
| \_categories | 存储所有工作分类列表 |  |
| void show()； |  | 获取所有工作的信息 |
| bool IsLogin()； |  | 检查用户是否登录 |
| void addJob(Job\* job，string CategoryName)； |  | 给某一个工作类别下  添加一项工作信息 |
| User\* Login(string \_name，string \_pwd)； |  | 用户登陆 |
| Job\* FindByName(string name)； |  | 根据工作名称进行搜索 |
| Category\* FindByCategoryName(string name)； |  | 根据分类名称进行搜索 |

### SystemUI

Systemui是一个界面上的请求处理器对象，用来获取用户的请求的ID，并协同DataProvider完成处理，返回需要的数据给请求者。

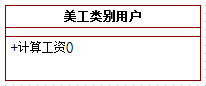
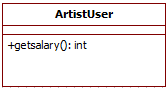
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对SystemUI作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| \_dataProvider | 提供数据DataProvider类 |  |
| void CalculateSalary(int); |  | 计算工资 |
| void LoginOperation()； |  | 用户登录操作 |
| void exitSystem ()； |  | 退出系统 |
| void findJob ()； |  | 根据名称找工作 |
| void addJob ()； |  | 添加工作 |
| void findCategory ()； |  | 根据分类名称查找 |
| void showAllJobs ()； |  | 展示所有工作 |
| void showHomeMenuOperation (); |  | 展示主菜单操作 |
| void showLoginMeunOperation ()； |  | 展示登陆菜单操作 |
| int showHomeMenu(); |  | 展示主菜单 |
| int showLoginMeun(); |  | 展示登陆菜单 |

### ArtistUser

ArtistUser的是系统中继承于用户类的来表示美工工作类别的一个子类, ArtistUser的类图及其成员说明如下图所示。

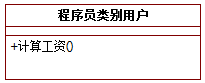
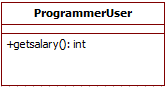
伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对ArtistUser作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| int getsalary() |  | 计算工资 |

### ProgrammerUser

ProgrammerUser的是系统中继承于用户类的来表示程序员工作类别的一个子类, ProgrammerUser的类图及其成员说明如下图所示。

伪类图转成实际类图

为了方便大家对类中各成员的了解，特别制作了如下的表格来对ProgrammerUser作详细说明。

|  |  |  |
| --- | --- | --- |
| **成员** | **属性** | **功能** |
| int getsalary() |  | 计算工资 |