Report

basic part:

system requirement analysis and definition

系统需求分析

价值点:

1. 以游戏等娱乐作品为契机,开始有兴趣了解更多相关的知识,是很多年轻人获取知识的方式之一,也可能是非常好的方式之一。
2. fgo 这款游戏受众较广,玩家超过一千万,足够引起重视,玩家对游戏或者说 fate 系列的忠诚度极高,因为爱而去了解相关英灵故事,最终成为某方面知识的专家的情况并不罕见,介绍英灵背景故事的视频也不在少数,举个例子,在 fate 出现之前,像吉尔伽美什史诗这种古老难懂,情节一点都不生动有趣的书,大多数人连听都没听说过,但现在,很多人都因为 fate 里面的角色吉尔伽美什去把这本书买来看,虽然因为这本书阅读难度较大,真正看完的人可能不多,但至少说明有这方面的需求,即使是这么冷门的东西,人们也会有兴趣去看。
3. 暂时没有满足这方面需求的产品出现。

不可替代性

现有的了解英灵的途径大概有三条, fgo wiki , 各种百科,以及专业的书籍各种百科,以及专业的书籍

- fgo wiki : 主要是面向玩家提供全面的信息,对玩游戏很有帮助,但如果想要进一步了解某个英灵难度较大。
- 各种百科: 基本所有相关词条,但因为这些百科太全面了,会出现很多不相关的东西,想要找到自己想找的那个词条可能不是那么容易(比如说搜索亚历山大,你需要在浩如烟海的亚历山大中努力找到那个建立亚历山大帝国的那个亚历山大)很多玩家只是一时兴起,这样的查找体验不好,而且很容易让三分钟热度过去,找到了,不想看了(就像你晚上想吃泡面,虽然你可以去商业中心,那里什么都有,你可以买到各种美味的泡面,但我猜你还是会选楼下的便利店,因为方便省事)我们这里只需一键,就能够抵达你想要了解的那个从者那里,体验超棒,尽可能赶在三分钟热度结束前,让你得到进一步探索的可能。
- 专业的书籍:进一步了解某个人物的首选,但对新人而言,如何选书,是一个非常困难的问题,关于哪方面的,选哪一本,都是问题。我们这里提供全套解决方案,从各个角度推荐图书,不只是局限于个人,还有关于那个时代的各种你可能需要的相关书籍供你选择。

系统需求定义:

这是一个面向 fate 爱好者的全方位的知识解决方案提供平台,整合优化了各个其他知识获取途径,让用户以最简单快乐的方式了解到最全面最优质的知识。

data requirement analysis and definition

1.英灵基本数据

需要收集游戏中关于英灵信息的基本数据,包括身高、体重、性别、职阶、游戏中的故事、图片等等,尽可能全面而完整的关于英灵的数据。

2.抽象数据

从游戏中的英灵中抽象出来的数据,包括人物原型(prototype), 地域(region), 来源(origin),这些数据作为游戏和进一步探索知识的桥梁,起到中介作用。

3.真实世界的知识数据

包括图书信息(book),介绍文章(article),百科链接(pedia),存储时与抽象数据建立关系,方便下一步查找。

data model design

我们使用关系数据库进行设计,使用postgresql进行数据库搭建，将上述三种数据分开存储,并通过关系表连接 英灵基本数据--抽象数据--真实世界的知识数据

techniques in database

级联删除

我们的数据分别存在很多个表中,各个表里面的数据通过多对多的关系表连接,如果删除一条数据的时候会因为多对多关系表的外键约束而变得非常复杂,所以我们使用级联删除的方法,定义外键时让外键在对应主键删除或更新时强制删除或更新。

视图

我们的数据库的核心功能是搜索,更新的频率很低,存入的数据基本不会变,增加的数据也非常少,所以我们为用户创建了一个视图,便于搜索,不需要使用过多的join,大大提高了开发效率。

sql语句变量分离

> The objects exposed by the `psycopg2.sql` module allow generating SQL statements on the fly, separating clearly the variable parts of the statement from the query parameters:

```
from psycopg2 import sql

cur.execute(
    sql.SQL("insert into {} values (%s, %s)")
        .format(sql.Identifier('my_table')),
    [10, 20])
```

我们使用了psgcopg2文档里面的sql语句变量分离的方法，消除了字符串拼接方法中部分符号需要引用的风险，让程序更加安全

database development

1.数据收集

根据以上的定义与设计进行相应的数据收集，我们使用了python的beautifulSoup库把数据拆分并存储在csv文件中

2.数据库构建

根据所收集到的数据类型对数据库设计进行修整，然后将csv文件导入，用postgresql的ddl和 sql语言实现数据库的搭建与数据的填充

3.数据再收集

根据数据库需求，再次针对性收集所需数据填入数据库，使得数据库内数据完整

can solve the tasks presented in mid-term

我们期中展示的功能如下,全部能够实现。

1.特征搜索功能

对应应用中的高级搜索。

2.深度搜索功能

对应应用总的servent界面以及explorer界面。

3.关联搜索

类似特征搜索,我们复用了同样的存在方式,对应explorer界面的第一部分。

4.增添英灵,,文章等等

我们使用python语言实现的,需要管理员操纵命令行,通过填写特定的csv文件来实现。

5.删除英灵

因为数据库中大量数据都是依靠英灵存在的,如果随意删除很可能导致数据库出问题,所以我们只允许删除角色,让程序自己决定该删除哪些相关信息。

6.更新数据

同样是使用python语言实现的,需要填写对应的csv文件,但允许更改任何一处数据。

reliable and security

1.数据可靠性

收集数据时,我直接爬取的fgo wiki的数据,与官方数据一致,可以保证数据的可靠性。真实世界的数据收集我们采取的是人工收集筛选的方式,尽可能做到可靠。

2,数据库安全性

我们所有对数据库的操作都是在服务器上完成的,传输通讯时,只是在相互之间传递json文件,杜绝了程序功能以外的非法操作,再加上核心功能是查询,大大减少了数据库被恶意攻击篡改的可能性。

主程序之外为管理员提供的python代码可以对数据库进行增删改,但也严格限制了管理员的行为,

首先,管理员需要输入数据库的账户和密码才可以进行操作,每次操作结束之后都需要重新输入,不会记忆,因为我们的数据库增删改的操作较少,所以这样权衡下来比较划算。 其次管理员所有的操作,插入\更新\删除都要严格按照规定填写对应的csv文件,提升了操作的规范性,更不容易出错。 最后,我们对管理员可以做出的修改也做出了限制,比如最危险的删除,我们规定管理员只能对英灵进行删除操作,并且一次只能删除一个英灵,删除时还需要连续四次输入同一个英灵的序号才可以成功删除.虽然操作繁琐,但基于我们的程序极少进行删除操作,我们认为这样的设计是比较合理的。

Document

前端(Android)

MainActivity

安卓应用的主界面:下面有三个界面,依次是FragmentTop, FragmentNameSearch以FragmentCustomSearch。 第一个是欢迎界面,第二个是按姓名搜索界面,第三个是按特征搜索界面。 姓名搜索界面和特征搜索界面在选定数据点击搜索后会向SearchResult发送相关数据,并跳转到SearchResult界面。

SearchResult

搜索结果界面,将搜索信息处理并发送给服务器,获得一串匹配的英灵信息。点击一次召唤按钮会出现十个搜索结果,不断点击会不断加载,直到匹配的所有英灵都显示出来,提示无法找到更多,用户点击英灵头像,程序会将相应英灵的ID传到ServentActivity并跳转过去。

ServentActivity

接受SearchResult传来的ID,并向数据库发送请求,获得英灵的相关数据,用服务器提供的数据填充界面信息,页面最底部有一个explore按钮,点击之后跳转到ExploreActivity界面,并传递英灵ID。

ExploreActivity

接受ServentActivity传来的ID,并向数据库发送请求,获得英灵explore界面的相关数据并填充界面,Explore界面由四部分构成。

1.相关搜索

复用了特征搜索的功能,用户点击相应的按钮(原型\地域\来源)之后会跳转到SearchResult界面,进行只有一个参数限制的特征搜索。

2.百科

由很多按钮组成,每个按钮代表一个百科的链接,点击按钮之后会跳转到Webpage界面,并传递参数url。

3.文章

展示出所有相关的文章,每个文章限定了最大长度,如果想要看全文,可以点击相应窗口,会跳转到ArticleActivity界面,并传递文章信息。

4.图书

展示出所有相关的书籍,因为没有版权,所有我们只提供了书名,作者以及ISBN书号方便用户自行查找。

Webpage

用于展示网页信息,接收url并打开。

ArticleActivity

用于展示文章,接受文章信息,并展示出来。

前端(微信小程序)

Index(.js ,.wxml,.wxss)

小程序的主界面:下面有三个界面,依次是首页,英灵检索以及特征检索。 第一个是欢迎界面,第二个是按姓名搜索界面,第三个是按特征搜索界面。 点击就可以进行跳转

sort1(.js ,.wxml,.wxss)

姓名搜索界面 在搜索栏里输入名字,来搜索相应的英灵 会向服务器发送请求,并跳转到搜索结果页面

sort2(.js ,.wxml,.wxss)

特征搜索界面 选择相应的特征,来搜索相应的英灵 会向服务器发送请求,并跳转到搜索结果页面

list(.js ,.wxml,.wxss)

搜索结果界面,将搜索信息处理并发送给服务器,获得一串匹配的英灵信息。点击load more按钮会出现十个搜索结果,不断点击会不断加载,直到匹配的所有英灵都显示出来 显示 no more 用户点击英灵头像,程序会将相应英灵的ID传到英灵介绍界面并跳转过去。

post(.js ,.wxml,.wxss)

接受传来的ID,并向数据库发送请求,获得英灵的相关数据,用服务器提供的数据填充界面信息,页面最底部有一个explore按钮,点击之后跳转到Explore界面,并传递英灵ID。

explorer(.js ,.wxml,.wxss)

接受post界面传来的ID,并向数据库发送请求,获得英灵explore界面的相关数据并填充界面,Explore界面由三部分构成。

1.相关搜索

复用了特征搜索的功能,用户点击相应的按钮(原型\地域\来源)之后会跳转到搜索结果界面,进行只有一个参数限制的特征搜索。

2.文章

点击可以跳转到 article_list 界面 显示与该英灵相关的文章

3.图书

展示出所有相关的书籍,因为没有版权,所有我们只提供了书名,作者以及ISBN书号方便用户自行查找。

article_list(.js ,.wxml,.wxss)

用于展示文章基本信息

article(.js ,.wxml,.wxss)

用于展示完整的文章

后端

前后端接口

姓名搜索

简介：根据英灵姓名获得该英灵的id,servent_name,servent_profile_pic属性。

接口网址：http://dararara.xyz/name_search

前端给后端json文件格式：

{name:英灵姓名}

后端给前端json文件格式：

[id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

...

]

特征搜索

简介：根据英灵起源，地域，类型，结盟，体重范围，身高范围。获得该英灵的id,servent_name,servent_profile_pic属性。

接口网址：http://dararara.xyz/characteristics_search

前端给后端json文件格式：

{ “origin”:“起源” “region”:“地域” “class”:“类型” “alignment”:“标签” “weight_down”:“体重下限” “weight_up”:“体重上限” “height_down ”:“身高下限” “height_up”:“身高上限” } 如果为空，就用”null”字符串代替。 “下限”,“上限”如果为空，用”-1”代替。

后端给前端json文件格式：

[id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

...

]

特定所有属性获取

简介：获取当前数据库中已存在的所有地域，起源，结盟，类型信息。

接口网址：http://dararara.xyz/get_attribute

前端给后端发送get请求。

后端给前端json文件格式：

{ region:[XXX,XXXX,XXX,...] origin:[XXX,XXXX,XXX,...] alignment:[XXX,XXXX,XXX,...] servent_class:[XXX,XXXX,XXX,...]

}

英灵

简介：前端给后端传递英灵id,后端返回前端该英灵的所有属性。

接口网址：http://dararara.xyz/servent_infomation

前端给后端json文件格式：

{servent_id:XXXXXX}

后端给前端json文件格式：

{ servent_id: XXXXXX servent_name: XXXXXX servent_name_japanese: XXXXXXX servent_name_english: XXXXXX height: XXX weight: XXX gender: XXX strength: XXX endurance: XXX agility: XXX mana: XXX luck: XXX noble_phantasm: XXX craft_name: XXX craft_description: XXX craft_src: XXX alignment:XXX class: XXX illustrator: XXX voice_actor: XXX region: XXX origin: XXX prototype: XXX full_picture: XXX bond_text: [XXX,XXX,XXX]（可能多出处） }

探索

简介：前端给后端传递英灵id,后端返回前端该英灵地域，原型，起源，还有相关的百科，文章，书籍信息。

接口网址：http://dararara.xyz/explorer_infomation

前端给后端json文件格式：

{servent_id:XXXXXX}

后端给前端json文件格式：

{ prototype: XXXXXX region: XXXXXX origin: XXXXXXX pedias: [{pedia_id:XXXXX,pedia_name:XXXXX,pedia_url:XXXXXX},....]

articles: [{article_id:XXXXX,article_title:XXXXX,article_content:XXXXXX,author:XXXXXX},....]

books: [{book_title:XXXXX,isbn_code:XXXXX,book_writer:XXXXXX},....]

}

原型搜索

简介：前端给后端原型，后端返回与该原型有关的所有英灵id,姓名，图片网址。

接口网址：http://dararara.xyz/prototype_search

前端给后端json文件格式：

{prototype: XXXXXX}

后端给前端json文件格式：

[id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

...

]

地域搜索

简介：前端给后端地域，后端返回与该地域有关的所有英灵id,姓名，图片网址。

接口网址：http://dararara.xyz/region_search

前端给后端json文件格式：

{region: XXXXXX}

后端给前端json文件格式：

[id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

...

]

起源搜索

简介：前端给后端起源，后端返回与该起源有关的所有英灵id,姓名，图片网址。

接口网址：http://dararara.xyz/origin_search

前端给后端json文件格式：

{origin: XXXXXX}

后端给前端json文件格式：

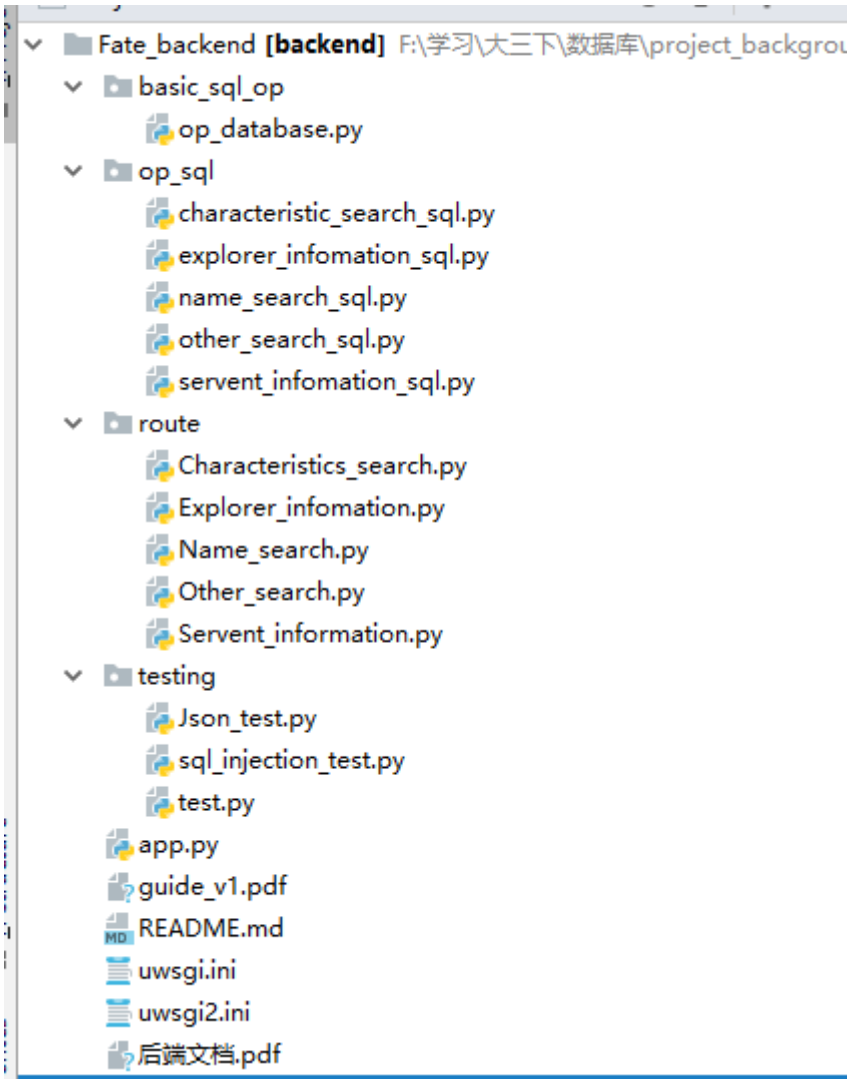
[id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

id: xxx servent_name: '英灵姓名1', 'servent_profile_pic': 网址},

...

]

代码结构



包

basic_sql_op: 其中包含了数据库的基本操作。

```
'''
用于连接数据库，创建游标
'''
def connect(self)
```

```
'''
用于关闭数据库连接
'''
def close(self)
```

```
'''
用于查询数据库中的相关信息
input: sql语句，参数
output: 查询结果
'''
def select(self,sql,para)
```

```
'''
用于增加，删除，修改数据库中的相关信息
input: sql语句，参数
output: 修改结果成功True或失败False
'''
def iur(self,sql,para):
```

op_sql包中保存到了和数据库交互到的函数。

```
'''
按特征查询相关英灵
input: 英灵起源，地域，类型，结盟，体重范围，身高范围
output: 按照json格式返回获得该英灵的id,servent_name,servent_profile_pic属性。
'''
def characteristic_search(info,weight,height)
```

```
'''
特定所有属性获取
output: 按照json格式返回数据库中已存在的所有地域，起源，结盟，类型信息。
'''
def get_attribute()
```

```
'''
英灵探索功能
input:英灵id,
output:按照json格式返回返回英灵地域，原型，起源，还有相关的百科，文章，书籍信息。
'''
def explorer_infomation(id)
```

```
'''
按姓名查询相关英灵
input: 英灵姓名
output: 按照json格式返回英灵的id,servent_name,servent_profile_pic属性。
'''
def name_search(name):
```

```
'''
按源类型查询相关英灵
input: 原型
output: 按照json格式返回与该原型有关的所有英灵id,姓名，图片网址。
'''
def prototype_search(prototype)
```

```
'''
按地域名查询相关英灵
input: 地域名
output: 按照json格式返回与该地域名有关的所有英灵id,姓名，图片网址。
'''
def region_search(region)
```

```
'''
按起源查询相关英灵
input: 起源
output: 按照json格式返回与该起源有关的所有英灵id,姓名，图片网址。
'''
def origin_search(origin)
```

```
'''
获取英灵信息
input: 英灵id
output: 按照json格式返回servent_id, servent_name, servent_name_japanese,
servent_name_english, height,
weight, gender, strength, endurance, agility, mana, luck, noble_phantasm, craft_name,
craft_description,
craft_src, alignment, class, illustrator, voice_actor, region, origin, prototype,
full_picture, bond_text
'''
def servent_infomation(id)
```

route包中保存着前后端交互的接口，上文已经提及。

testing包中保存着一些测试代码。

管理员端(python命令行)

insert_servent.py

向数据库中插入一个英灵,插入的数据模板详见insert.csv,需要插入时需严格按照模板填写。

update.py

更新某个特定表中的某个特定列的数据，需要设定一个参数进行过滤筛选，更新的数据模板参考update.csv。

upload.py

上传文章或书籍，运行脚本时会有选择的提示，根据上传的类型不同，文章参考article.csv，书籍参考book.csv，填写相应的数据。

delete_servent.py

删除某个英灵，需要谨慎查找需要删除英灵的序号，然后根据序号重复输入三次，然后删除成功。