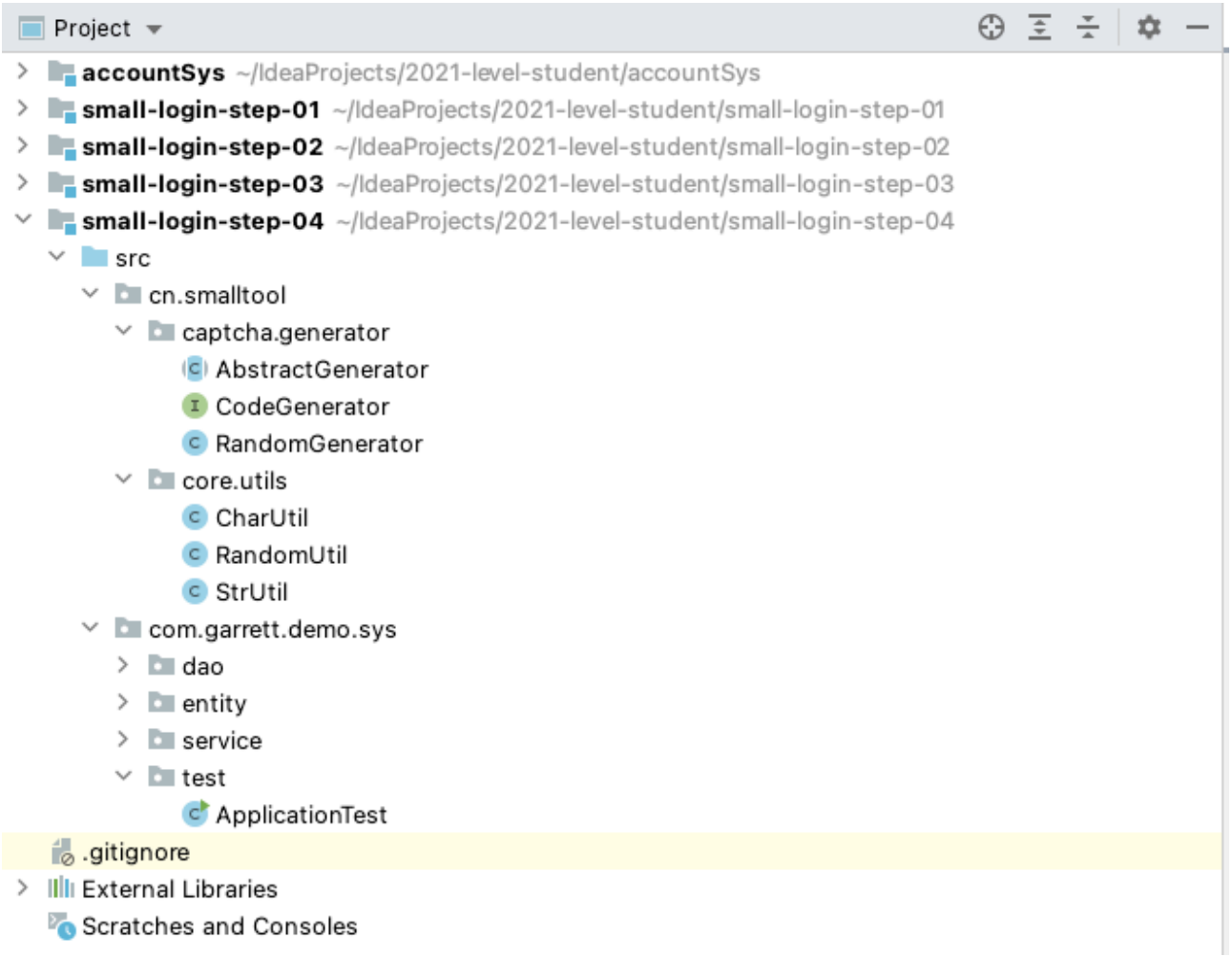
介绍

本次作业以小见大，通过逐个功能的实现，完成一个简化板的登录项目，了解登录模块的基本流程。



---

1. 如果你在学习的过程中遇到什么问题，包括：不能运行、优化意见、文字错误等任何问题都可以询问助教
2. 在每一个章节都提供了清晰的设计图稿和对应的类图，所以学习过程中一定不要只在乎代码是怎么编写的，更重要的是理解这些**设计**的内容是如何来的。

目录

登录模块

第 01 章：开篇介绍

一. 学习目标

本次作业以登录模块学习为目的，通过引导同学们一点点手写简化版的登录框架，锻炼设计思路，为后续再深入学习Java打下基础。

所有的内容实现都会由简开始，一步步带大家实现，最终所有的内容完成后，在提供一个相对完整的单点登录框架，在这个过程中只要你能跟着走下来，那么最后一定可以较容易的实现一些开发实战的功能。

二. 获取源码

本章节是整个学期大作业的开篇，所以这里先把源码地址交代给读者，方便后续大家可以参考学习到这部分内容。

源码实现：https://gitee.com/garrettxia/study_work_21 – 拆解实现步骤，搭建组织工程，展示每一章节的具体源码实现过程。

三. 总结

- 在源码的学习过程中，我会和你一起从最简单、最原始的登录开始，可能有些时候某些章节内容并不会太多，不过我会帮你建立一些知识关联，尽可能让你在这个学习的过程中，收获更多。
- 那么本章节关于**登录模块**的开篇介绍就到这了，接下来你可以阅读到文章，获取到源码，直至我们把所有的内容全部完成，到时候就可以开发出一个相对完整的项目了。希望这个过程中你能和我一直坚持学习打卡！

第 02 章：构建简单的登录

一. 目标

简单的登录定义是什么?

登录是构建所有涉及权限/隐私的重要模块，在项目/应用中对于用户来说是操作的开始，对于大部分应用都有这么一个登录模块。

当一个登录模块被定义实现以后，在逐步的实现人机验证，网络请求中的加密解密，分布式中的单点登录，登录状态的保持等，最终我们可以相对完整的使用一个登录功能。

而本章节的案例目标就是构建一个最简单的登录功能，用于登录和获取用户信息。

二. 设计

接下来我们就按照这个设计，做一个简单的 登录模块的实现。编码的过程往往并不会有多复杂，但知晓设计过程却更加重要!

三. 实现

1. 工程结构

```
small-login-step-01
└─── src
    └─── com
        └─── garrett
            └─── demo
                └─── sys
```

```
        ├── entity
        │   ├── Account.java
        │   └── UserInfo.java
        └── test
            └── ApplicationTest.java
```

整个实现内容非常简单，也仅仅是包括了一个简单的测试主类和POJO，我们在后续的实现过程中再不断的添加内容。

1. Account，备用，代表了账户登录的信息
2. UserInfo，备用，用于存放用户信息，当用户登录成功调用使用

2. 启动类测试

com.garrett.demo.sys.test.ApplicationTest

```
public class ApplicationTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("用户名:");
        String account = scanner.nextLine();
        System.out.print("密码: ");
        String password = scanner.nextLine();

        if ("admin".equals(account) && "admin".equals(password)){
            System.out.println("登录成功!!!");
        }else{
            System.out.println("登录失败!!!");
        }
    }
```

```
}
```

- 目前是最为简单的登录验证方式。
- 不过在后面陆续的实现中会逐步完善登录模块的详细设计，例如：验证码，加密解密，登录持久化，单点登录，第三方登录，授权登录，短信验证登录，重复访问限制等设计。

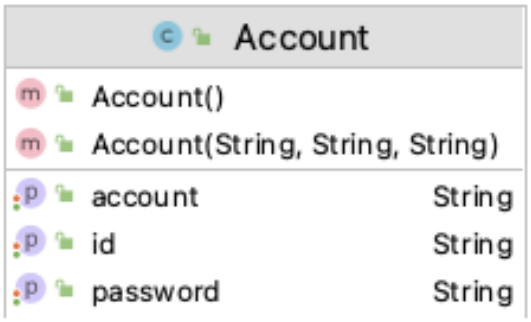
3. 控制台UI

com.garrett.demo.sys.test.ApplicationTest

```
public class ApplicationTest {


     public static void commandTemplate(){
           System.out.println("====================================");
           System.out.println("|                                  |");
           System.out.println("|            极简登录系统          |");
           System.out.println("|                                  |");
           System.out.println("====================================");
     }


    public static void main(String[] args) {
           // 命令窗口启动模版
           commandTemplate();
        Scanner scanner = new Scanner(System.in);
           // ……其余代码
    }
}
```

4. 封装对象（扩展）

学有余力的同学可进一步实现POJO（简单java对象）：



com.garrett.demo.sys.entity.Account

```
/**账号资源*/
public class Account {

    /**编号*/
    private String id;

    /**账户名*/
   private String account;

    /**密码*/
    private String password;

    // ...有参，无参构造方法
    // ...Getter/Setter
}
```

以此类推，UserInfo类，为下一章节做准备

5. 运行结果

```
/Library/Java/JavaVirtualMachines/adopto
====================================
|                                  |
|            极简登录系统            |
|                                  |
====================================
用户名:admin
密码: admin
登录成功!!!

Process finished with exit code 0
```

- 通过运行结果可以看到，目前登录功能，已经稍有雏形。

四. 总结

- 整篇关于最简陋的登录模块的一个雏形已尽实现完成了，相对来说这部分代码并不会难住任何人，只要你稍加尝试就可以接受这部分内容的实现。
- 但对于一个知识的学习来说，写代码只是最后的步骤，往往整个思路、设计、方案，才更重要，只要你知道了因为什么，所以什么，才能让你有一个真正的理解。
- 下一章节会在此工程基础上扩容实现，要比现在的类多一些。不过每一篇的实现上，我都会以一个需求视角进行目标分析和方案设计，让大家在学习编码之外更能注重更多的技术价值的学习。

第 03 章：模拟数据库读取

一. 逻辑设计

你是否能预见复杂内容的设计问题?

讲道理，无论产品功能是否复杂，都有很大一部分程序员会写出一堆 if…else 来完成开发并顺利上线。这主要原因是没法预见当前的需求，发胀是否长远、流量是否庞大、迭代是否迅速，所以被催促上线的情况，不写 if…else 是不可能的!

那你说，既然 if...else 实现的这么快，还考虑数据结构、算法逻辑、设计模式、系统架构吗？当然这基本要看你的项目在可预见下能活多久，如果一个项目至少存活一年，并且在这一年中又会不断的的迭代。就像；你做了一个营销优惠券系统，在各种条件下发放各种类型的券，如果在最开始没有考虑好系统设计和架构模式，那么当活动频发、流量暴增、需求迭代下、最后你可能会挂在系统事故上!

我们在把系统设计的视角聚焦到具体代码实现上，你会有什么手段来实现你想要的设计模式呢？其实编码方式主要依托于：接口定义、类实现接口、抽象类实现接口、继承类、继承抽象类，而这些操作方式可以很好的隔离开每个类的基础功能、通用功能和业务功能，当类的职责清晰后，你的整个设计也会变得容易扩展和迭代。

接下来在本章节继续完善 登录 的功能开发，在这个开发过程中会用到一些接口、类，它们之间会有类的实现、类的继承。可以仔细参考这部分内容的开发实现，虽然并不会很复杂，但这种设计思路是完全可以复用到我们自己的业务系统开发中的。

二. 目标

在上一章节我们初步完成登录的雏形，实现了一个粗糙版本的代码实现。那么本章节我们需要已实现的登录模块进行功能完善，模拟实现数据库的读取校验。
这一次我们把数据的比对交给实例化对象，而不是我们写死在一行判断代码中。此外不仅要实现功能还需要完善基础框架的类结构体，否则将来就很难扩容进去其他的功能了。

三. 设计

鉴于本章节的案例目标，我们需要将登录功能完善起来，首先非常重要的一点是构建一个标准，或者说是接口，用于模拟连接数据源。接下来再需要做的是将数据存取到实例化对象。整体设计如图 3-1

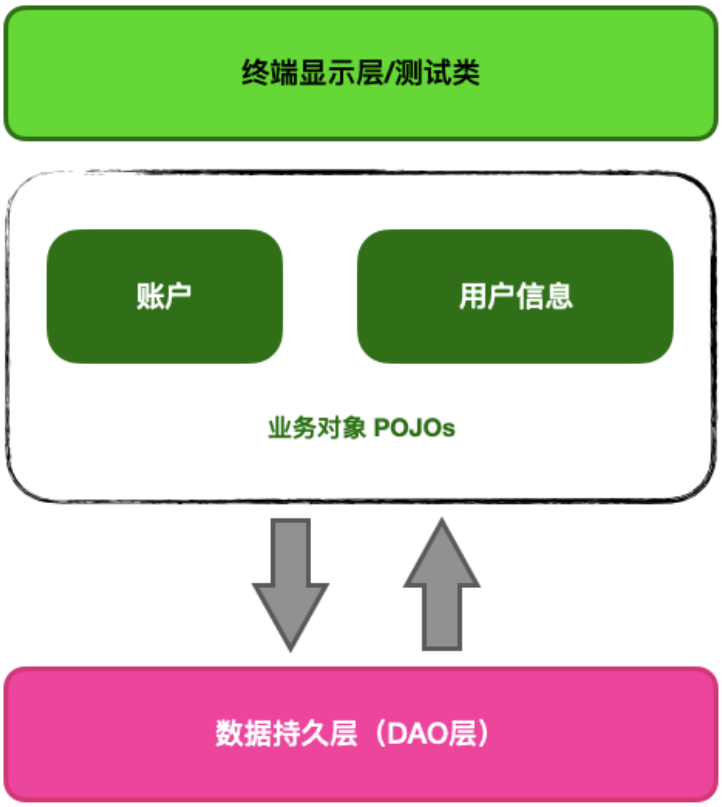


图 3-1

•首先我们需要定义 UserDao 接口，提供用户数据源的获取方法，之后这个UserDao接口由UserDataBaseDemo类实现。这样使用模板模式的设计方式，可以统一收口通用核心方法的调用逻辑和标准定义，也就很好的控制了后续的实现者不用关心调用逻辑，按照统一方式执行。那么类的继承者只需要关心具体方法的逻辑实现可。

•那么在实现UserDao接口后的UserDataBaseDemo就可以实现相应的抽象方法。这里就体现了类实现过程中的各司其职，你只需要关心属于你的内容，不是你的内容，不要参与。这一部分内容我们会在代码里有具体的体现

四. 实现

1. 工程结构

```
small-login-step-02
└── src
    └── com
        └── garrett
            └── demo
                └── sys
                    ├── dao
                    │   ├── UserDao.java
                    │   └── UserDataBaseDemo.java
                    ├── entity
                    │   ├── Account.java
                    │   └── UserInfo.java
                    └── test
                        └── ApplicationTest.java
```

2. UserDao 接口定义

com.garrett.demo.sys.dao.UserDao

```
public interface UserDao {


    /**
     * 根据用户编号获取用户信息
     * @param id 用户编号
     * @return 用户信息
     */
    UserInfo getUserById(String id);


    /**
     * 根据登录账号获取用户信息
     * @param account 账户名
     * @return 用户信息
     */
    Account queryAccount(String account);
}
```

- 其中UserInfo和Account已在上一章节中定义
- 由于接口是后期知识点，同学们也可以直接拉取代码跳过这个步骤，接着构建UserDataBaseDemo实现接口

3. UserDataBaseDemo 接口实现

本接口的实现，同学们可通过implements关键字实现相关方法，具体代码同学们可自行实现，只要符合接口定义的标准即可。也可以参考以下实现类：

com.garrett.demo.sys.dao.UserDataBaseDemo

```java
import com.garrett.demo.sys.entity.Account;
import com.garrett.demo.sys.entity.UserInfo;

public class UserDataBaseDemo implements UserDao{

    private Account onlyAccount = new Account( id: "1", account: "admin", password: "admin");
    private UserInfo onlyUser = new UserInfo( id: "1", stuId: "学号 (212006677) ", realName: "你的姓名");

    /**
     * 根据用户编号获取用户信息
     *
     * @param id 用户编号
     * @return 用户信息
     */
    @Override
    public UserInfo getUserById(String id) {
        if(onlyUser.getId().equals(id)){
            return onlyUser;
        }
        return null;
    }

    @Override
    public Account queryAccount(String account) {
        if (onlyAccount.getAccount().equals(account)) {
            return onlyAccount;
        }
        return null;
    }
}
```

五. 测试

1. 测试用例

com.garrett.demo.sys.test.ApplicationTest

```java
public static void main(String[] args) {

    commandTemplate();

    Scanner scanner = new Scanner(System.in);

    System.out.print("用户名:");
    String account = scanner.nextLine();
    System.out.print("密码: ");
    String password = scanner.nextLine();

    loadLogin(account,password);
}
```

```
static void loadLogin(String account,String password){
    // 标准接口(推荐)
    UserDao userDao =

            // 这里也可以是你自己的实现类
            new UserDataBaseDemo();

    // 如果对接口理解不够清晰的话，暂且可以这么写
    // UserDataBaseDemo userDataBaseDemo = new UserDataBaseDemo();

    Account ac = userDao.queryAccount(account);
    if (ac != null && ac.getPassword().equals(password)) {
        UserInfo userInfo = userDao.getUserById(ac.getId());
        loginResMessage(userInfo);
    }
}
static void loginResMessage(UserInfo userInfo){
    if (userInfo != null) {
        System.out.println("登录成功!!!");
        System.out.println("欢迎登录系统，" + userInfo.getRealName());
    }else{
        System.out.println("登录失败!!!");
        System.out.println("提示：账号或密码错误.");
    }
}
```

- 这里我们把一些一些相关联的代码块提取出来，分装为比较独立的行为方法，这有点类似于C语言的面向过程。
- 我们会发现测试类中有多处的null（空值）判断，这会导致代码冗余以及降低了代码可读性，在实际业务开发中是要尽量避免的。由于不在当前考察的知识范围，我们暂且先这么做一下。感兴趣的同学可以自行提前了解下Java的异常处理

2. 测试结果

```
/Library/Java/JavaVirtualMachines/adopt
==================================
|                                |
|          极简登录系统           |
|                                |
==================================
用户名:admin
密码: admin
登录成功!!!
欢迎登录系统，你的姓名

Process finished with exit code 0
```

六. 总结

- 相对于前一章对登录的简单功能实现，本章节中增加了对数据层的模拟。
- 几乎所有的程序功能设计都离不开接口抽象类、实现、继承，而这些不同特性类的使用就可以非常好的隔离开类的功能职责和作用范围。而这些知识点也是在学习项目搭建过程中非常重要的知识。

①这里我们把interface比作扩展接口，比如USB扩展器之类的，而我们的实现类相当于每个数据接口，至于数据线的另一端接的是鼠标还是打印机等，接口并不关心，只要符合它提供的标准方法即可。

②细心的朋友可能发现，相同的插口也能接不同的显示器，说明接口之下还能再实现接口。

- 最后要强调一下关于整个系列内容的学习，可能在学习的过程中会遇到像第二章节那样非常简单的代码实现，但要做一个有成长的程序员要记住代码实现只是最后的落地结果，而那些设计上的思考才是最有价值的地方。就像你是否遇到过，有人让你给一个内容做个描述、文档、说明，你总觉得太简单了没什么可写的，即使要动笔写了也不知道要从哪开始！其实这些知识内容都来源

你对整体功能的理解，这就不只是代码开发还包括了需求目标、方案设计、技术实现、逻辑验证等等过程性的内容。所以，不要只是被看似简单的内容忽略了整体全局观，要学会放开视野，开放学习视角。

第 04 章：独立出业务逻辑

一. 查缺补漏

技术成长，是对场景设计细节不断的雕刻！

你可能会想什么才是编程能力提升？其实更多的编程能力的提升是你对复杂场景的架构把控以及对每一个技术实现细节点的不断用具有规模体量的流量冲击验证时，是否能保证系统稳定运行从而决定你见识了多少、学到了多少、提升了多少！

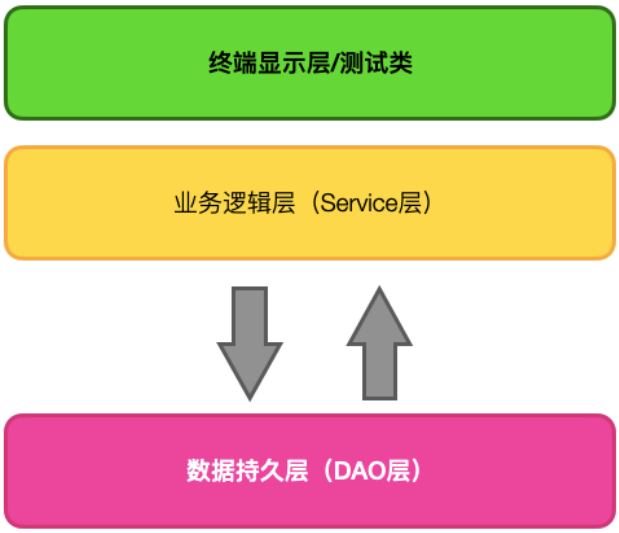
最终当你在接一个产品需求时，开始思考程序数据结构的设计、核心功能的算法逻辑实现、整体服务的设计模式使用、系统架构的搭建方式、应用集群的部署结构，那么也就是的编程能力真正提升的时候！

二. 目标

这一章节的目标主要是独立出相对代码块，封装相对具体的业务逻辑服务层。同时巩固接口的用法。

三. 设计

这里我们不再显式表明POJO，目前简单的将应用分层为服务层，数据层



我们将登录的业务逻辑处理，抽离提取出来，放到Service层中，尽可能的使每个类和方法相对独立，专注于处理一件事。我们称之为单一职责。

四. 实现

1. 工程结构

```
small-login-step-03
└── src
    └── com
        └── garrett
            └── demo
                └── sys
                    ├── dao
                    │   ├── UserDao.java
                    │   └── UserDataBaseDemo.java
                    ├── entity
                    │   ├── Account.java
                    │   └── UserInfo.java
                    ├── service
                    │   ├── AccountService.java
                    │   └── AccountServiceImp.java
                    └── test
                        └── ApplicationTest.java
```

整体关系图，如图 4-2

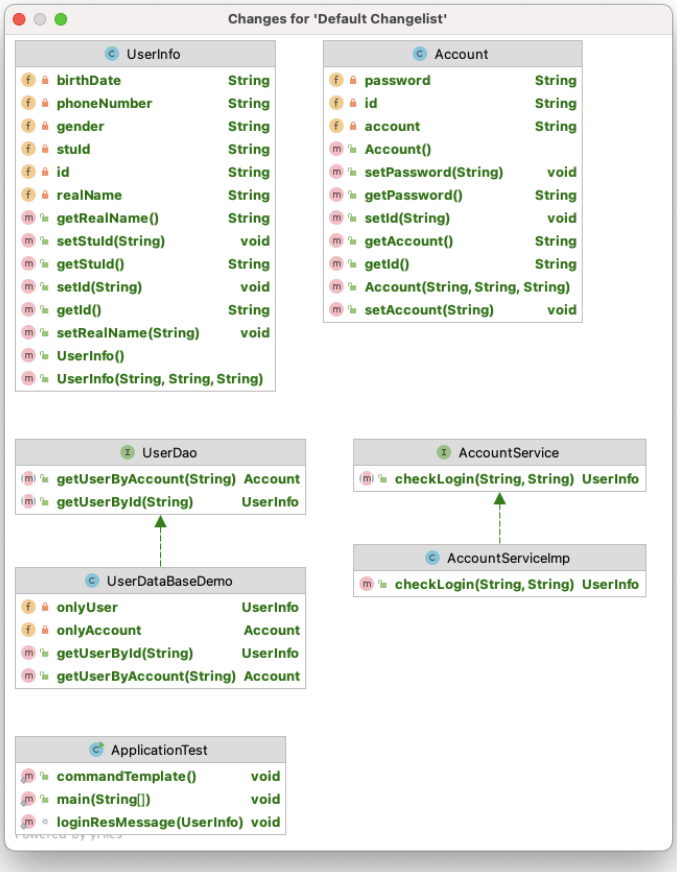


图 4-2

2. 新增 AccountService 接口

com.garrett.demo.sys.service.AccountService

```java
public interface AccountService {

    /**
     * 验证信息
     * @param account 账户名
     * @param password 密码
     * @return 用户信息
     */
    UserInfo checkLogin(String account, String password);
}
```

3. 构建 接口 的实现类

本次我们统一使用该实现类，同学们尽量不要额外构建取它名

com.garrett.demo.sys.service.AccountServiceImp

```
public class AccountServiceImp implements AccountService{

    @Override

    public UserInfo checkLogin(String account, String password) {

        UserDao userDao = new UserDataBaseDemo();

        Account ac = userDao.queryAccount(account);

        // 这里注意我们要用到空值判断，后期我们会优化掉这一部分

        if (null != ac && ac.getPassword().equals(password)) {

            return userDao.getUserById(ac.getId());

        }

        return null;

    }

}
```

五. 测试

1. 测试用例

这一步我们只需将原有的loadLogin(account,password);做一个调整即可

```
public static void main(String[] args) {

    // ...原有代码片段

    String password = scanner.nextLine();


    AccountService accountService = new AccountServiceImp();

    UserInfo userInfo = accountService.checkLogin(account, password);

    loginResMessage(userInfo);

}
```

2. 测试结果

测试结果同上一章

六. 总结

- 本章节的主要以完善层次结构操作，增加接口，巩固上一章节基础。
- 在我们自己业务需求实现的过程中，也要尽可能的去考虑一个良好的扩展性以及拆分好类的职责。
- 动手是学习起来最快的方式，不要让眼睛是感觉看会了，但上手操作就废了。也希望有需要的读者可以亲手操作一下，把你的想法也融入到可落地实现的代码里，看看想的和做的是否一致。

第 05 章：人机识别·验证码

一. 功能完善

超卖、掉单、秒杀，你的程序总是不抗揍！

想想，运营已经对外宣传了七八天的活动，满心欢喜的等着最后一天页面上线对外了，突然出现了一堆异常、资损、闪退，而用户流量稍纵即逝，最后想死的心都有！

就程序 Bug 来讲，会包括产品流程上的 Bug、运营配置活动时候的 Bug、研发开发时功能实现的 Bug、测试验证时漏掉流程的 Bug、上线过程中运维服务相关配置的 Bug，而这些其实都可以通过制定的流程规范和一定的研发经验积累，慢慢尽可能减少。

而另外一类是沟通留下的 Bug，通常情况下业务提需求、产品定方案、研发做实现，最终还要有 UI、测试、运营、架构等等各个环节的人员参与到一个项目的承接、开发到上线运行，而在这一群人需要保持一个统一的信息传播其实是很难的。比如在项目开发中期，运营给产品说了一个新增的需求，产品觉得功能也不大，随即找到对应的前端研发加个逻辑，但没想到可能也影响到了后端的开发和测试的用例。最后功能虽然是上线了，可并不在整个产研测的需求覆盖度范围里，也就隐形的埋下了一个坑。

所以，如果你想让你的程序很抗揍，接的住农夫三拳，那么你要做的就不只是一个单纯的搬砖码农！

二. 目标

这几章我们初步完成一个登录需求，以及代码结构上的优化。那么和常见的线上环境中我们还缺少什么呢？其实还缺少人机验证，也就是验证码这一块。

本章节我们就针对验证码功能做一个模拟操作。这里我们暂时不考虑图片式的验证码，否则会把整个功能实现撑大，这样初学时就把握不住了，待后续先把UI界面引入后，再逐步完善

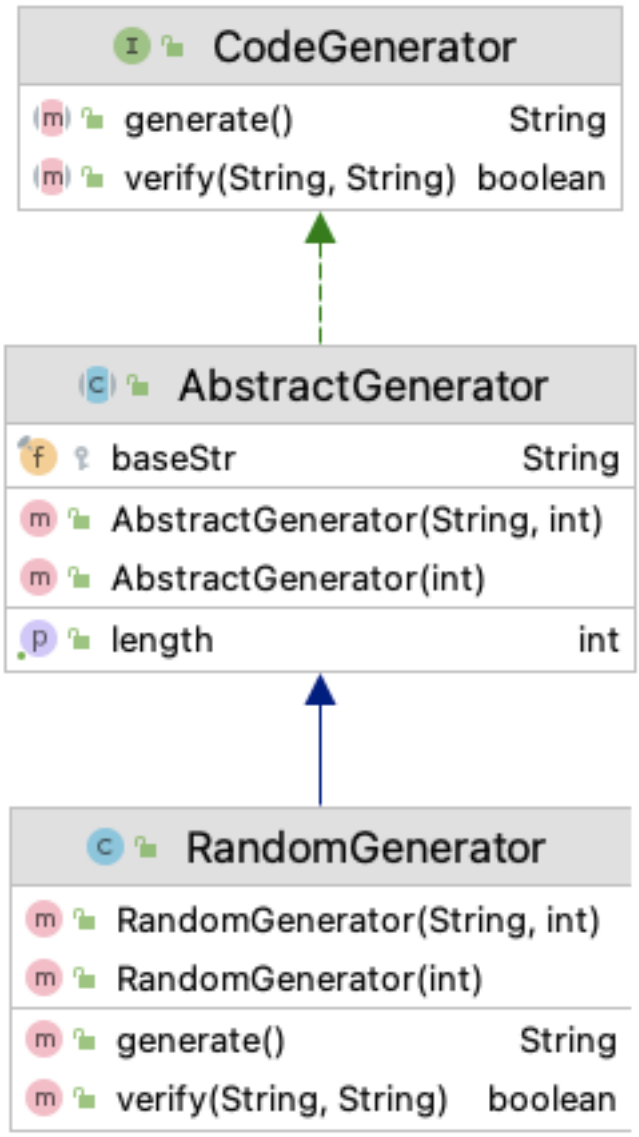
三. 设计

首先我们需要提供基本的验证码生成和校验。其次，最好的话，我们定制生成验证的参数和长度，或者给予默认值。
由于其具备一定的通用性，我们将其放到一个工具类包中，以便可以长期复用

四. 实现

1. 工程结构

```
small-login-step-04
└── src
    ├── cn
    │   └── smalltool
    │       ├── captcha
    │       │   └── generator
    │       │       ├── AbstractGenerator.java
    │       │       ├── CodeGenerator.java
    │       │       └── RandomGenerator.java
    │       └── core
    │           └── utils
    │               ├── CharUtil.java
    │               ├── RandomUtil.java
    │               └── StrUtil.java
    └── com
        └── garrett
            └── demo
                └── sys
                    ├── dao
                    │   ├── UserDao.java
                    │   └── UserDataBaseDemo.java
                    ├── entity
                    │   ├── Account.java
                    │   └── UserInfo.java
                    ├── service
                    │   ├── AccountService.java
                    │   └── AccountServiceImp.java
                    └── test
                        └── ApplicationTest.java
```



2. 定义接口

cn.smalltool.captcha.generator.CodeGenerator

```
public interface CodeGenerator {

    /**
     * 生成器
     * @return 验证码
     */
    String generate();

    /**校验*/
```

```
    boolean verify(String var1, String var2);
}
```

3. 定义抽象类生成器

cn.smalltool.captcha.generator.AbstractGenerator

```
public abstract class AbstractGenerator implements CodeGenerator {
    protected final String baseStr;  // 基础字符串
    protected final int length; // 验证码长度
    public AbstractGenerator(int count) {
        this("abcdefghijklmnopqrstuvwxyz0123456789", count);
    }
    public AbstractGenerator(String baseStr, int length) {
        this.baseStr = baseStr;
        this.length = length;
    }
    public int getLength() {return this.length;}
}
```

- 在生成器构建时，至少需要提供验证码的长度。默认提供A-Z以及数字的字符串值。
- 也可以由用户自定义验证码内容，比如中文验证，符号验证等。

4. 一些常用工具的前提

我们这里先编写一些常用工具类，为生成随机码做准备。
首先我们需要在给定范围内生成随机字符串
而给定范围的字符串我们常常要做，边界测试，所以我们可以封装一个字符串判断的工具

```
public class StrUtil {
  // ...其他代码片段...
    public static boolean isEmpty(String str) {
        return str == null || str.length() == 0;
    }
}
```

亦可以延伸出字符的空值（无意义值）判断：

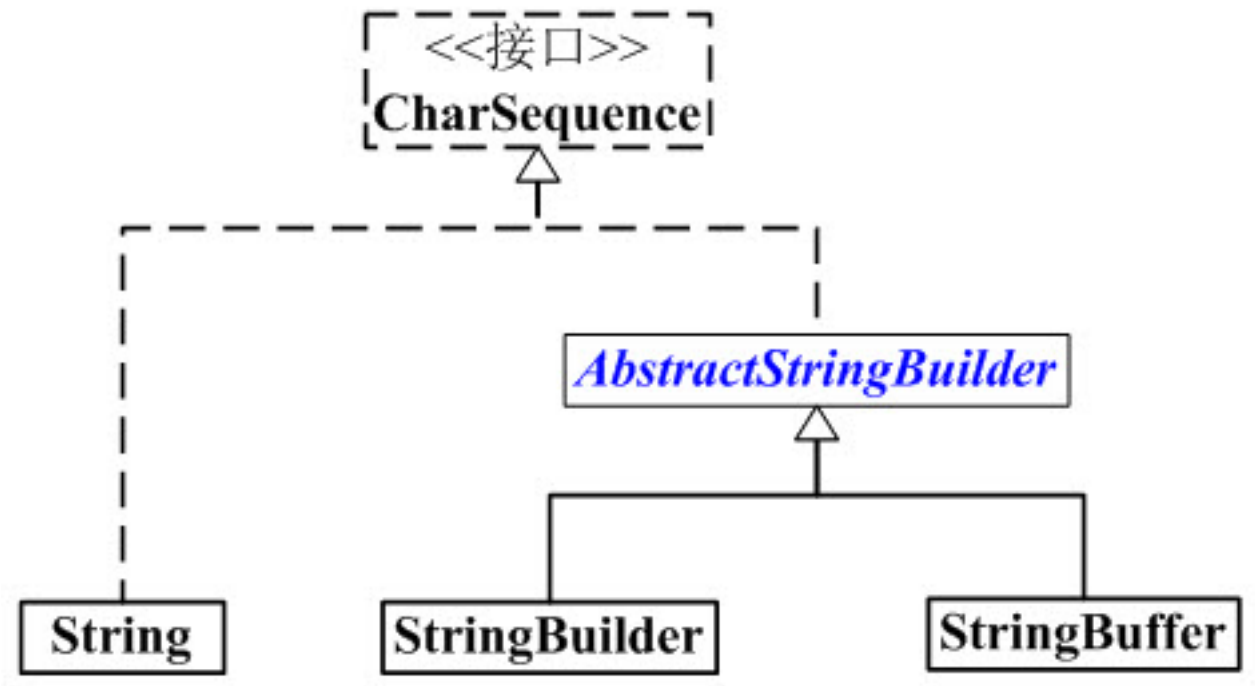
cn.smalltool.core.utils.CharUtil

```
public class CharUtil {
    public static boolean isBlankChar(char c) {
        return isBlankChar((int)c);
    }
    public static boolean isBlankChar(int c){
        // 空字符判断以及不同编码格式下的无意义字符判断
        return Character.isWhitespace(c) || Character.isSpaceChar(c) || c
== 65279 || c == 8234;
    }
}
```

那么StrUtil工具类中就可以具备空/无意义的字符串判断：

```
public static boolean isNotBlank(CharSequence str) {
    return !isBlank(str);
}
public static boolean isBlank(CharSequence str) {
    int length;
    if (str != null && (length = str.length()) != 0) {
        for(int i = 0; i < length; ++i) { // 遍历
            if (!CharUtil.isBlankChar(str.charAt(i))) {
                return false;
            }
        }
        return true;
    } else {
        return true;
    }
}
```

- 这里要补充说明的是，方法的参数使用的CharSequence类而不是String类，首先见关系图如下：



这一块也是本次大作业知识的重点，为什么这些方法不干脆定义String作为参数类型?

因为还有其他的CharSequence类型的类，比如StringBuffer和StringBuilder这两个很重要的类。String对象是不可变的，这两个可变，所以我们在构造字符串的过程中往往要用到StringBuffer和StringBuilder。

如果那些方法定义String作为参数类型，那么就没法对它们用那些方法，先得转化成String才能用。

但StringBuffer和StringBuilder转换为String再转换过来很花时间的，用它们而不是直接用String的“加法”来构造新String本来就是为了省时间，所以如果用String作为参数类型就悲剧了。

我们这样做提高了方法的通用性，使得3种类型都能快速使用该方法。后面的一些应用也是如此。

其次是一些字符串的比对，如忽略大小写，或者大小写全匹配：

```
public static boolean equalsIgnoreCase(CharSequence str1, CharSequence
str2) {
    return equals(str1, str2, true);
}
```

```
public static boolean equals(CharSequence str1, CharSequence str2,
boolean ignoreCase) {
    if (null == str1) {
        // 如果验证码为空指针，那么判断用户输入是否也是空指针，保证业务的合法性
        return str2 == null;
    } else if (null == str2) { // 如果验证码不为空，但用户输入是null，不合法!
        return false;
    } else { // 进行内容匹配
        return ignoreCase ?
str1.toString().equalsIgnoreCase(str2.toString()) :
str1.toString().contentEquals(str2);
    }
}
```

这样，我们的随机验证码就能够使用了，并且具备一定的复用性.

cn.smalltool.captcha.generator.RandomGenerator

```java
import cn.smalltool.core.utils.RandomUtil;
import cn.smalltool.core.utils.StrUtil;

/**
 * 随机验证码
 */
public class RandomGenerator extends AbstractGenerator{

    public RandomGenerator(int count) {
        super(count);
    }

    public RandomGenerator(String baseStr, int length) { super(baseStr, length); }

    /**
     * 生成器
     *
     * @return 验证码
     */
    @Override
    public String generate() {
        return RandomUtil.randomString(this.baseStr, this.length);
    }

    /**
     * 校验
     *
     * @param code 生成码
     * @param userInputCode 用户输入的验证码
     */
    @Override
    public boolean verify(String code, String userInputCode) {
        return StrUtil.isNotBlank(userInputCode) && StrUtil.equalsIgnoreCase(code, userInputCode);
    }
}
```

五. 测试

1. 事先准备

```
static void codeCheck(){
    Scanner sc = new Scanner(System.in);
    while (true) {
        //提前吞掉空格，防止后面吞掉
        //  sc.nextLine();

        // 长度为4的随机码生成器
        RandomGenerator randomGenerator = new RandomGenerator(4);
        String randomStr = randomGenerator.generate();
        System.out.println("\n请输入验证码，以区分您不是机器人");
        System.out.println("==========================");
        System.out.println("|  验证码  |    " + randomStr + "     |");
        System.out.println("==========================");

        // 获取输入的验证码
        String code = sc.next().trim();
        if (randomGenerator.verify(randomStr,code)) {
            return;
        }else{
            System.out.println("验证码错误！");
            System.out.println("正在重置验证码...");
        }
    }
}
```

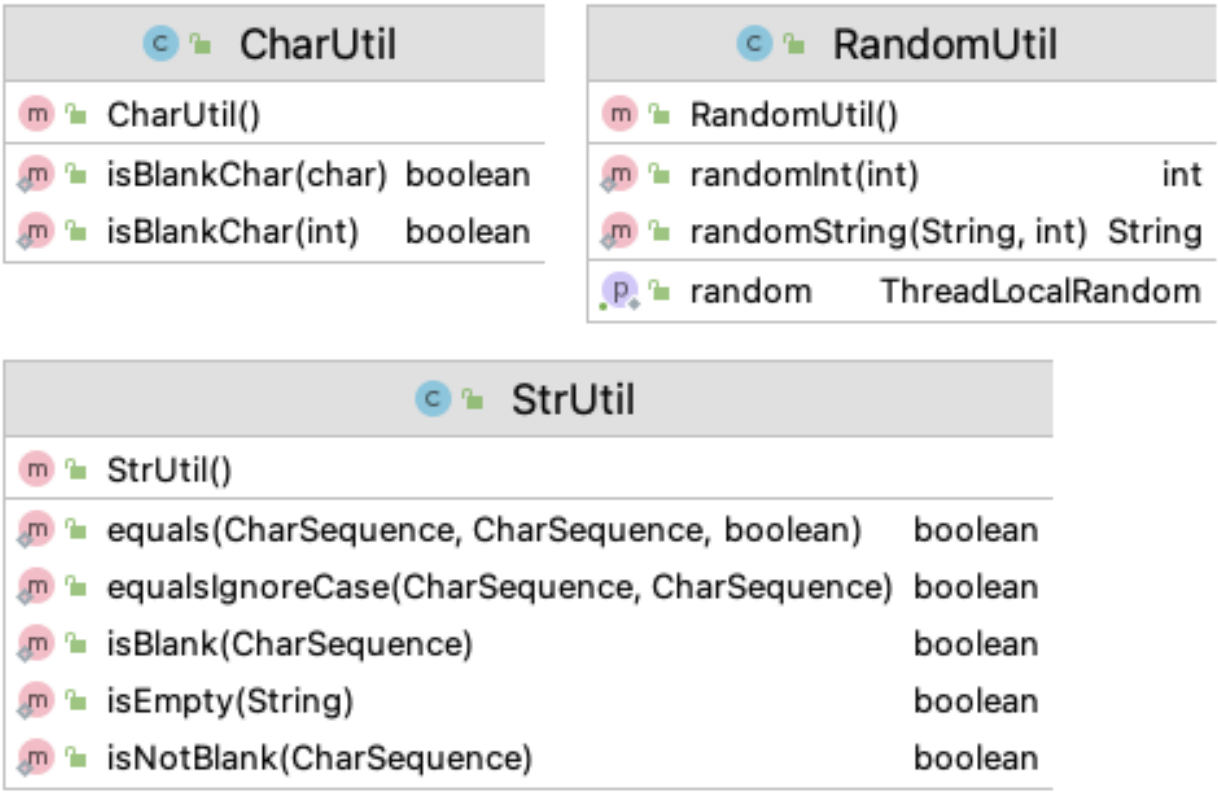
2. 测试用例

```
public static void main(String[] args) {
    // 命令窗口启动模版
    commandTemplate();
    Scanner scanner = new Scanner(System.in);
    System.out.print("用户名:");
    String account = scanner.nextLine();
    System.out.print("密码: ");
    String password = scanner.nextLine();

    // 验证码功能
    codeCheck();

    AccountService accountService = new AccountServiceImp();
    UserInfo userInfo = accountService.checkLogin(account,
password);

    // 登录结果消息提示
    loginResMessage(userInfo);
}
```

3. 测试结果

```
/Library/Java/JavaVirtualMachines/ado
====================================
|                                  |
|             极简登录系统           |
|                                  |
====================================
用户名:admin
密码: admin

请输入验证码，以区分您不是机器人
=========================
|   验证码   |    o339     |
=========================
osdf
验证码错误!
正在重置验证码...

请输入验证码，以区分您不是机器人
=========================
|   验证码   |    cv8g     |
=========================
cv8g
登录成功!!!
欢迎登录系统，你的姓名

Process finished with exit code 0
```

六. 总结



- 在本章节中我们把一些常用的工具类封装出来，提高内聚性，降低耦合度。
- 同时也了解String、StringBuilder、StringBuffer三者关系，字符串的大小写比较和随机验证码的生成，其中涉及字符串内容的查找和随机数机制
- 每一个章节的功能点我们都在循序渐进的实现，这样可以让新人更好的接受关于整个项目的设计思路。尤其是在一些已经开发好的类上，怎么扩充新的功能时候的设计更为重要。学习编程有的时候学习思路设计要比仅仅是做简单实现，更能提升编程思维。
- 到这一章节关于 验证码 的生成操作就开发完成了，接下来需要整个框架的基础上完成友好的用户体验，接下来会直接让读者对接前端页面，只需运行起来即可，让我们这小项目越来越像完善。

第 06 章：用户界面与测试

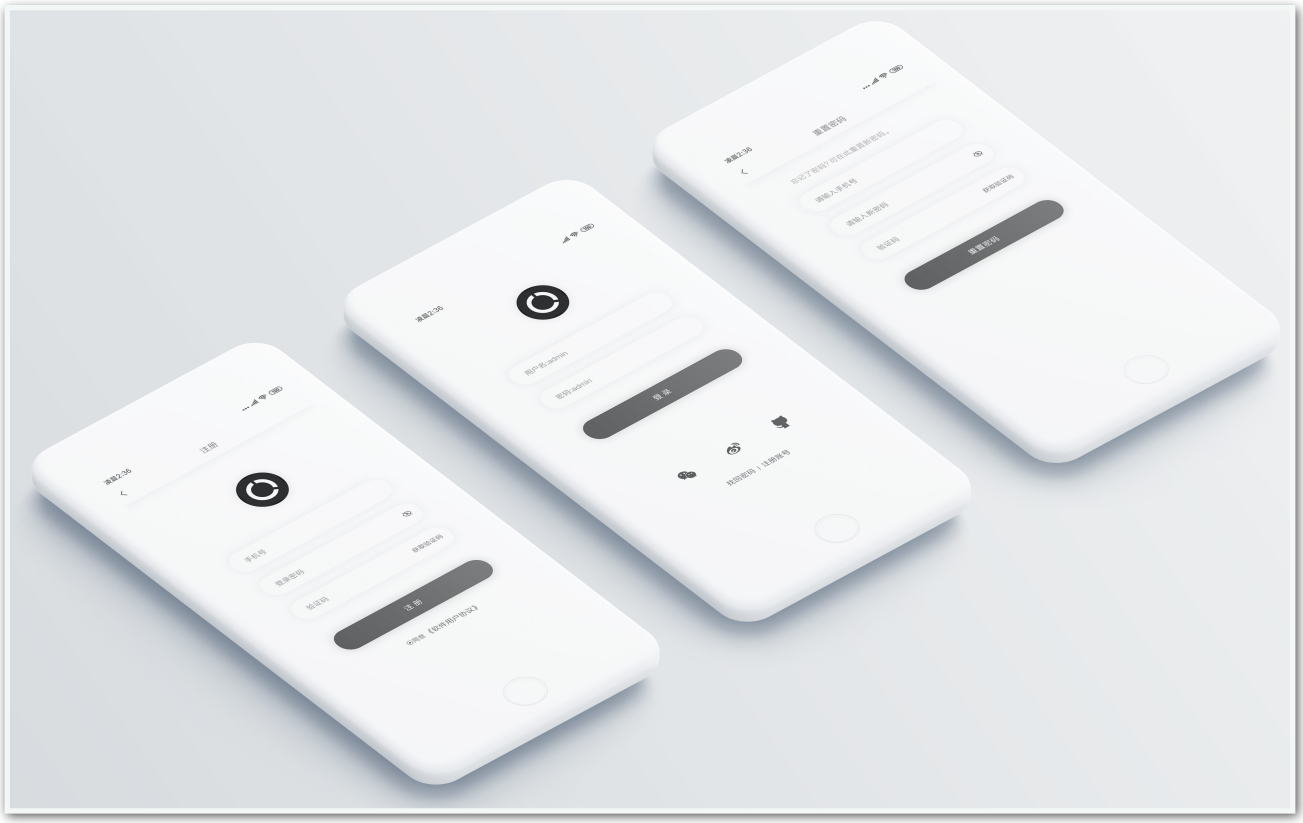
一. 美中不足

佛靠金装，人靠衣装

我们整体完成一个阶段的开发功能，现在我们可以用命令行去使用它。但这毕竟缺少点遗憾，实际用户使用过程中，用户是不懂如何使用命令行的，就像我们刚学习编程调试环境的时候。这是非常差的用户体验的。

二. 目标

本章节非必要掌握。



由于UI这一部分并不再本课程学习范围之内，本章节主要是对接我给大家调试好的前端程序，使得同学们更熟悉开发流程。

同时一部分代码是由笔者提供，加上同学们自己手写的代码，这之间的对接，在于代码是否健壮，高内聚和松散耦合。这一部分我建议同学们先按照要求进行学习，再完成后做自己的定制化调整，UI的调试总能让人爱不释手。

PS. 本章节**强烈建议**使用IDEA （旗舰版/专业版）学习，因为这样不需要关心额外的配置和知识点。如果你使用的并非该版本，你需要简单了解下Maven，以及开发工具里集成Maven的配置。

三. 实操

开发环境

- Java环境jdk1.8
- IDEA 下载地址：**JetBrins 官网**（Ultimate/专业/旗舰版）
- HBuilder 下载地址：**HBuilder官网**
- 前端代码 下载地址：**极简登录模版** OR git：https://codechina.csdn.net/m0_62802534/login_demo.git

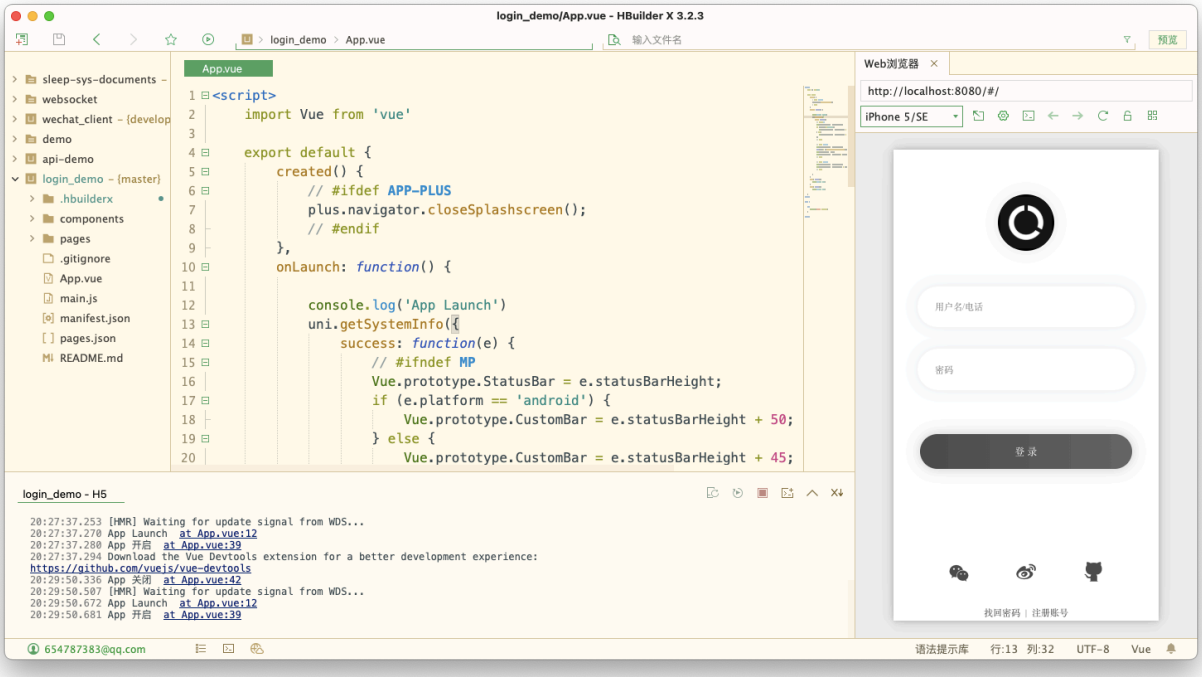
导入前端代码

- 点击HBuilder菜单选项 文件 ——> 导入 ——> 本地目录 导入。

- 选中下载好的前端的 login_demo 目录，并导入
- 如果有会git的的同学，可直接git导入

运行前端项目

- 打开该项目下的任意文件，点击菜单选项 运行 ——》运行到内置浏览器（如果遇到需要安装内置浏览器插件，直接点击安装，片刻后待安装完成重复此步骤）。



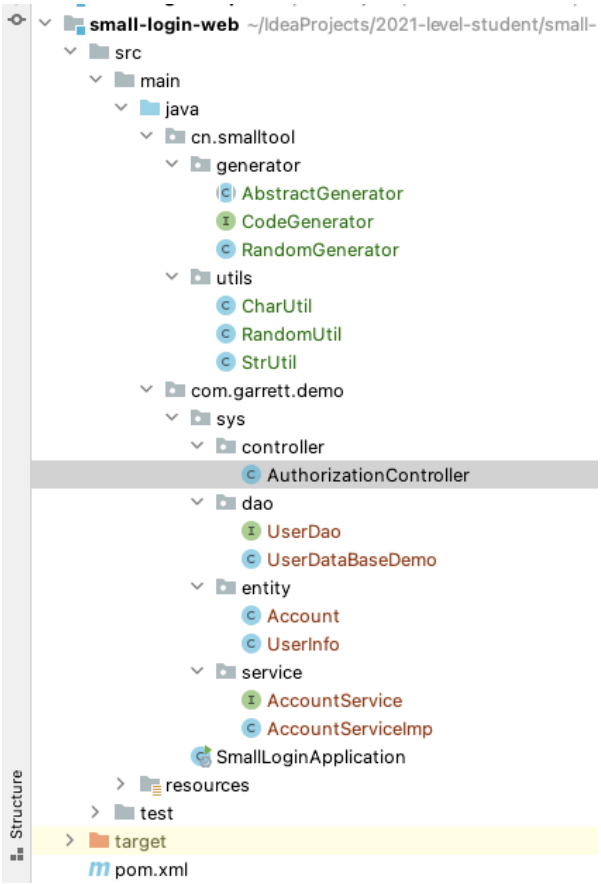
- 位于右部的web浏览器，类型选择iPhone 5/SE；如果出现样式于上图不符，可点击web浏览器中的刷新按钮

准备后端代码部署

- 拉取最新的java代码，确保项目中存在 small-login-web 模块
- 将上一章节的代码内容复制/拷贝到模块中，模块结构如下：

```
small-login-web
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── cn
│   │   │   │   └── smalltool
│   │   │   │       ├── generator
│   │   │   │       │   ├── AbstractGenerator.java
│   │   │   │       │   ├── CodeGenerator.java
│   │   │   │       │   └── RandomGenerator.java
│   │   │   │       └── utils
│   │   │   │           ├── CharUtil.java
```

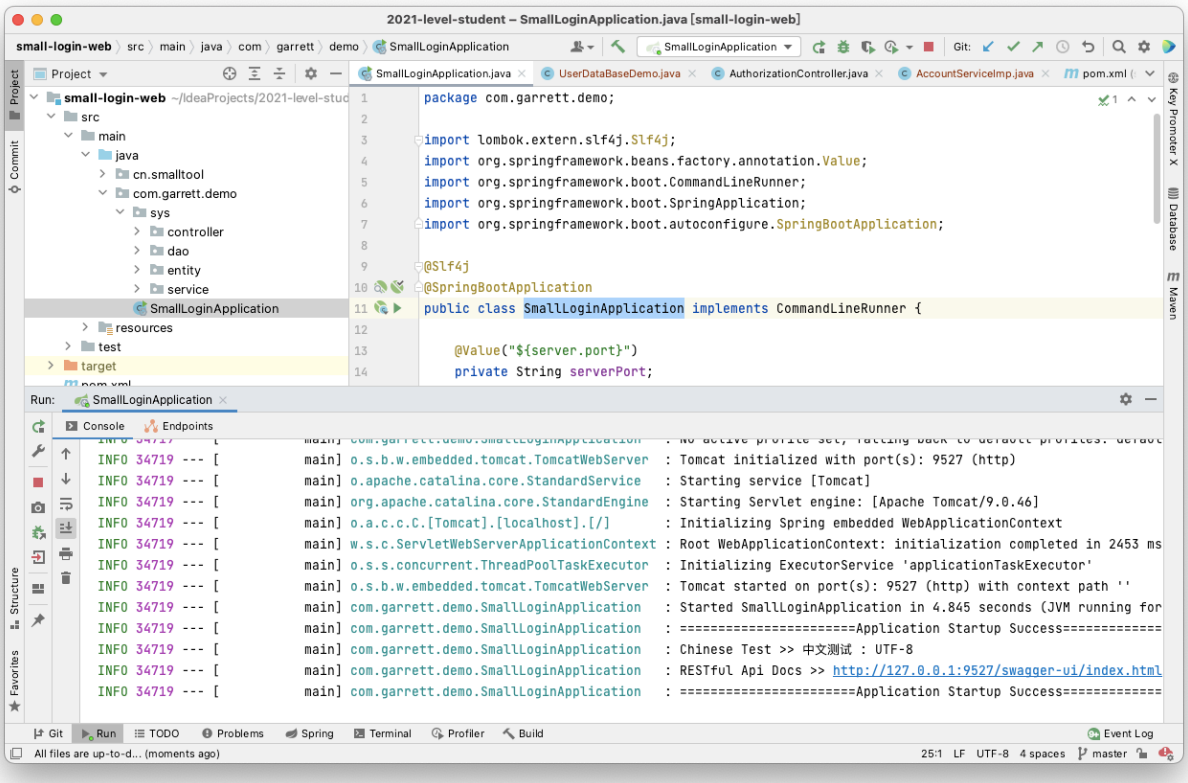
```
│   │   │   │           ├── RandomUtil.java
│   │   │   │           └── StrUtil.java
│   │   │   └── com
│   │   │       └── garrett
│   │   │           └── demo
│   │   │               ├── SmallLoginApplication.java
│   │   │               └── sys
│   │   │                   ├── controller
│   │   │                   │   └── AuthorizationController.java
│   │   │                   ├── dao
│   │   │                   │   ├── UserDao.java
│   │   │                   │   └── UserDataBaseDemo.java
│   │   │                   ├── entity
│   │   │                   │   ├── Account.java
│   │   │                   │   └── UserInfo.java
│   │   │                   └── service
│   │   │                       ├── AccountService.java
│   │   │                       └── AccountServiceImp.java
│   │   └── resources
│   │       └── config
│   │           └── application.yml
```



修改 AccountServiceImp 类:

```
// 其它导入包……
import org.springframework.stereotype.Service;


// 增加注解
@Service
public class AccountServiceImp implements AccountService{
        // 原有代码片段……
}
```

- 最后一步，让根据你编辑器的检测，确保没有显式的红色波浪线标红报错
- 找到 SmallLoginApplication 主运行类中的main函数，点击运行。



如果成功出现以上类似代码，说明程序已被开启。

四. 测试

现在我们来验证一下
画面切换到 HBuilderX工具这边，我们在输入框输入测试用的账号和密码（demo中使用的是admin：admin，如有自定义，请自行变通）

点击登录

如果出现登录成功的提示，那就说明测试通过啦～

对了，顺便一提，如果你的手机连的是笔记本电脑的wifi或者是在同一局域网下，你甚至可以在手机浏览器中查看效果哦～

手机浏览器访问地址：http://你电脑的ip地址:8080/

常见问题

Q：前后端代码都跑起来了，但是前端登录并没有出现理想的登录成功效果?
A：由于所有代码基本已经配置好环境了，请务必保证两个项目的运行都是在同一台电脑上

五. 总结

本章节我们主要了解UI/前端界面的对接过程。也希望读者完成这一部分的同时能够给自己带来不一样的成就感。希望能够让你提起对开发的兴趣。如果操作复现过程中出现一些问题，也不要灰心，可以请教一下已经完成的同学，或者询问下助教。

阶段检测作业01

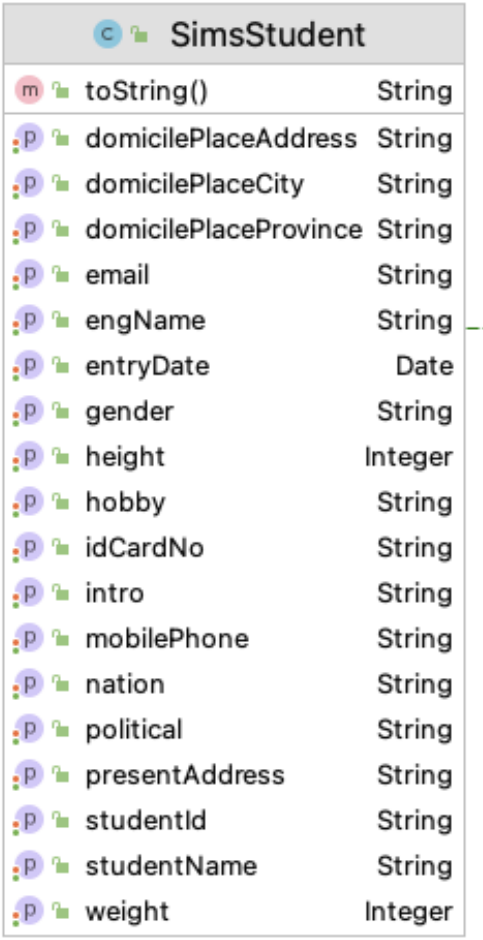
一. 实训任务1

截止上一章节的学习，我们已经简单完成了登录的整体流程。接下来请同学们根据要求完成注册功能模块。
PS. 可拉取 small-login-step-06 模块的代码

二. 实训要求

1. 工程结构

已提供Account账户类，SimsStudent学生类
(1) 按照 small-login-step-06 的工程结构创建自己的模块，模块名是自己的学号（9位）
(2) 请构建UserDao接口的实现类，并重写方法。取名UserDataBaseDemo
(3) 请构建AbstractAccountService抽象类的实现类，并重写方法。类名可自拟，或取名AccountServiceImp。在Service层的实现类上加上@Service注解，如图T01-04
(4) 在ApplicationTest类中测试其注册过程。注意解藕（高内聚，低耦合）



2. 业务逻辑 T01–3

- 如图T01–03所示（仅供参考），用户可选择登录，注册等模式。

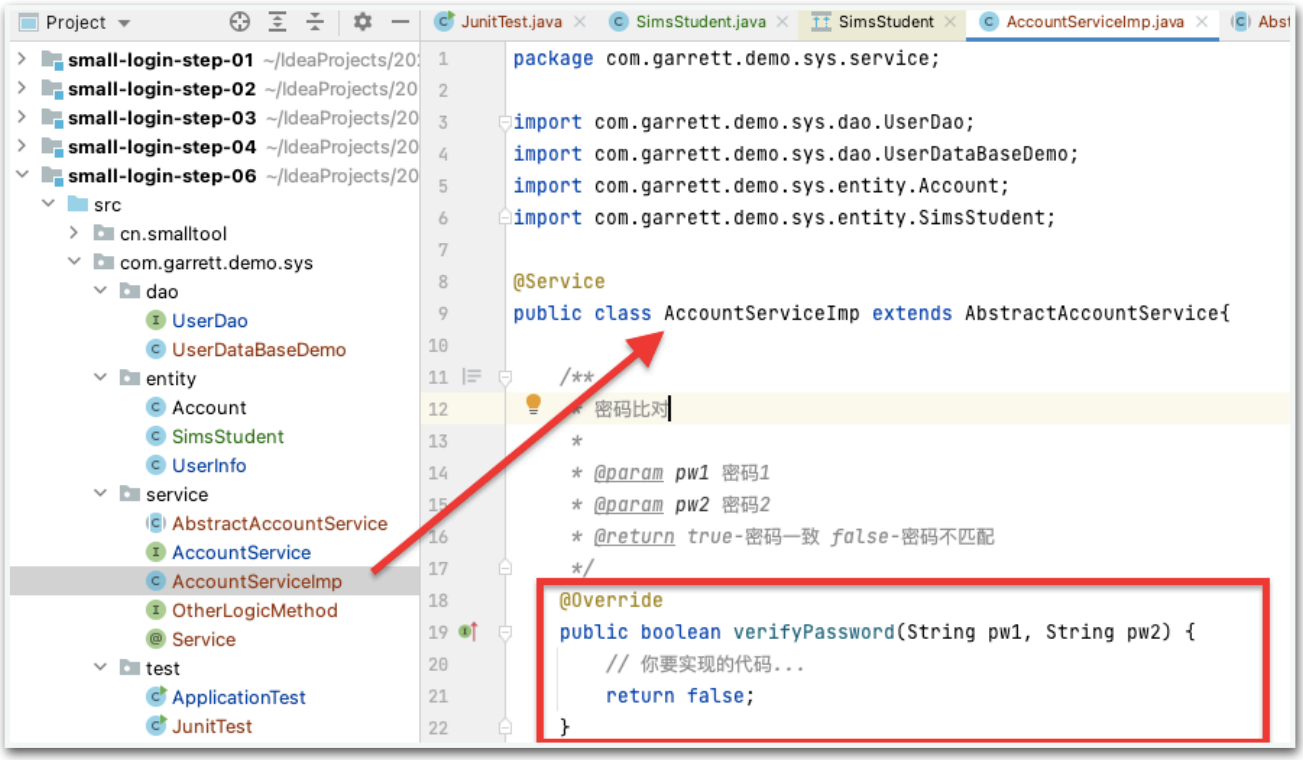
- 进入注册模式后，需要用户提供账户名（即学号），密码等信息。其中：

1) 验证码审核过程。注意，密码需要输入两次，判断是否一致。验证码可使用之前的工具类，没必要额外写多余的代码或实现方法。
2) 其中身份证、学号、手机号、性别为必填项

图片仅供参考

3. 代码考查点

1) 密码比对(用户输入两次密码，用于确认密码是否一致),继承并实现verifyPassword方法



T01–04

2）数据的写入和读取，如图T01–5 ，创建UserDao的实现类，重写所有方法，实现类必须取名UserDataBaseDemo。

JunitTest.java | src/.../UserDao.java | UserDataBaseDemo.java

```java
package com.garrett.demo.sys.dao;

import ...

public interface UserDao {

    /**
     * 根据学号获取学生信息
     * @param stuId 学号
     * @return 成功-用户信息 失败-null
     */
    SimsStudent getStudentById(String stuId);

    /**
     * 根据登录账号获取账户资源信息
     * @param account 账户名
     * @return 成功-账户信息 失败-null
     */
    Account getUserByAccount(String account);

    /**
     * 注册账户信息
     * @param account 账户
     * @return 1-成功插入一条数据 0-数据创建失败
     */
    int addAccount(Account account);

    /**
     * 添加用户信息
     * @param userInfo 用户信息
     * @return 1-成功插入一条数据 0-数据创建失败
     */
    int createUser(SimsStudent userInfo);
```

T01–5

3）数组空间设定大小为[3]，作为UserDataBaseDemo的私有属性，

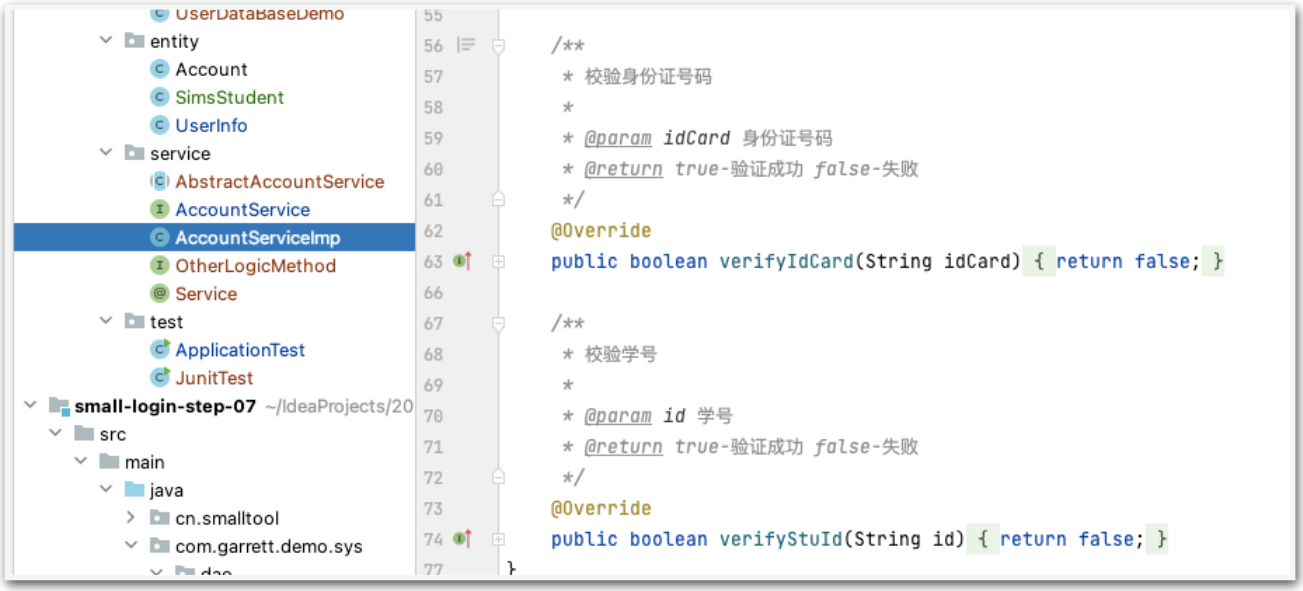
```
private SimsStudent[] students = new SimsStudent[3];

private Account[] accounts = new Account[3];
```

```java
    /**
     * 注册账户信息
     * @param account 账户
     * @return 1-成功插入一条数据 0-数据创建失败
     */
    int addAccount(Account account);

    /**
     * 添加用户信息
     * @param userInfo 用户信息
     * @return 1-成功插入一条数据 0-数据创建失败
     */
    int createUser(SimsStudent userInfo);
```

如果超出容量上限，则返回0；

4）主键id/或者学号重复问题，则插入失败。返回值为0
5）密码强度
　　最短6位，最长16位
6）身份证、学号、手机号、性别
- 身份证需要符合18位编码规则
- 身份证携带的性别信息应与用户输入的性别保持一致
- 学号需满足9位长度
- 手机号允许中国区号开头（+86）
- 手机号需符合中国号码规则
- 性别字典：1–表示男性 0 – 表示女性

以上规则有一条不通过，则数据不能写入
需在service层实现接口方法：

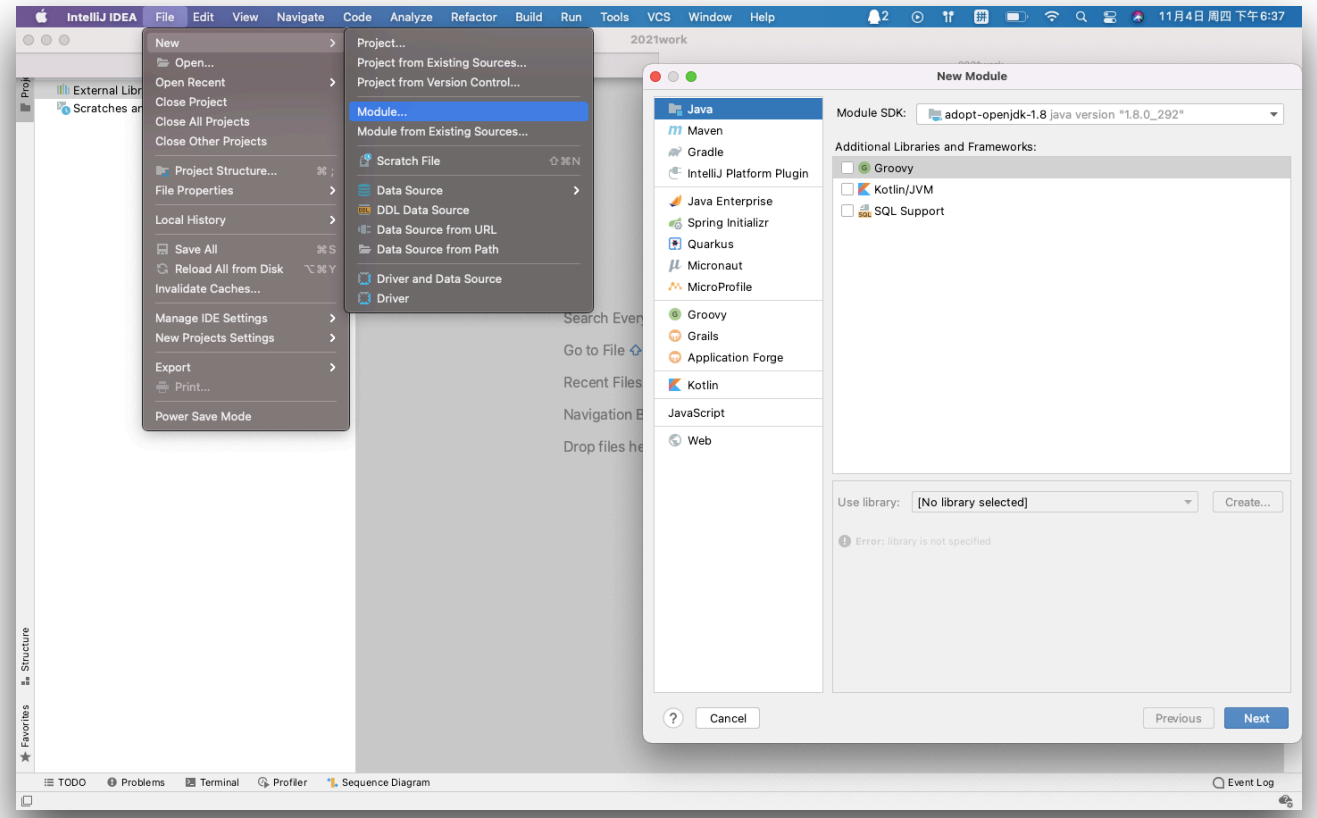


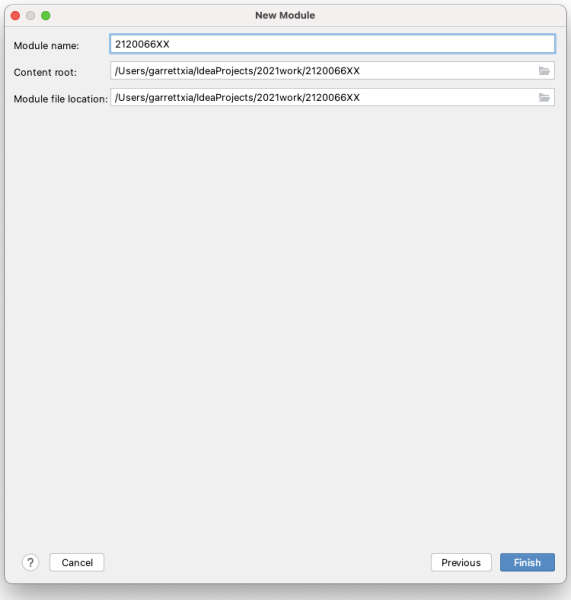
4. 操作方式
可以是命令行，也可以是html等其他用户交互方式

5. 扩展
学有余力的同学，可以对接下前端代码，亦可以完成忘记密码的功能等其他功能，属于加分项

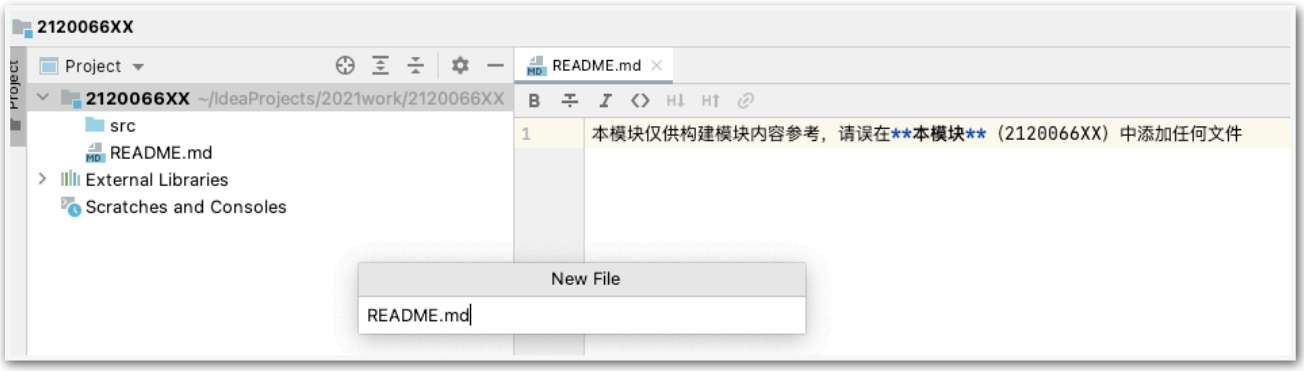
三. 作业提交规范

1）请参照“使用Gitee提交实训作业的流程”，先fork仓库（https://gitee.com/garrettxia/work_2021），即在码云上创建了自己的远程仓库
2）将仓库clone到本地，成为IDEA的项目，work_2021仓库中已有的内容，在本地不允许做任何修改和删除
3）在原有项目的基础上添加新模块，以学号命名（9位）


IntelliJ IDEA
File
Edit
View
Navigate
Code
Analyze
Refactor
Build
Run
Tools
VCS
Window
Help
New
Open...
Open Recent
Close Project
Close All Projects
Close Other Projects
Project Structure...
File Properties
Local History
Save All
Reload All from Disk
Invalidate Caches...
Manage IDE Settings
New Projects Settings
Export
Print...
Power Save Mode
Project...
Project from Existing Sources...
Project from Version Control...
Module...
Module from Existing Sources...
Scratch File
Data Source
DDL Data Source
Data Source from URL
Data Source from Path
Driver and Data Source
Driver
New Module
Java
Maven
Gradle
IntelliJ Platform Plugin
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Groovy
Grails
Application Forge
Kotlin
JavaScript
Web
Module SDK:
adopt-openjdk-1.8 java version "1.8.0_292"
Additional Libraries and Frameworks:
Groovy
Kotlin/JVM
SQL Support
Use library:
[No library selected]
Create...
Error: library is not specified
Cancel
Previous
Next
TODO
Problems
Terminal
Profiler
Sequence Diagram
Event Log


New Module
Module name:
2120066XX
Content root:
/Users/garrettxia/IdeaProjects/2021work/2120066XX
Module file location:
/Users/garrettxia/IdeaProjects/2021work/2120066XX
Cancel
Previous
Finish

4）在IDEA的模块中添加文档，统一命名：README.md



Markdown内容要求:

0. 总体字数在200字以上
1. 应用操作使用说明
2. 所学知识点概括（小结回顾）
3. 扩展功能说明（可选，加分项）

5）将新增/修改的代码提交到本地仓库（PS.这里建议每修改一小部分，及时地将修改提交到本地仓库）

6）推送到远程仓库

7）最后通过PR提交。提交后，若管理员没有处理，不能再次PR

第 07 章：落地数据对接

一. 不能写死

你这代码，可不能写死了呀！

依照项目落地经验来看，我们在承接紧急的产品需求时候，通常会选择在原有同类项目中进行扩展，如果没有相关类型项目的储备，也可能会选择临时搭建出一个工程来实现产品的需求。但这个时候就会遇到非常现实的问题，选择完整的设计和开发就可能满足不了上线时间，临时拼凑式的完成需求又可能不具备上线后响应产品的临时调整。

上线后的调整有哪些呢？项目刚一上线，运营了还不到半天，老板发现自己的配置的活动好像金额配置的太小了，用户都不来，割不到韭菜呀。赶紧半夜联系产品，来来来，你给我这改改，那修修，把人均优惠 1 万元放大大的，把可能两字缩小放在后面。再把优惠的奖金池配置从 10 元调整 11 元，快快快，赶紧修改，你修改了咱们能赚 1 个亿！！！

好家伙，项目是临时开发堆出来的，没有后台系统、没有配置中心、没有模块拆分，老板一句句改改改，产品来传达催促，最后背锅的可就是研发了。你这不能写死，这优惠配置得抽出来，这文案也后台下发吧，这接口入参也写死了，再写一个新接口吧！ 一顿操作猛如虎，研发搬砖修接口，运营折腾好几宿，最后 PV150！

无论业务、产品、运营如何，但就研发自身来讲，尽可能的要不避免临时堆出一个服务来，尤其是在团队建设初期或者运营思路经常调整的情况下，更要注重设计细节和实现方案。哪怕去报风险延期，也不要让自己背上一个明知是烂坑还要接的活。

而本章节说到不把代码写死，我们需要继续在第六章的基础上延伸数据的对接，如一个新用户的注册，是否可以满足我们进行持久化操作，保证服务暂停重启之后数据依然存在呢?

二. 目标

本章节侧重于提供数据对接的案例给同学们。以供同学们根据自己的需要实现不同的数据源对接，或者额外扩展自定义的持久层数据对接。

我们将会：
- 演示关于text，properties文件类型的IO读写
- 演示以Excel表格类型模拟传统数据库二维表结构方式的数据存储

三. 设计

首先，我们提供使用Text文本内容的进行存储，我们以逗号为分隔符，例如：
1,1,212006677,夏晶玮,Garrett,350124199*******14,177****7676,1,1996-01-01,https://img.zcool.cn/community/01c3d55e7713a8a80120a89510edcc.jpg ,172,120,汉,群众

四. 实现

数据源类型

TEXT

采用.txt格式的,提前在模块7的资源文件下创建好sims_student.txt文件。

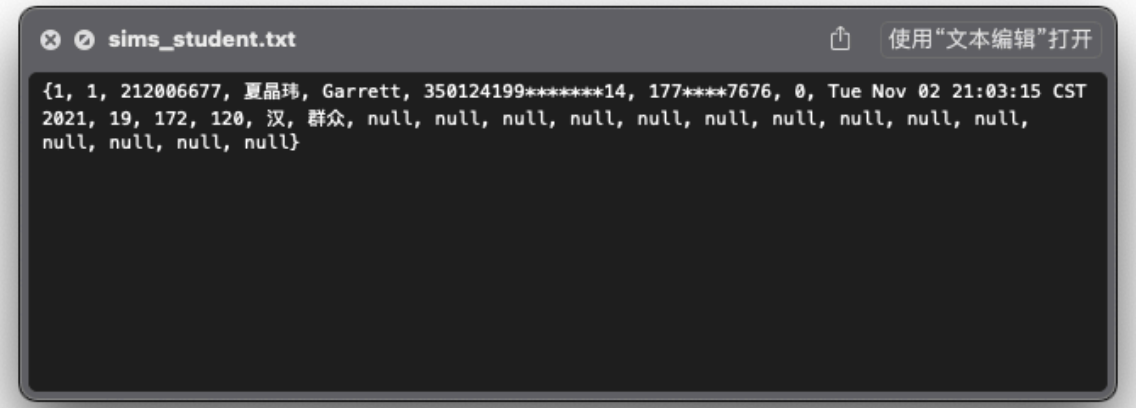
或者可直接执行DataBaseInitTest测试类中的初始化方法，由程序直接生成。
关于Java IO操作这一块的知识点，以下代码供同学们参考

1. UserDaoTextImp 接口实现

com.garrett.demo.sys.dao.UserDaoTextImp

```
public class UserDaoTextImp implements UserDao{

      // … 其他代码片段

    /**

     * 添加学生信息

     */

    @Override

    public void createUser(SimsStudent student) {

        String path = "./small-login-step-07/src/main/resources/
sims_student.txt";

        File file = new File(path);

        OutputStreamWriter osw = null;

        if(!file.exists()){

            System.out.println("数据源不存在!!!");

            throw new RuntimeException("数据源不存在");
```

```
        }


        try {
            // 创建基于文件的输出流
            FileOutputStream fos = new FileOutputStream(file,true);
            // 指定编码格式，以免读取时中文字符异常
            osw = new OutputStreamWriter(fos, "UTF-8");
            osw.write(student.toString()+"\n");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(null != osw) {
                try {
                    osw.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```



最终写入文本内容如图所示。

文本文件数据读取可用字符流或字节流处理：

```
/**
 * 根据用户编号获取用户信息
 */
@Override
public SimsStudent getUserById(String id) {
    String path = "./small-login-step-07/src/main/resources/sims_student.txt";
    File file = new File(path);
    try {
        if(file.isFile() && file.exists()){ //判断文件是否存在
            InputStreamReader read = new InputStreamReader(
                    new FileInputStream(file),"UTF-8");//考虑到编码格式
            BufferedReader bufferedReader = new BufferedReader(read);
            String lineTxt = null;
            while((lineTxt = bufferedReader.readLine()) != null) {
                SimsStudent stu = recordFormat(lineTxt);
                if(stu.getStudentId().equals(id)) return stu;
            }
            read.close();
        }else{
            System.out.println("找不到指定的文件");
        }
    } catch (Exception e) {
        System.out.println("读取文件内容出错");
        e.printStackTrace();
    }
    return null;
}

/**
 * 单行数据记录处理
```

```
     * @param row 行记录
     * @return 学生信息
     */
    SimsStudent recordFormat(String row){
        SimsStudent stu = null;

        int beginIndex = row.indexOf("{") + 1;
        int endIndex = row.indexOf("}");
        String content = row.substring(beginIndex,endIndex);
        String[] split = content.split(",");
        String[] fields = Arrays.stream(split).map(String::trim).toArray(String[]::new);
        if(fields.length > 0){
            stu = new SimsStudent();
            stu.setCollegeId(fields[0]);
            stu.setClassId(fields[1]);
            stu.setStudentId(fields[2]);
            stu.setStudentName(fields[3]);
            stu.setEngName(fields[4]);
            stu.setIdCardNo(fields[5]);
            stu.setMobilePhone(fields[6]);
        }
        return stu;
    }
```

以上代码片段已实现Text文本数据的新增和读取数据的功能，同学们可根据已学习的String字符串处理和IO操作，实现学生信息的修改和删除。业务场景可应用于个人信息的更新维护，或学生管理系统的的记录删除维护等。

EXCEL

采用 .xsl 格式的, 由程序直接生成，本小点使用阿里巴巴的easyExcel工具类。主要用于锻炼同学们的课外扩展知识。

1. 导入第三方包或maven项目引入
    * 离线版jar已提供在附件中
    * Maven可使用以下包依赖

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>easyexcel</artifactId>
    <version>3.0.5</version>
</dependency>
```

2. 使用第3方提供的API

这里仅仅一些最简单的例子，需要完整的上手请参照：文档

读Excel

DEMO代码地址：https://gitee.com/garrettxia/work_2021/blob/master/2120066XX/src/main/java/com/garrett/demo/sys/dao/UserDaoExcelImp.java

```
/**
    * 最简单的读
    * <p>1. 创建excel对应的实体对象 参照{@link DemoData}
    * <p>2. 由于默认一行行的读取excel，所以需要创建excel一行一行的回调监听器，参照{@link DemoDataListener}
    * <p>3. 直接读即可
    */
   @Test
   public void simpleRead() {
       String fileName = TestFileUtil.getPath() + "demo" + File.separator + "demo.xlsx";
       // 这里 需要指定读用哪个class去读，然后读取第一个sheet 文件流会自动关闭
       EasyExcel.read(fileName, DemoData.class, new DemoDataListener()).sheet().doRead();
   }
```

写Excel

DEMO代码地址：https://github.com/alibaba/easyexcel/blob/master/src/test/java/com/alibaba/easyexcel/test/demo/write/WriteTest.java

```
/**
    * 最简单的写
    * <p>1. 创建excel对应的实体对象 参照{@link com.alibaba.easyexcel.test.demo.write.DemoData}
    * <p>2. 直接写即可
    */
   @Test
   public void simpleWrite() {
       String fileName = TestFileUtil.getPath() + "write" + System.currentTimeMillis() + ".xlsx";
       // 这里 需要指定写用哪个class去读，然后写到第一个sheet，名字为模板 然后文件流会自动关闭
       // 如果这里想使用03 则 传入excelType参数即可
       EasyExcel.write(fileName, DemoData.class).sheet("模板").doWrite(data());
```

```
}
```

五. 测试

这一步我们只需将原有的Dao层做一个调整对接，保证数据持久化。重启后依然能够使用过去保存的数据即可。

六. 总结

- 本章节的主要实现数据持久化的操作，巩固上一章节基础。

阶段检测作业02

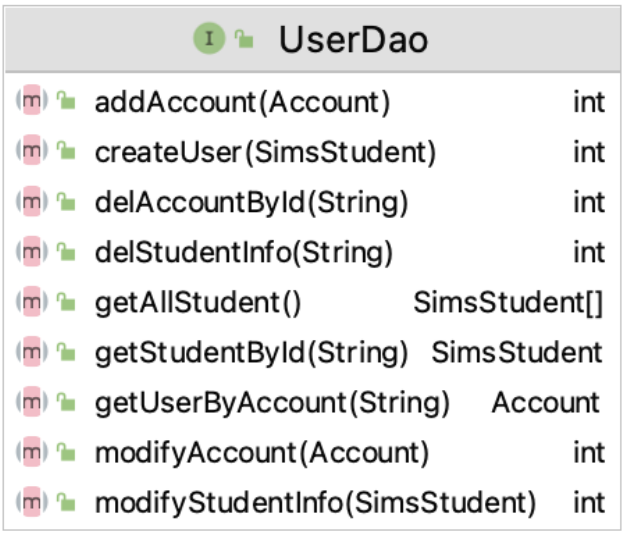
一. 实训任务2

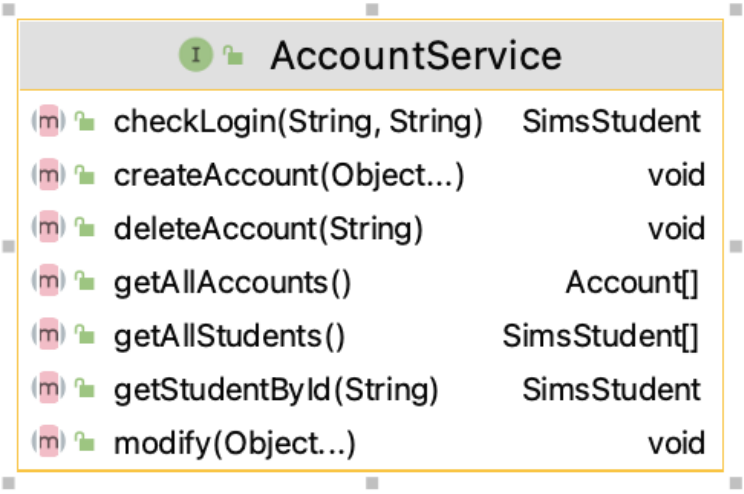
我们已经完成了登录注册的整体流程。接下来请同学们根据要求补充完整的增删改查以及完成数据的持久化。

二. 实训要求

1. 工程结构

(1) 按照 第一次提交作业 的工程结构基础上继续编写代码
(2) 补充完善UserDao接口以及AccountService接口





具体功能点的说明和用途可参考

https://gitee.com/garrettxia/work_2021/blob/master/2120066XX/src/main/java/com/garrett/demo/sys/dao/UserDao.java

https://gitee.com/garrettxia/work_2021/blob/master/2120066XX/src/main/java/com/garrett/demo/sys/service/AccountService.java

(3) 请对UserDao接口的实现类进行补充。类名UserDataBaseDemo不变
(4) 请对AccountService接口的实现类进行补充。
(5) 在ApplicationTest类中测试其查看和修改的连贯过程。注意解藕（高内聚，低耦合）

2. 业务逻辑(任选，组合可选，至少完成一项)

* 可以查看所有用户账户及学生信息
* 可以修改学生信息数据

* 可以删除用户数据

3. 数据存储方式（任选其一）：

* 保持任务1原有的数组缓存方式
* 将用户注册信息持久化：当程序重启后，依然可以使用注册过的账号登录。可以是text文本，或excel存储方式

4. 操作方式
可以是命令行，也可以是html等其他用户交互方式

三. 作业提交规范

1） 在原有你的学号为命名的模块基础上补充或者修改代码。

2）补充 README.md 文档内容

Markdown内容要求:

0. 添加二级标题：任务2
1. 总体字数在200字以上
2. 本次任务采用的数据存储的方式
3. 本次任务完成的功能点。如，删除功能还是查询功能，或者修改功能等。
4. 不允许粘贴代码，文字描述即可
5. 所学知识点概括（小结回顾）

3）将新增/修改的代码提交到本地仓库（PS.这里建议每修改一小部分，及时地将修改提交到本地仓库）
4）推送到远程仓库
5）最后通过PR提交，本次提交到 **student2**分支。
6）慎用PR，一旦PR被合并后，再次补充提交PR，将大概率产生冲突，请尽量保持一次PR。