**基于Android和java web编程的护眼应用**

**摘要**

随着电子产品的普及，工作、学习压力的日趋增大，人们眼睛疲劳过度问题愈加严重。研究表明，在中国，由于眼部疲劳过度引起青少年们的眼部疾病占60%以上，同时，电子产品的普及使得青少年们从小变受到电子产品辐射的影响，这更加剧了他们眼部疲劳、用眼过度的问题。正因如此，护眼问题日益重要，如何科学地用眼、护眼，让自己的眼睛健康、明亮起来，也日益重要。本应用试图通过专业的而科学的方法为用户提供一种科学合理的眼部护理方案，前期通过应用的眼部测试功能，运用图像识别等技术，自动检测用户眼部问题，提供专业的方案供用户进行眼睛视力（包括辨色能力、明视能力、识光能力等）的测试，并根据结果，采用云端技术为用户提供合理的专家建议以及护眼方案，并根据用户目前用眼情情况，利用相关程序技术动态调整电子产品亮度、过滤蓝光，动态提醒用户在适当的时候进行眼部的休息，为用户提供一个科学、方便的护眼平台。

市场上充斥着各种护眼应用，然而，根据调查，这些应用只是简单地提供蓝光过滤，并没有针对用户目前的用眼状况，动态地帮助用户科学用眼，对于眼睛的保护作用微乎其微。本应用通过科学的检测，及时根据用户的眼部状态，提供合理科学的用眼建议；在用户方用眼建议方面，通过云端技术，动态地推送相关专家的用眼建议和护眼方案，对症下药，达到科学护眼的目的。

**关键字：android，java web，web框架，mysql，redis**

目录

**第一章 绪论3**

1.1课题背景3

1.2系统客户端功能介绍3

1.3系统服务端功能介绍4

**第二章 可行性分析4**

2.1技术可行性分析4

2.2经济可行性分析5

**第三章 相关技术简单介绍5**

3.1Android操作系统5

3.2MVC模型5

3.3Spring框架体系6

3.4数据库mysql和redis7

3.5数据库持久化框架8

3.6网络请求与json8

**第四章 系统设计架构原理9**

4.1系统总体架构9

4.2客户端模块架构10

4.3服务端模块架构11

**第五章 模块具体实现分析12**

5.1视力测试模块12

5.1.1色盲测试模块12

5.1.2散光测试模块14

5.1.3明视距离测试模块14

5.1.4敏感度测试模块17

5.2知识库模块18

5.2.1饮食习惯模块18

5.2.2提问模块24

5.3用眼数据24

5.4个人中心、蓝光过滤28

**第一章 绪论**

**1.1课题背景**

随着互联网发展，出现越来越多云端产品，而在web端编程，java作为当之无愧的首选编程语言，它拥有众多的开源框架、丰富的文档资料以及活跃的编程社区，这使得利用java进行web开发尤其方便、高效且稳定。另外，android系统作为移动端用户最多的操作系统，它开源、稳定，加上Google等科技巨头的支持，android应用开发的类库、组件丰富且成熟，开发android应用非常便捷、高效，因此，我们的应用选用了android作为客户端，java作为服务端的搭配选择。

另外，护眼应用虽然软件市场上有不少，但是基本上都是功能较为单一且针对性不强。本款护眼应用，集视力测试、蓝光过滤、护眼相关知识等多个功能，一方面满足用户基本的护眼需求，另一方面也使得护眼的过程更有针对性和可行性。

**1.2系统客户端功能模块介绍**

客户端模块按应用的功能分为以下几大模块：视力测试模块、用户模块、知识库模块、蓝光护眼模块、用眼数据模块。

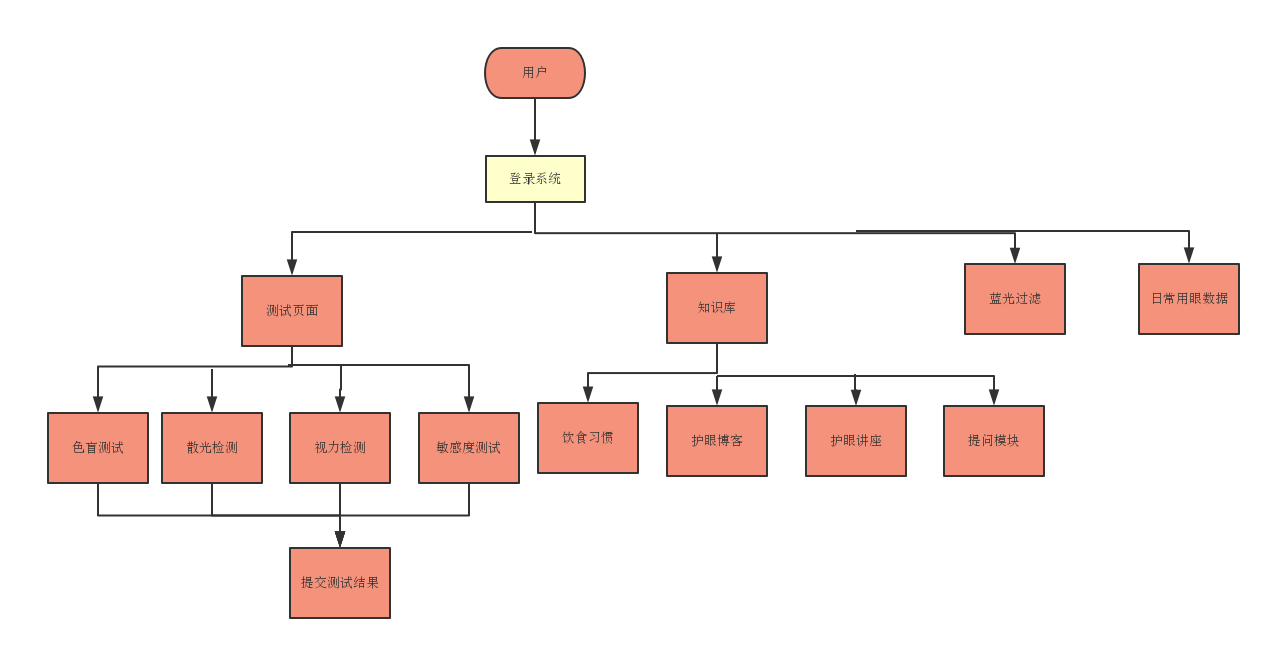
视力测试模块主要提供色盲、散光、视力测试和敏感度测试。主要测试方式是通过图片素材，根据用户的正确率判断患病可能性。其中，敏感度测试主要是测试用户用户分辨相近物体的能力，这里主要用相近的颜色块对用户进行测试。在用户测试完毕，可以选择把测试结果提交到云端进行报保存，方便专家对其用眼情况进行分析。下图是该模块的相关截图：

个人中心主要用于用户相关的数据浏览，主要包括修改个人资料、当天的用眼情况、眼部测试数据、专家交流记录和消息中心。其中，专家交流记录主要记录和用眼专家的聊天记录，用户可以在这里向专家进行用眼咨询。

知识库主要是用眼相关的文章以及建议，用户可以在这里查看专家们的用眼文章。文章类型有饮食习惯、护眼博客文章，同时也会发布一些线上的讲座信息，另外，用户也可以对专家进行专门的提问。

蓝光过滤模块主要用于调节手机屏幕达到护眼效果，主要有几种模式选择：夜间模式、护眼模式和正常模式，另外，用户也可以根据自己的喜好，选择其他的背景光调节。

用眼数据主要记录用户使用手机相关的次数，例如今天开屏次数、连续使用手机时间、本次开屏使用手机时间等等。

下面是主要的应用的功能分布简略图：

**1.3系统服务端简介**

后端的功能模块主要分为：用户模块，测试模块、文章模块，模块对应前端客户端的各个模块，后端对应模块对前端的模块提交的数据进行处理、存储等后台操作。

**第二章 可行性分析**

**2.1技术可行性**

目前，web开发技术领域总，java web体系是主流的技术，很多开源框架可以免费使用，同时开源社区也很多现成的技术资料可以参考。我们系统基于Android+java后端的开发，技术上已经非常成熟，开发集成环境也有很多选择，各种java类库应有尽有。另外，开源的数据库系统、ORM框架也应有尽有，技术成熟度较高，开发门槛也比较低，所以，开发难度上完全是可控的。

另外，服务端利用linux作为操作系统，加上tomcat的容器，对于本应用而言，运行环境完全是很好搭建的。

**2.2经济可行性分析**

经济上而言，开发软件除了人力外，基本无经济支出，加上阿里云的廉价学生服务器和开源社区上应有尽有的开源产品，开发系统更多是脑力层面的成本支出，所以，开发该系统的经济成较低，有很强的可行性。

**第三章 相关技术简单介绍**

**3.1 Android操作系统**

Android作为目前移动端使用最多的操作系统，具有代码开源、应用程序易于开发、特性丰富等优点。Android操作系统内核其实是基于Linux内核进行开发的，支持多任务、多种网络制式（包括WiFi/蓝牙/移动数据等等）。另外，由于Android 基本用java进行开发，而java具有很强的面向对象特性，加上Google丰富的Android类库、插件和强大的集成开发环境， Android应用的开发是非常低门槛、高效率的。

利用Android集成开发环境，特别是Google官方推荐的AndroidStudio，它很好地封装了整套针对Android的开发组件，加上先进的Gradle 依赖管理系统，开发的过程让人有种赏心悦目的感觉，高效而优雅。

在Android开发中，优秀的Google大神把Android的类库封装得非常优雅，利用xml进行布局文件的设置，通过java 语言进行页面动态操作，分离了界面和业务，降低了耦合度，灵活易于扩展，这种Android应用开发方式也是一种非常了不起的编程设计思想。

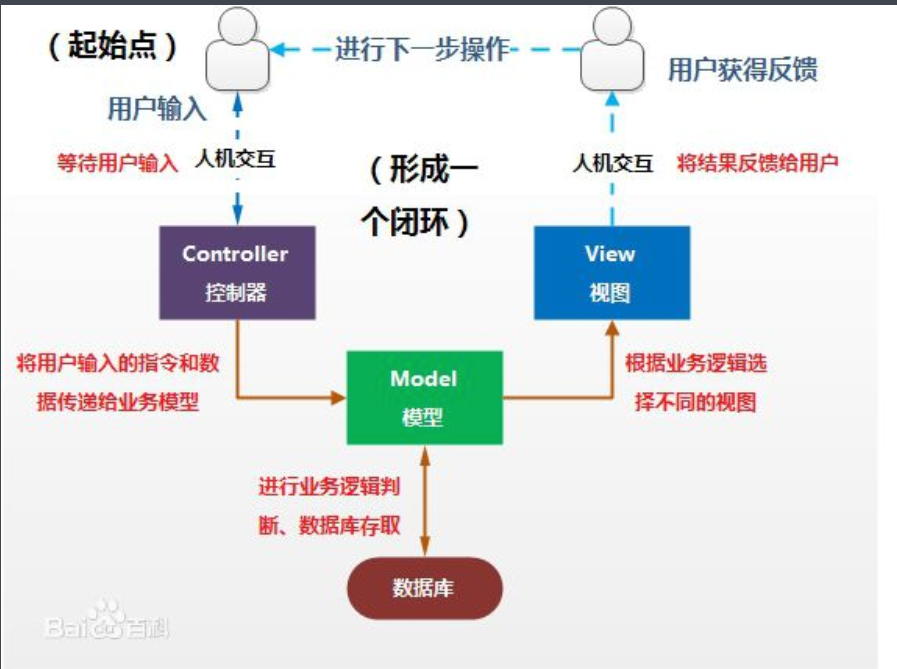
**3.2 MVC模型**

MVC的分层思想是软件设计中最为重要的解耦思想之一，通过MVC的模型思想，可以大大降低软件各个模块（界面模块、业务模块等）的耦合度，增加代码复用度，提高代码模块的可读性。

MVC其实是下列三个英文的缩写：Model、View、Controller，分别代表的意思是模型、视图和控制器。Model代表的是编程里面的对象，可以理解为数据以及逻辑的集合，对应业务中现实的具体模型和行为；View其实是界面，我们平常可以看得到的软件界面、浏览器界面都可以理解为软件架构中的View；Controller其实是业务代码的入口，它负责把View传过来的数据进行分发，交给不同的业务逻辑代码进行处理，并返回前端处理完成的结果，可以理解为一个路由器。总的来说，Model是数据，View代表界面，而Controller代表逻辑代码的处理入口。

MVC的架构思想，很好地解决了传统软件开发中界面和逻辑代码混在一起的问题。在软件开发一开始，人们并没有MVC的思想，而是把所有代码放在一个地方，页面实现代码、逻辑代码混在一起，加大了系统扩展的难度，也不利于代码复用，而MVC设计思想的出现，很好地解决了上述问题。

下面是MVC的一般架构思路原理图：



在上图中，Controller负责接收用户的数据，进行数据分发；Model层是核心的数据层；View是视图，负责展示数据。

**3.3 Spring框架体系**

由于后台是用java体系的技术进行后台代码开发，而spring体系的框架又是java语言中的主流框架，所以后台选用的框架主要是spring+springMVC的形式。Spring 在我们后台系统中，其实是充当一个对象管理者的角色，它负责业务对象和数据库事务的管理；SpringMVC其实是利用spring之后实现了上面所说的MVC模型思想的一个技术框架， 它是整个后台系统的请求入口、数据出口。

Spring 框架具有独到的优势，主要以其独特的控制反转以及面向切面式编程思想使得web后端编程变得灵活、高效且易于扩展、复用。控制反转的思想可以很方便地把对象进行注入使用，对象的初始化、销毁等过程均由spring完成，我们自己本身只需要专注于核心业务即可；而AOP思想则是一种切面式的编程，也是一种代码复用的手段，主要是利用动态代理的方式对代码进行复用：通过一个代理对象，调用实际的方法，在调用实际方法前后插入对应的复用代码，而数据库事务管理正是用到这一思想，在业务前后插入了事务的开启、关闭相关代码，达到复用效果。

SpringMVC是spring 技术体系下的一个MVC框架。由于其易于使用，且和spring配合良好，轻量、多功能、灵活，使其非常流利。在本项目中，springMVC主要是用于控制层入口，返回给前端的数据一般是以json格式的一串字符串。

**3.4数据库mysql和Redis**

在后端中，我们选用的数据库系统主要是包括两个：关系型数据库mysql和非关系型的数据库Redis。Mysql用来存储主要数据源；Redis主要用于消息队列、用户验证类信息存放等。

Mysql数据是一个关系型的数据库，作为互联网行业的数据库系统标配，它具有很多很好的特性：灵活、支持事务、开源免费等，尽管其比较小巧，没有oracle那样高度集成化的多功能，但是通过中间件和数据库集群的方式完全可以解决规模、实时性等一系列问题。

Redis数据库是一种内存型的非关系型数据库，相比于关系型数据库，它最大的优点就是速度快，所以对应身份验证的信息，为了避免每次请求验证身份中读取数据库的时间开销，我们吧token放在了redis中。它其实是充当一个常用内存数据的缓存、管理者的角色。

**3.5数据库持久化框架**

在数据库持久化框架中，我们主要采用了mybatis这个框架。由于关系型数据库的编程可以理解为面向过程的一种编程，而java是一种典型的面型对象的语言，所以，为了使数据库的操作具有面向对象的特性，便我们把数据库中的表数据转换为一个个java对象以及把一个个java对象转换为数据库表中的数据，而数据库持久化框架便是充当该角色。

数据库持久化框架又很多，例如大名鼎鼎的hibernate，互联网闻名的mybatis等等。由于hibernate使用过于复杂，配置繁琐，我们便选用了相对较为轻量级的mybatis框架。

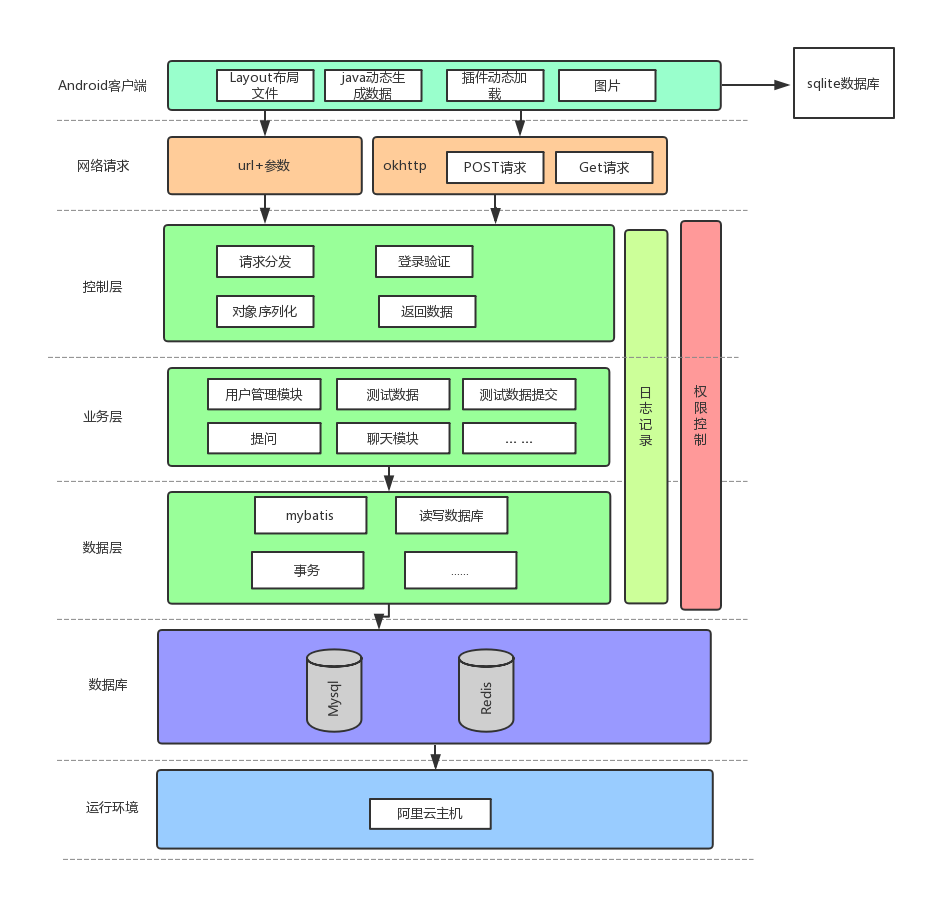
**3.6网络请求与json**

网络请求主要采用http的请求协议，在Android客户端主要使用okhttp这个网络框架进行请求发起和数据接收，okhttp是一个非常强大的http网络请求框架，支持异步发起请求以及多种请求方式。在和后端交互数据中，主要采用json字符串的形式进行交互。Json字符串是一种非常简洁、轻便的数据表现形式，主要是以key-value的形式对数据进行序列化。客户端接收到json字符后，通过对应json处理框架便可把数据提取出来，显示在页面上。

Json是一种字符串数据的表示形式，主要以key-value的形式，对数据进行序列化表示。Json字符具有简洁明了、易于解析等优点，用于数据传输具有非常大的优势。

**第四章 系统设计原理**

**4.1系统总体架构**



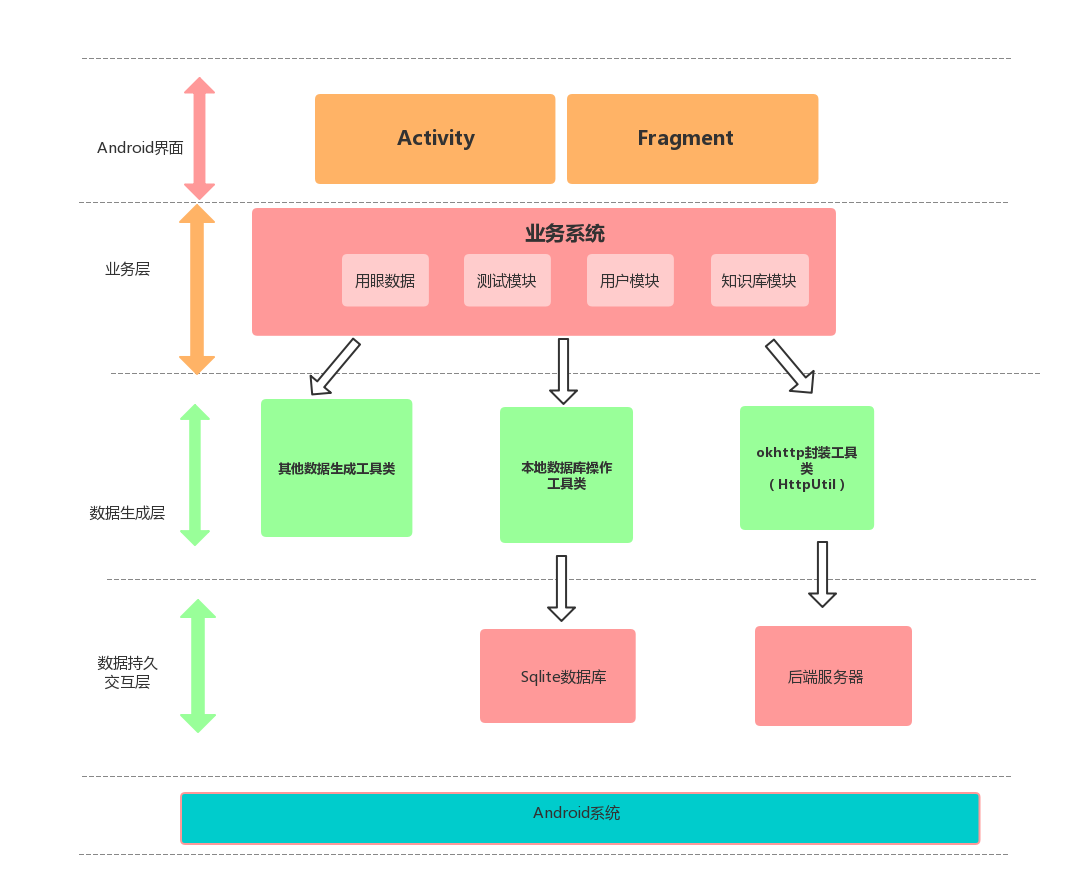
系统的总体架构图如上。总的请求过程：Android客户端，在用户进行输入操作并提交数据以及发起请求之后，Android端通过okhttp这个http请求框架发起网络请求或者请求本地的sqlite数据库查询数据。

如果是需要请求后端数据，则直接发起http请求，把数据提交到web容器。后端系统入口是springMVC这个框架所实现的Controller层，数据经过service、DAO之后最终到达数据库，在进行数据库查询、增加等操作之后，操作结果的数据集会沿路返回，在Controller调用序列化相关的框架对数据进行序列化处理后以json字符串的形式，通过网络输出流返回给客户端。

如果请求无需请求后台系统，直接操作android本地数据库即可的话，那么，客户端将会直接调用封装好的本地数据库操作工具类进行本地数据库操作。

**4.2客户端模块架构**

客户端的架构图大概如下：

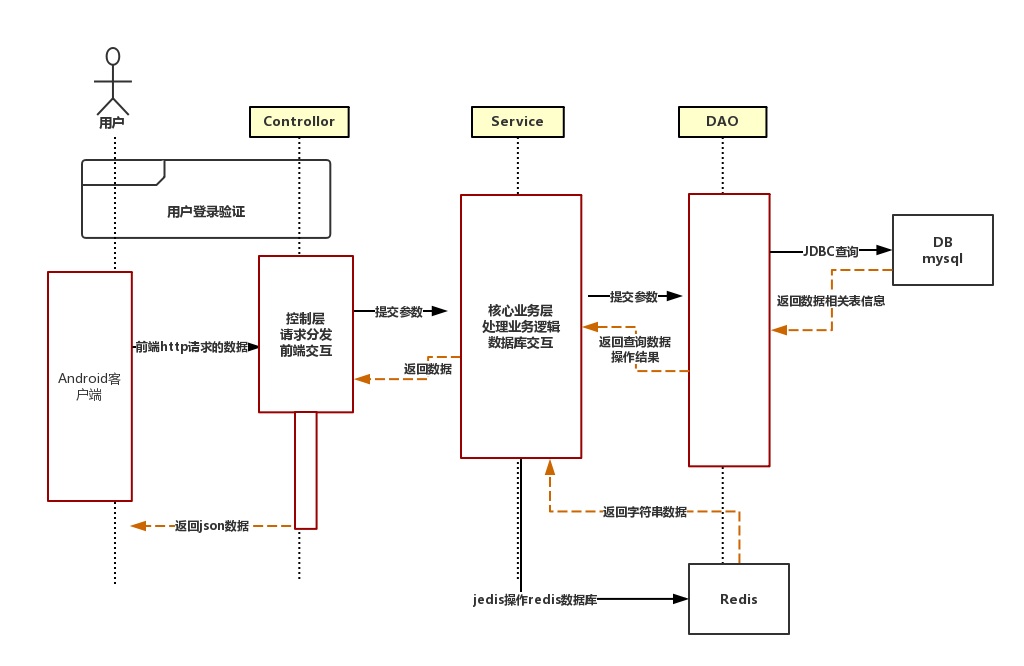


客户端的总体架构较为简单，界面主要通过Activity和Fragment进行数据的动态展示，在开发时，用到一些动态加载数据的组件实现下拉刷新、向上拉去加载更多等动态功能。

在客户端的业务层，主要有三种方式获取数据：工具类生产，本地数据库读取以及向服务端请求。对于敏感度测试等模块，主要通过工具类自动生成测试所需数据；而一些色盲测试、用户数据相关以及知识库等等则需要请求后端进行数据的读取；当然，对于当天的用眼情况统计，主要存储在本地的sqlite数据库，一般通过封装好的一个工具类进行直接数据操作即可。

**4.3服务端模块架构**

系统的后端系统UML图如下



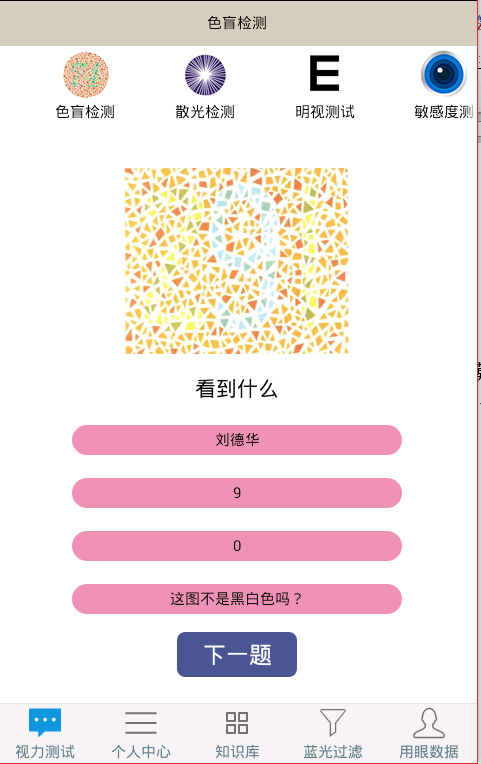
总的来说，后端系统分为三层编程结构：Controller层，service层和DAO层。每层各有作用：Controller层是请求的入口，负责吧网络请求进行分发，是一个请求控制器；service层是系统核心层，主要处理具体的业务逻辑，负责调用数据库持久化框架和数据库进行交互；DAO层是专门用于操作数据库的，它通过mybatis的框架，封装了jdbc的数据库操作动作。

**第五章 模块具体实现分析**

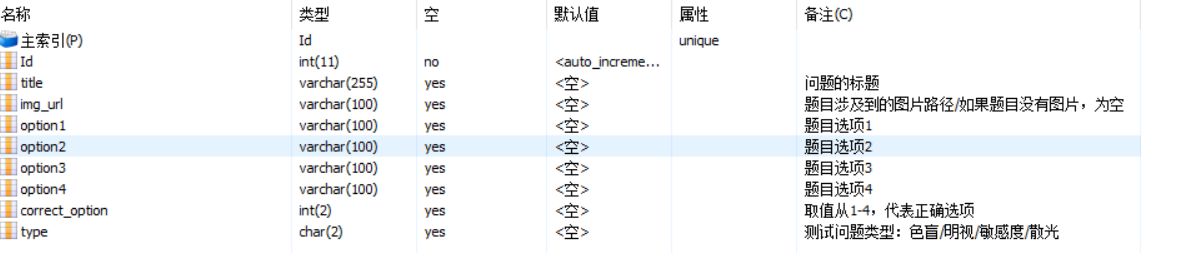
**5.1视力测试模块**

**5.1.1色盲测试模块**

下图是色盲测试模块的截图，其他模块类似：



测试模块中，色盲测试和敏感度测试均是通过读取后端的数据获取测试题目的数据。测试数据的后端表结构如下图：



题目的构成主要有一下部分：题目素材（一般是一个图片素材），题目内容、各个选项以及对应的正确选项，type是代表不同类型的题目。测试相关的数据动态地从后端读取，在打开Android页面的时候，便进行数据初始化了，系统自动一次性地缓存当次测试的所有数据。具体的初始化数据代码大概如下：

ArrayList<TextQuestion> questions;//这是存储问题数据的列表

int nowAnswerQuestion = 0;//存储当前答题的序号

private void init(){

//前面初始化控件过程已经省略

TestService service = new TestService();

//初始化数据

try {

questions = service.getTestQuestions(QUESTION\_NUM, GlobalConst.TEST\_TYPE\_COLORBIND);

} catch (HttpException e) {

}

//初始化第一个答题页面

showNewQuestion(questions.get(nowAnswerQuestion));

}

总的来说，从后端读取数据的过程，其实是调用服务层（Service相关的类）对数据进行读取的，而数据主要以一个ArrayList<TextQuestion>列表来进行存储的。在读取完数据之后，会进行数据的展示，调用的是showNewQuestion()这个方法。在服务端读取数据的核心代码大概如下：public List<TestQuestionVO> getTestQuestions(int num, String type) throws HttpException {

//前面还有一些参数正确性的判断，这里省略掉

List<TestQuestionVO> questions = new ArrayList<>();

Map<String,String> params = new HashMap<>();

params.put("type",type);

params.put("questionsize",num + "");

String result = HttpUtil.synGet(GET\_QUESTION\_PATH,params);

Map<String,Object> resultMap = (Map<String,Object>)JavaBeanUtil.jsonToObj(result);

int status = (Integer) resultMap.get("status");

if(status==10){

throw new UserException(GlobalConst.REMIND\_NOT\_LOGIN);

}

if(status==1){

throw new BackstageException(GlobalConst.REMIND\_BACKSTAGE\_ERROR);

}

List<Map<String,Object>> data = (List<Map<String,Object>>)resultMap.get("data");

for(Map<String,Object> m : data){

questions.add(mapToQuestion(m));

}

return questions;

}

在读取数据中，核心代码是这个：String result = HttpUtil.synGet(GET\_QUESTION\_PATH,params);这段段代码是发起http请求的通用代码，主要是调用HttpUtil这个封装好的工具类进行请求的发起，请求成功之后，会返回一个json字符串（result），猴面再对字符串进行反序列化过程，将之变为一个对象即可得到对应的对象数据。由于具体的http请求代码比较多，这里就省略了。总的来说，色盲测试模块的请求过程可以用如下的图表示：

**5.1.2散光测试模块**

功能实现过程和色盲测试基本一致，略。

**5.1.3明视距离测试模块**

明视距离测试的页面图如下：



在进行明视距离测试时，首先需要用户输入测试眼睛和手机之间的距离，系统会根据用户输入的数字，自动计算需要显式图标的大小。总的数据请求过程和色盲测试部分基本一致，只是视力相关的测试数据无需发起后台请求，直接在本地生成即可。具体的核心代码调用如下：

questions = service.getVisionQuestions(testDistance,VISION\_NUM);总的来说，就是调用对应的查询方法获取数据，下面具体分析下生成用眼数据的相关算法过程，核心代码如下：

public static TestVisionQuestionVO getTestVisionQuestionVO(float distance,float vision){

//省略参数逻辑判断代码

TestVisionQuestionVO question = new TestVisionQuestionVO();

//随机获取方向

int randomInt = (int)(Math.random() \* 4);

char direction = '左';

if(randomInt<1){

direction = '左';

}else if(randomInt<2){

direction = '右';

}else if(randomInt<3){

direction = '上';

}else {

direction = '下';

}

question.setDirection(direction);

//5.0标准分辨大小距离

float normalVisionSize = distance \* EYE\_MIN\_ANGLE; int multiple = (int)((5.0-vision)/0.1);//计算vision值和5.0视力值的倍率关系

float size = (float)(normalVisionSize \* (Math.pow(EYE\_ANGLE\_INCREASE\_RATE,(float)multiple)));//计算对应vision的尺寸

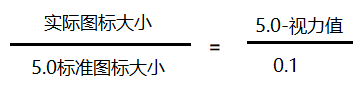
//标准视力表是五分最小视角，所以乘以5，乘以10是因为question //的size返回值是毫米

question.setSize(size \* 10 \*5);

return question;

}

明视距离核心算法是，把距离乘以人的最小分辨角便是最小的可分辨距离，另外，对于视力4.0-5.0这个区间的视力值和图标大小的计算，主要遵循以下的计算原则：在标准视力中，视力之间和图标的大小满足一定的倍率关系，标准视力表5.0视力是5分最小视角，具体公式如下：



而对于不同的距离的5.0最小可分辨距离可以直接如下公式计算：

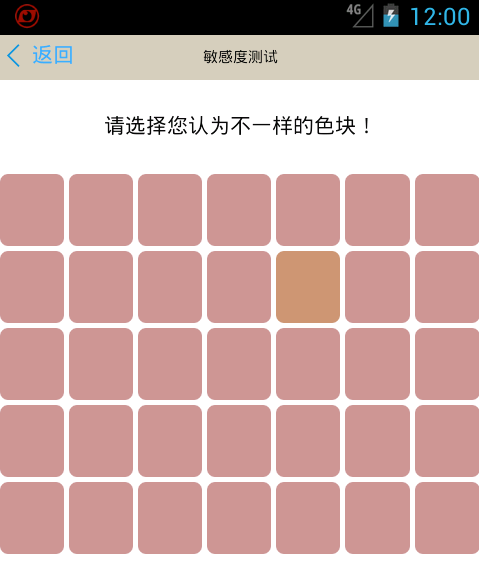


在以上计算中，最小可分辨视角是一个常量，由于国际标准对一标准视力（5.0）的规定是5倍的最小可分辨视角，所以公式需要乘以5。

在实际调整图标大小的时候，还需要结合手机屏幕的尺寸大小，对图片的像素大小进行计算，然后，再按比例进行图像像素的设置，保证图标的实际大小和实际测试的效果相近。

**5.1.4敏感度测试模块**

敏感度测试页面如下：



敏感度测试主要采取的原理是：利用程序动态生成16进制相近的颜色块，用户需要选择颜色不一样的颜色块。在测试的题目中， 难度会依次地递增，颜色块的颜色会越来越接近，具体的核心算法如下：

public static List<TestSensitivityQuestionVO> defaultCreateQuestions(){

int defaultQuestionNum = 10;

int maxDifficulty = 35;

int minDifficulty = 3;

int gradient = (maxDifficulty - minDifficulty) / (defaultQuestionNum - 1);

List<TestSensitivityQuestionVO> questions = new ArrayList<>();

for(int i=0;i<defaultQuestionNum;i++){

questions.add(createQuestion(maxDifficulty));

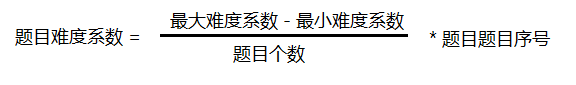
maxDifficulty-=gradient;

}

return questions;

}

在色块生成中，默认问题个数是10，最大难度数为50，最小为10，每个题目的难度系数计算公式如下：



另外，在生成颜色相关数据之后，关于哪个按钮显示正确的颜色，我们也采取简单的随机生成的策略，算法实现不难，代码省略。

**5.2知识库模块**

**5.2.1饮食习惯模块**

饮食习惯界面图如下：



界面的数据查询流程和色盲测试的测试数据获取流程基本一致，这里不详细展开。这个页面主要实现的难点是下拉刷新以及上拉动态加载更多这个过程，下面详细展开这 两部分的实现过程。

总的来说，展示的文章列表利用Android动态展示组件ListView，这个组件支持动态地添加、展示数据。在动态加载数据过程中，主要采用了SwipeRefreshLayout 这个安卓组件。当然，由于需要判断下拉的距离，所以我继承了该类并进行了相应方法的重写过程，下拉距离判断的核心代码如下：

public boolean dispatchTouchEvent(MotionEvent ev) {

switch (ev.getAction()) {

case MotionEvent.ACTION\_DOWN:

// 移动的起点

mDownY = ev.getY();

break;

case MotionEvent.ACTION\_MOVE:

// 移动过程中判断时候能下拉加载更多

if (canLoadMore()) {

// 加载数据

Log.d(TAG, "-------> 加载更多数据");

loadData();

}

break;

case MotionEvent.ACTION\_UP:

// 移动的终点

mUpY = ev.getY();

break;

}

return super.dispatchTouchEvent(ev);

}

上面方法主要获取两个距离：手指按下准备滑动时的初始位置和滑动停止抬起的位置，对应地两个距离会在canLoadMore这个方法中判断能否进行动态加载，canLoadMore核心代码如下：

private boolean canLoadMore() {

// 1. 是上拉状态

float dinstance = mDownY - mUpY;

boolean condition1 = dinstance >= mScaledTouchSlop;

if (condition1) {

Log.d(TAG, "-------> 是上拉状态");

}

// 2. 当前页面可见的item是最后一个条目,一般最后一个条目 //位置需要大于第一页的数据长度

boolean condition2 = false;

if (mListView != null && mListView.getAdapter() != null) {

if (mItemCount > 0) {

int adaptorCount = mListView.getAdapter().getCount();

if (adaptorCount < mItemCount) {

// 第一页未满，禁止下拉

condition2 = false;

}else {

int vPostion = mListView.getLastVisiblePosition();

int cItem = mListView.getAdapter().getCount();

condition2 = (vPostion >= cItem - 1);

}

} else {

// 未设置数据长度，则默认第一页数据不满时也可以上拉

condition2 = mListView.getLastVisiblePosition() == (mListView.getAdapter().getCount() - 1);

}

}

if (condition2) {

Log.d(TAG, "-------> 是最后一个条目");

}

// 3. 正在加载状态

boolean condition3 = !isLoading;

if (condition3) {

Log.d(TAG, "-------> 不是正在加载状态");

}

return condition1 && condition2 && condition3;

}

该方法中，通过判断手指按下和抬起的位置判断用户是否要进行下拉加载更多数据的过程。当然，能否进行加载更多数据，处理用户手指滑动屏幕位置距离外，还有另外两个条件：当前是否是该页的底端以及是否正在加载数据中，前者防止用户下拉数据时也对数据进行加载，后者避免层面加载数据。

总的来说，动态加载的流程图如下：

在实现该模块过程，除了实现下拉加载更多这个功能外，另一个关键功能便是下拉刷新的功能。在下拉刷新这个功能中，借助了Google的Android组件SwipeRefreshLayout组件，核心的接口注册事件代码如下所示：

swipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

new Handler().postDelayed(new Runnable() {

@Override

public void run() {

refresh();

swipeRefreshLayout.setRefreshing(false);

}

}, 0);

}

});

该组件实现下拉刷新是一个异步的过程，主要是新开一个线程任务进行后台数据加载，第二个参数代表是延迟多久加载数据，0代表立即刷新加载数据。在refresh刷新方法中，刷新完成会有一个通知ListView刷新数据过程，refresh核心代码如下：

private void refresh(){

switch (paperType) {

case PAPER\_TYPE\_DEFAULT :

//清空数据

clearPaperList();

//重新初始化

initPapers();

//刷新页面

listViewAdaptor.notifyDataSetChanged();

//回到顶部

papersList.setSelectionAfterHeaderView();

break;

case PAPER\_TYPE\_BLOG :

changePaperType(PAPER\_TYPE\_BLOG);

break;

case PAPER\_TYPE\_LECTURE :

changePaperType(PAPER\_TYPE\_LECTURE);

break;

case PAPER\_TYPE\_EATINGHABIT :

changePaperType(PAPER\_TYPE\_EATINGHABIT);

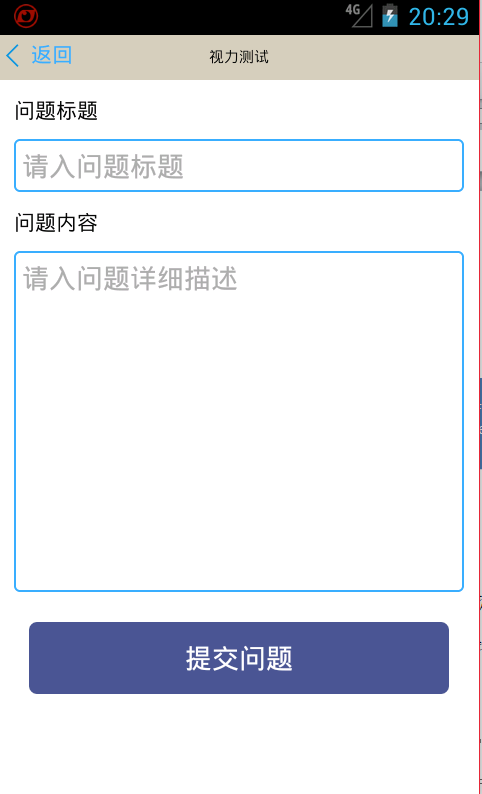
}

}

红色字样是核心代码，它保证了ListView在加载数据完成之后对页面进行刷新操作，把加载完成的数据呈现出来。总的来说，下拉刷新的逻辑流程大概如下：

**5.2.2 提问模块**

用户根据自己的测试结果、用眼疑惑进行提问操作，吧问题提交到云端，相应地会有专家对用户的问题进行解答。主要的截图如下：



**5.3用眼数据**

用眼数据界面效果图如下:



该功能主要实现的是对用户使用手机的数据记录：例如开屏次数、当天使用手机时长和手机待机时长等数据，这些数据主要存储在本地的数据库中，每天进行更新。下面详细介绍该部分功能实现的原理细节。

在实现客户使用手机相关数据统计过程，采用了一个定时器进行定时地计时和监听用户行为，定时器核心代码如下：

//设置定时器，定时刷新数据

if(timerTask==null){

timerTask = new TimerTask() {

@Override

public void run() {

//判断日期是否是今天，防止和昨天重复计算

if(!RegularUtil.dateIsSame(eyedata.getDate(),new Date())){

//如果已经过了一天,更新数据库并重新初始化eyedata

updateEyedata();

}

eyedata.setOpenScreenTimeCountRecent(eyedata.getOpenScreenTimeCo untRecent() + 1);

eyedata.

setOpenScreenTimeCountToday(eyedata.

getOpenScreenTimeCou ntToday() + 1);

handler.post(refreshUI);//刷新界面

if(timerCounter>=AUTO\_SUBMIT\_DATA\_TIME){

//每隔AUTO\_SUBMIT\_DATA\_TIME秒往数据库保存用眼数据， //默认是五分钟

timerCounter=0;

service.updateEyedata(getContext(),eyedata);

}

}

};

}

if(timer==null){

timer = new Timer();

timer.schedule(timerTask,1000,1000);//每隔一秒更新数据

}

位了提高性能，定时器设置每秒运行一次，进行计时，根据手机当前状态是待机休眠还是用户正在使用来进行分情况计时。另外，定时器每隔五分钟会吧记录的数据插入到数据库当中去。具体的定时器工作流程如下：

在监控用户开屏、锁屏行为中，采取了Android本身的一个广播监听组件BroadcastReceiver，当系统有特定的动作例如锁屏、开屏等全局性事件时，该监听组件可以接受这些时间的通知，具体的代码如下：

private class ScreenBroadcastReceiver extends BroadcastReceiver {

private String action = null;

@Override

public void onReceive(Context context, Intent intent) {

action = intent.getAction();

if (Intent.ACTION\_SCREEN\_ON.equals(action)) { // 开屏

mScreenStateListener.onScreenOn();

} else if (Intent.ACTION\_SCREEN\_OFF.equals(action)) { // 锁屏

mScreenStateListener.onScreenOff();

} else if (Intent.ACTION\_USER\_PRESENT.equals(action)) { // 解锁

mScreenStateListener.onUserPresent();

}

}

}

//回调接口

public interface ScreenStateListener {// 返回给调用者屏幕状态信息

public void onScreenOn();

public void onScreenOff();

public void onUserPresent();

}

上述的ScreenBroadcastReceiver是一个自己实现的内部类，重写了onReceiver方法，在接收到广播的时候，根据事件类型进行接口的回调。总的来说，客户端程序只需要实现ScreenStateListener接口即可达到监听锁屏开屏的目的，在对应的方法里面实现锁屏、开屏次数的数据库记录操作即可。

具体的调用核心代码如下：

creenObserver.setScreenStateListener(new ScreenObserver.ScreenStateListener() {

@Override

public void onScreenOn() {

Log.d("开屏事件：","------>开屏");

//更新用眼数据

openScreenRefreshEyedata();

}

@Override

public void onScreenOff() {

Log.d("闭屏事件：","------>关闭屏");

//更新用眼数据

closeScreenRefreshEyedata();

}

@Override

public void onUserPresent() {

//解锁触发的方法

}

});

上述代码在对应的锁屏、开屏回调方法调用了对应的记录方法openScreenRefreshEyedata和closeScreenRefreshEyedata，两个方法会更新数据库、内存中关于开屏、锁屏次数的记录。总的开屏、锁屏统计的实现流程大概如下：

**5.4个人中心、蓝光过滤**

该部分是另外一个成员完成，这里不对其详细实现过程进行展开。