Thss 技术总结

本项目是一个手机应用，用户可以上传自己的食谱，然后获得每顿饭摄取的卡路里。

根据自身的状况，后台的营养专家会给予建议。

客户端

|  |  |
| --- | --- |
| Andorid | 原生应用 + WebView |
| IOS | 原生应用 + WebView |

服务器端

|  |  |  |
| --- | --- | --- |
| Front | 手机WebView使用的静态资源 | Apache |
| API | Front通过Ajax调用API | SpringBoot2 + Postgresql |
| 管理Site | 后台管理系统 | SpringBoot2 + Postgresql |
| 申请Site | 用户申请使用手机应用的网站 | SpringBoot2 + Postgresql |
| Batch | 定时Batch | SpringBoot2 + Postgresql |

Android

1. minSdkVersion targetSdkVersion compileSdkVersion 的区别。

|  |  |
| --- | --- |
| compileSdkVersion | 编译时使用的版本 |
| targetSdkVersion | 最适合运行的版本 |
| minSdkVersion | 能够运行的最低版本 |

一般minSdkVersion < targetSdkVersion = compileSdkVersion

minSdkVersion 和targetSdkVersion 之间的版本如果在某API上有差异

应该在代码中显示的取得运行版本然后区别对待。

1. Fitness API

项目中需要取得指定时间运动的步数。在Android上使用了Google Fit APIs里的History API

官网 ： <https://developers.google.com/fit/android/>

官方提供的Sample代码<https://developers.google.com/fit/android/samples>

在使用Fitness API的时候还必须有下列信息

开发者需要一个Google Account

开发者需要在Google API Console 创建一个项目，并将FitnessAPI设置为有效

并根据Android APP(APK)的签名用的数字证书的指纹信息

和Android项目的package名生成一个OAuth 2.0 Client ID，

这样AndroidAPP就有了访问Fitness API的权限。

上面的内容参照<https://developers.google.com/fit/android/get-api-key>

用户在使用AndroidAPP的时候登陆自己的Google Account

并同意APP去访问Fitness API。

1. Notification

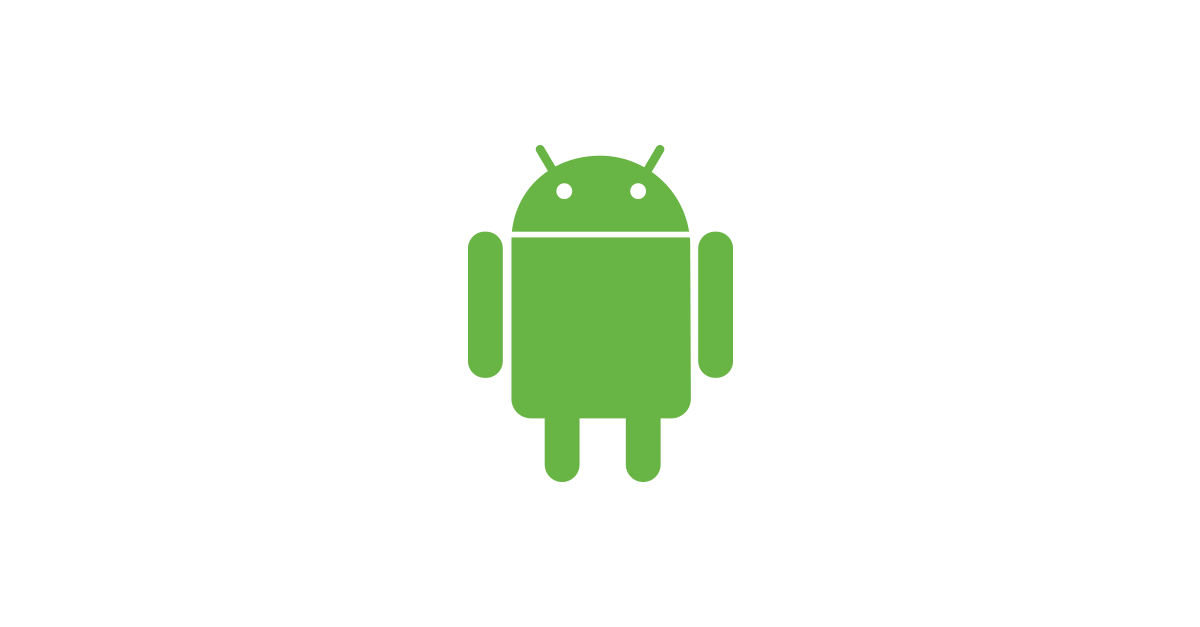
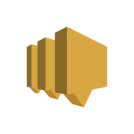
这次Android通过AWS的SNS作为桥梁，利用Firebase完成通知服务。

* 注册DeviceToken

利用FireBase的SDK可以为Android的Device生成一个DeviceToken。

这个DeviceToken会被我们的程序从手机端发送到服务器端。

服务器端会把DeviceToken利用AWS的SDK保存在AWS的SNS中。

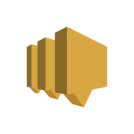
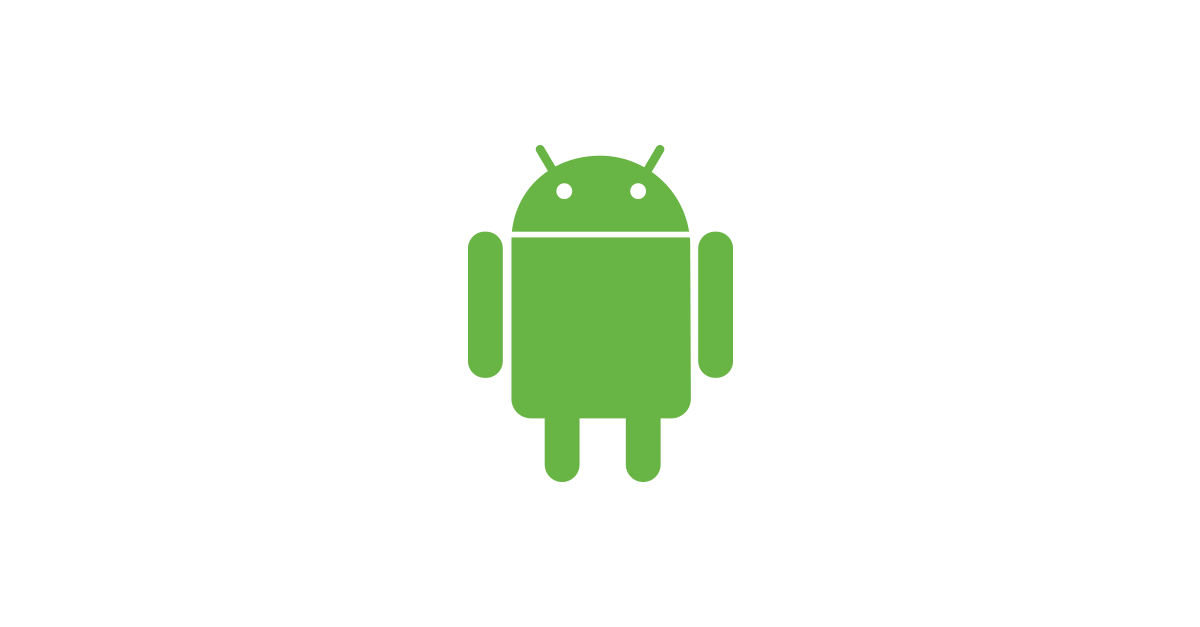
  

* 发送通知

服务器端向SNS发送请求，希望发送信息到某个Device。

SNS联络FireBase传送请求。

FireBase向具体的Device发送请求。

* 接收通知

FireBase发送的消息分两部分，通知部分和数据部分

通知部分notification的Key是固定的，数据部分data的Key是自定义的

比如



|  |  |  |  |
| --- | --- | --- | --- |
|  | 只有通知部分 | 只有数据部分 | 两者都有 |
| 前台 | onMessageReceived | onMessageReceived | onMessageReceived |
| 后台 | 系统任务栏 | onMessageReceived | 通知：系统任务栏 数据：intent 的 extras 参数。 |

注：onMessageReceived是在手机端的一个回调函数。用于开发者自己实装完成接受到任务的处理

当APP在前台运行的时候，不管收到的信息是只有通知部分，还是只有数据部分，还是两者都有，都会走到回调函数。

当App在后台运行（包括关闭）的时候，收到的信息中如果只有通知部分会显示在消息提示栏里，只有数据部分会走回调函数，两者都有则通知部分显示在消息提示栏，数据部分会以参数形式传递给App的启动页面。

这次项目的通知信息里是通知部分和数据部分都包含的。

所以我们对onMessageReceived做了处理，保证不管是前台还是后台。

都按照后台的处理方式

所以在onMessageReceived里，手动把通知消息追加到了消息提示栏里，

并增加了消息提示栏的点击事件，使得点击消息栏时打开启动页面，传递数据信息。

这次使用SNS，主要是为了让服务器在发送通知的时候，

不管是IOS还是Android能使用同样的API，

比如使用同样的AWS SDK完成发送通知服务。

在SNS里配置IOS和Android对应的通知服务。

SNS里还可以配置比如Windows Phone，百度，Amazon Device的通知服务。

1. 欧姆龙体重计血压计连携

欧姆龙外部设备连携基本上按照欧姆龙提供的内容配置和使用。

* 需要在手机上安装欧姆龙的App，并准备好外部设备。

当外部设备获取数据后通过蓝牙传递给欧姆龙的App。

* 需要欧姆龙提供的SDK，以及提供调用的Sample和文档
* 需要向欧姆龙提供一个登陆申请。包含了App的package，证明书Hash等信息。

欧姆龙会给我们一个APP ID。

* 利用APPID，就可以根据Sample来读取数据
* 我们的App使用欧姆龙的话，欧姆龙的App也需要把我们APP的信息加入并升级版本

服务器端

1. 框架

本系统采用SpringBoot2.0

* 在基本上不需要配置的情况下就可以运行简单的WebApp。

其他根据不同的需要可以在ClassPath的application.properties里增加配置信息

* 可以使用外部Tomcat启动war文件。
* 也可以使用java命令直接启动jar文件。springboot内嵌了Tomcat，这个在对Tomcat版本没有特别需求的时候，在服务器上发布启动脚本都比较简单。
* 文档资料比较完备，基本上遇到的问题在网上都可以搜索到。
* SpringBoot也包含Profile。可以为不同的环境配置不同的配置信息。然后在启动时指定profile选择不同的配置文件。

在使用JAR和WAR方式启动的时候，对于某些功能并不完全一致。

我们的测试环境使用War方式启动，而客户的验收环境使用Jar方式启动。

所以导致了3到4个Bug。以后要注意在使用SpringBoot的时候开发环境和客户环境应该使用同样的方式启动

1. 定时batch

定时Batch使用了SpringBoot的Schedule功能。

简单Sample, 每10秒执行一次

SpringBootBatchApplication.java

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication

@EnableScheduling

**public** **class** SpringBootBatchApplication {

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

SpringApplication.*run*(SpringBootBatchApplication.**class**, args);

}

}

application.properties

cron.myTask=\*/10 \* \* \* \* \*

MyTask.java

**import** org.springframework.scheduling.annotation.Scheduled;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** MyTask {

@Scheduled(cron = "${cron.myTask}")

**public** **void** runTask() {

System.***out***.println("Hello Task");

}

}

因为这次客户对Batch的要求比较简单，所以在设计上没有考虑更多的情况，

只是使用了Spring 的最常规的配置。一些其他需求列在最后的课题里，做后续调查。

1. Http缓存

FrontEnd都是静态资源，所以牵扯到了缓存处理。

如果不使用缓存，将会增加网络通信影响客户端表示速度。

如果使用缓存，新的变更将无法再客户端及时反映。

本项目由于事先客户要求资源变化在一个小时内反映。

Apache会自动再响应头部加上ETAG

然后手动再Apache配置里增加Cache-Control: private, max-age=3600

判断缓存有没有过期

Yes No

使用缓存

发送请求判断资源有没有变化

Yes No

使用缓存，延长缓存时间

请求新资源

以后如果其他项目客户的要求有变的话。可以根据如下策略设置缓存



参考：

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=zh-cn>

1. 事务控制

前提：本次使用的是Spring，Mybatis，Postgresql

规定：

* 事务控制放在Service层
* Controller中一个处理只能调用一个外层Service。

其他的共通Service只能在这个外层Service中调用。

* 外层Service中的Public方法上加

@Transactional(propagation = Propagation.REQUIRED)

如果外层Service中的方法没有更新操作则加上

@Transaction(propagation = Propagation.REQUIRED , readOnly = true)

共通Service中Public的方法上加

@Transactional(propagation = Propagation.MANDATORY)

如果共通Service中的方法没有更新操作则加上

@Transactional(propagation = Propagation.MANDATORY, readOnly = true)

其他有DB操作的组件的设置参照共通Service

|  |  |  |
| --- | --- | --- |
| 外层Service | 共通Service |  |
| @Transactional(propagation = Propagation.REQUIRED) | @Transactional(propagation = Propagation.MANDATORY) | OK |
| @Transactional(propagation = Propagation.REQUIRED) | @Transactional(propagation = Propagation.MANDATORY, readOnly = true) | OK |
| @Transactional(propagation = Propagation.REQUIRED, readOnly = true) | @Transactional(propagation = Propagation.MANDATORY) | Exception |
| @Transactional(propagation = Propagation.REQUIRED, readOnly = true) | @Transactional(propagation = Propagation.MANDATORY, readOnly = true) | OK |
| 忘记加@Transactional | @Transactional(propagation = Propagation.MANDATORY)  @Transactional(propagation = Propagation.MANDATORY, readOnly = true) | Exception |
| Controller里直接调用共通Service | @Transactional(propagation = Propagation.MANDATORY)  @Transactional(propagation = Propagation.MANDATORY, readOnly = true) | Exception |

这样设置的目的

1. 一次处理在一个事务中
2. 防止本应无更新处理的方法中调用了DB更新处理。
3. 防止调用共通Service的外层Service没加@Transactional
4. 防止共通Service被在Controller里直接调用。

上面的规定需要列入Review的CheckList

在不同的DB或者ORM情况下，上述的规定有可能不适用，这个要在开发前做验证。

1. 用户密码加密

用户密码加密采用不可逆的方式

第一次加密用Bcrypt加随机盐将密码明文进行哈希。

第二次加密用SHA-256将第一次生成后的哈希值再进行哈希。

加密后的密码和盐保存在DB中。

在验证密码的时候使用同样的加密逻辑然后和DB中的值进行比较。

比较的时候使用SlowEquals。保证每次验证的时间一致。

不过现在有一种新的password哈希的工具。

Argon2

Java版本在这里<https://github.com/phxql/argon2-jvm>

后面的项目可以使用。

1. JWT加密

Login完了以后，手机用户访问API通过AccessToken访问API。

JWT的做成和验证使用jose4j工具做成。

Sample：<https://bitbucket.org/b_c/jose4j/wiki/JWT%20Examples>

然后我们用AES\_128对JWT再做了一次可逆的加密，该AccessToken保存在手机客户端。

1. 数据加密

客户数据库中某些敏感字段的信息需要做加密处理。

为了能使用LIKE等语句。所以使用了数据库端的加密处理。

使用的是PostgreSql 的pgcrypto模块的 encrypt和dencrypt方法。

encrypt(convert\_to('用户名','UTF8'), 'pass'::bytea, 'aes')

convert\_from(decrypt(name, 'pass'::bytea, 'aes'), 'UTF8')

1. SQL Log

本系统要求服务器端要出SQL Log，但是如果里面有敏感信息的话不能显示。

使用了Mybatis的Intercepts然后根据参数的名称选择是否需要屏蔽。

其他部分按照SQL的形式出Log。

但是其实使用在参数上加上Annotation应该更好。

并且之前应该对Mybatis传入的参数类型规范化，这样在Intercepts里比较容易处理。

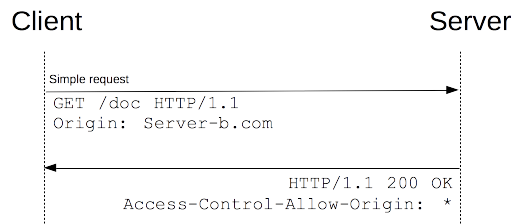
1. 跨域访问

当frontend和backend部署在不同域，端口，或者协议上面的时候。

Frontend通过ajax访问backend的时候会产生跨域访问。

在跨域访问中有两种类型的Request

一种称为simple request。一种称为preflight request。



SimpleRequest的场合，浏览器会直接发送请求，并在请求中加入Origin.

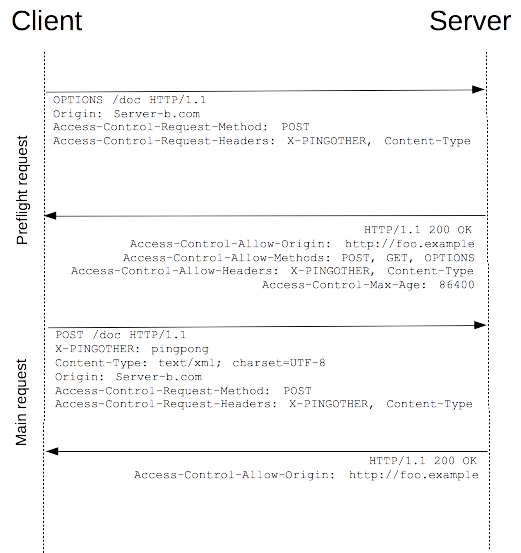
服务器端会做检查是否允许该Origin访问本服务器。

如果允许 返回200，并在响应header中加入Access-Control-Allow-Origin

以及允许的Origin范围

如果不允许 返回200，在响应header里无Access-Control-Allow-Origin信息。

浏览器判断Access-Control-Allow-Origin来显示是否表示错误。



preflightRequest的场合，会先发送一个Method为OPTIONS的Http请求，

并在请求中加入Origin，Access-Control-Request-Method，Access-Control-Request-Headers

服务器端会做检查是否允许该Origin访问本服务器。

如果允许 返回200，并在响应header中加入

Access-Control-Allow-Origin

Access-Control-Allow-Methods

Access-Control-Allow-Headers

浏览器将发送真正的Request到服务器。

如果不允许 返回403，在响应header里无上诉信息。

在SpringBoot里需要加上配置来配置允许跨域访问的范围

@Configuration

**public** **class** CorsConfiguration {

@Bean

**public** FilterRegistrationBean<CorsFilter> corsFilter() {

UrlBasedCorsConfigurationSource source = **new** UrlBasedCorsConfigurationSource();

org.springframework.web.cors.CorsConfiguration config = **new** org.springframework.web.cors.CorsConfiguration();

config.setAllowCredentials(**true**);

config.addAllowedOrigin("xxx");

config.addAllowedHeader("xxx");

config.addAllowedMethod(HttpMethod.***GET***.name());

config.addAllowedMethod(HttpMethod.***POST***.name());

config.addAllowedMethod(HttpMethod.***PUT***.name());

config.addAllowedMethod(HttpMethod.***DELETE***.name());

config.addAllowedMethod(HttpMethod.***OPTIONS***.name());

source.registerCorsConfiguration("/\*\*", config);

FilterRegistrationBean<CorsFilter> bean = **new** FilterRegistrationBean<CorsFilter>(**new** CorsFilter(source));

bean.setOrder(0);

**return** bean;

}

}

版本管理

本次代码的版本管理使用公司的GitLab。

master 从Master做本番版本。

develop 开发用的分支。最初从Master分支切出。

feature 所有的开发按机能分建立自己的feature，下两次release的机能必须建分支，等到下一次release完了以后才能merge会develop

注：项目开始时候由于对项目文件的规划上不够清晰，出现很多人修改一个文件的情况，比如一个Module里的多个画面对应一个Controller文件，对应一个Service文件。结果导致了很多人同时修改一个文件，而一般都是在做完一个机能后才会Merge，所以造成了大量的冲突Merge工作。

release 准备release的版本,测试版本

release从develop切出，作为测试版本。

测试出的Bug，直接在release上修改。Bug测试完成后Merge回develop

等完成测试后准备本番前Merge进master版本。

hotfix 本番的紧急Bug对应

hotfix从master切出。 merge回develop和master

本番出了紧急Bug对应，需要从Master切出分支。然后对应，在此分支上做测试。

测试成功后merge到develop和master，然后从master做本番版本。

参考：<https://nvie.com/posts/a-successful-git-branching-model/>

在开发过程出现冲突的主要原因是两个人同时改了一个文件。

从根本上回避这个问题的办法是把业务代码分为单个机能相关代码和共通代码。

单个机能相关代码要尽可能只有一个人去开发。共通代码由专门的人来维护上传至Develop。

每次develop代码发生变化后，通知各发开人员Merge到自己的Feature分支。这样基本不会出现冲突问题。这样可以充分利用feature分支，使得测试的版本更加清晰。

Infrastructure

本系统服务器端利用了AWS云服务平台。

南京测试环境使用了Elastic BeanStalk.

使用Elastic BeanStalk可以快速配置服务器，负载均衡，DB，网络，AutoScaling，服务器版本升级。可以快速发布WebApplication，并提供发布版本管理，监控，日志。

所有的基本操作都只需要在控制台完成。

我们还可以通过在Application中的配置文件Ebextension来改变Elastic BeanStalk的配置。

配置内容很多，要查阅相关文档，这一点相对于直接修改EC2上的文件要复杂一些。

也基于此原因，以及客户将来还要在服务器上自己安装一些软件。

所以在验收环境，本番环境换成了手动配置Infra。

关于本番运行环境还在整理中。

部署

本项目初期

对ElasticBeanstalk的发布，使用本地取代码，编译代码，上传war文件，

在AWS控制台上点击发布

对 EC2的发布同样使用本地取代码，编译代码，上传jar文件，连上EC2执行脚本。

本项目中期

更改为使用Jenkins，并利用Jenkins的Aws plugin对ElasticBeanstalk及EC2做一键发布。

课题

本系统开发周期比较短，涉及的内容却比较多，所以在开发过程中的遇到的一些问题没有充分的时间来调查出根本原因，而是采用了一些workaround，还有一些内容来不及总结报告里详细描述，这里把这些问题记录下来，后面继续调查。

1. 在frontend和backend的时候如何能减少使用JavaScript动态改变画面layout

或者说有什么更好的方式。

1. 为什么手机WebView在用JQuery异步发送多个Ajax Request的时候，有些画面出现某些Request并没有发出而是被终止了。
2. SpringBoot和Mybatis如何使用Optional 来解决NullPointException的问题。
3. SpringBoot和Mybatis如何处理日期型数据的处理和传递。
4. 在frontend和backend的认证方式的实现的最佳实践。包含使用和不使用第三方托管。
5. Spring Boot的单体测试。
6. Spring Boot在结合BeanValidation时候的通用设置。
7. Spring Boot的异常处理，包括单纯验证处理，业务异常，系统异常。
8. SpringBoot在处理日志文件时的通用设置
9. SpringBoot在处理Restful Webservice时的通用设置
10. Mybatis如何更好的处理SqlLog中回避敏感信息。
11. 使用Spring的Schedualer功能处理定时Batch的时候如何做到

* 如何手动触发Task
* 如何动态暂停Task的scheduler
* 如何动态改变Task的scheduler

1. 如何使用Git的Branch对开发及维护阶段做好版本管理。
2. 如何使用Jenkins对AWS做持续部署。
3. 使用Jar方式和War方式启动SpringBoot App的时候会有哪些细微的差异。

在本项目遇到的有如下几个。

* 上传文件的临时目录不一样
* ClassPath下的resouce文件读取时不一样
* Thymeleaf在使用include fragment的时候不一样。

1. 本系统处理PDF的时候使用的是Apache PDFBox2，由于前期调查不够充分

结果在PDF出动态数据的时候非常麻烦。要动态去计算坐标。好在PDF机能只有一个。

后面对PDFBox2的使用要加以更充分的调查。

1. 关于AWS的服务及内部概念的详细调查。

VPC，Subnet，SecurityGroup，GateWay，Route53，S3，LoaderBanlancer，DB的备份和restore，

Codedeloy等。

1. Android和IOS的拍照，上传图片，图片压缩，展示，保存。
2. Android和IOS的通知服务，APP版本升级。
3. 手机APP的安装测试，HockeyApp的使用。
4. 一般的防止攻击的开发时的对策。