



北京交通大学

Beijing Jiaotong University

基于华为鲲鹏云的 大数据智能推荐系统

项目开发文档

小组成员：丛子渊 张欣雨 周映希 贾婷婷 赵亚楠

所在专业：软件工程

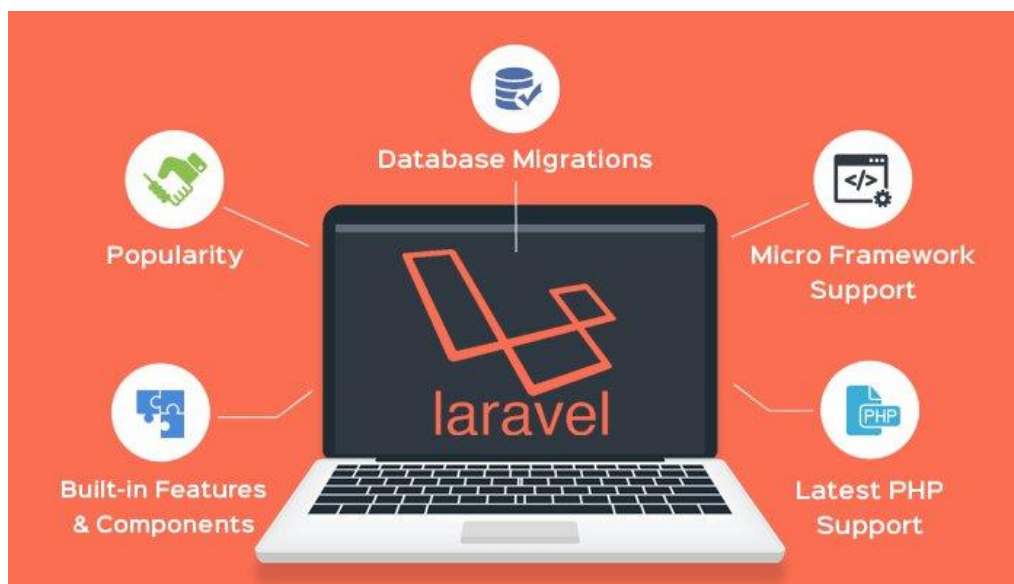
任课教师：张迪

目录

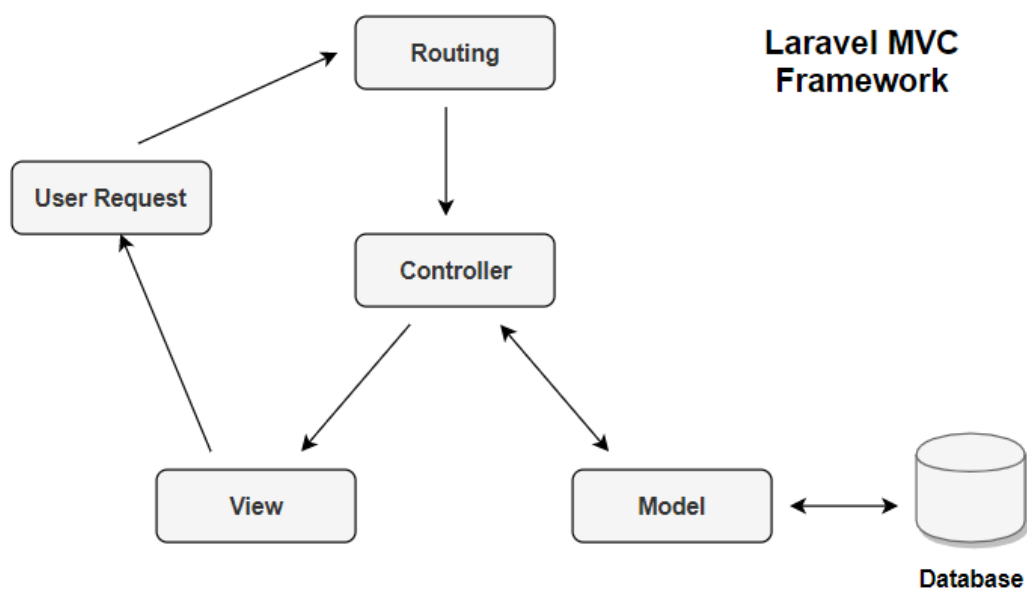
1. web 的设计与实现	3
1.1 web 设计	3
1.2 实现	4
2. 爬虫设计与实现（基于 Scrapy 的爬虫设计）	6
2.1 爬取流程	6
3. 推荐系统设计	8
3.1 程序描述	8
3.2 功能	8
3.3 输入项	10
3.4 输出项	10
3.5 算法	10
3.6 流程逻辑	11
3.7 具体代码	12
3.8 注释设计	13
3.9 测试计划	13
3.10 尚未解决的问题	13
4. 系统测试	14
4.1 测试目标	14
4.2 测试策略	14
4.3 测试结果	14
5. Git 提交记录截图	16

1. web 的设计与实现

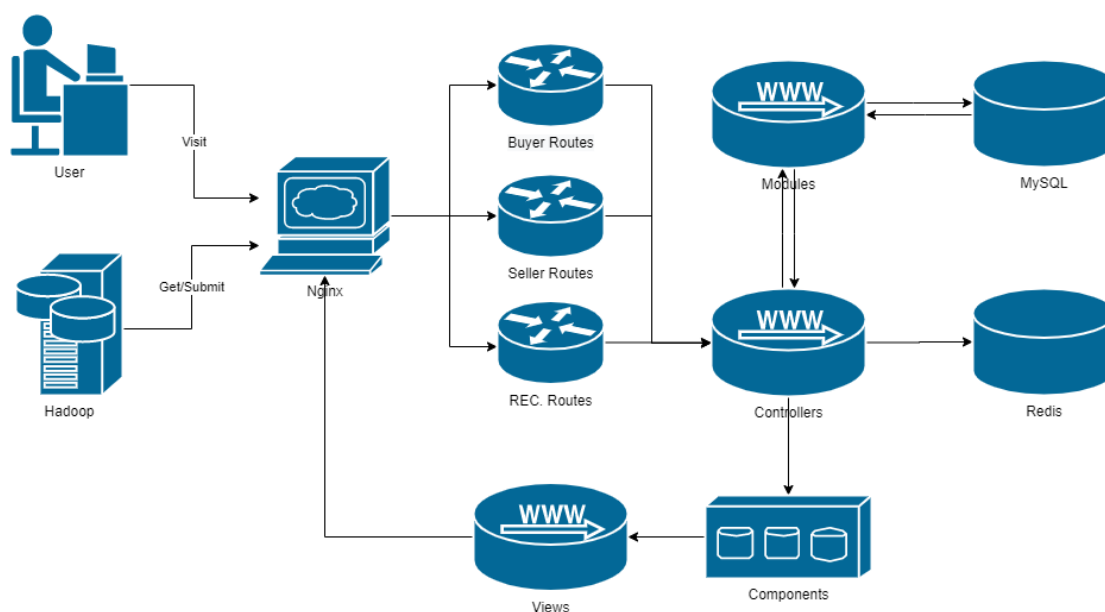
1.1 web 设计



主要使用 Laravel 和 Bootstrap 框架，使用 MVC 架构，将用户请求通过路由转发到指定控制器进行业务实现，并整合前端组件反馈用户。



其中，如个性化推荐系统中提到的，我们将用户图书浏览行为记录至 Redis 并为 Hadoop 提供 API 拉取数据，减少性能损失，并且相对基于日志分析效率和可操作性有很大提高；每次拉取成功 Web 端将不作保留，将数据全部留存在 Hadoop 分布式存储中，体现大数据的实际应用场景。



1.2 实现

路由实现：将用户请求转发到相关控制器处理

```

$router->resources([
    'buyer/rec' => Buyer\RECController::class,
    'buyer/store' => Buyer\StoreController::class,
    'buyer/cart' => Buyer\CartController::class,
    'buyer/order' => Buyer\OrderController::class,

    'seller/store' => Seller\StoreController::class,
    'seller/order' => Seller\OrderController::class,

]);

Route::get('interest/get', '\App\Admin\Controllers\InterestController@get');
Route::get('interest/pull', '\App\Admin\Controllers\InterestController@pull');
Route::any('interest/recommend',
'\App\Admin\Controllers\InterestController@recommend');
  
```

控制器实现：读取数据对象，整合前端组件并返回

```

protected function grid()
{
    $grid = new Grid(new CartModel);
    //$grid->model()->where('user_id', Admin::user()->id);
    $grid->column('book.picture')->image();
    $grid->column('book.id', 'No. ');
    $grid->column('book.title', 'Title')->limit(20);
    $grid->column('quantity', 'Quantity')->label('info');
  }
  
```

```

$grid->column('book.price', 'Unit Price')->display(function ($price) {
    return $price . ' USD';
})->label('info');

$grid->filter(function($filter){
    $filter->disableIdFilter();
    $filter->like('book.id', 'No. ');
    $filter->like('book.title', 'Title');
});

$grid->disableCreateButton();
// $grid->disableActions();
$grid->actions(function ($actions) {
    //$actions->disableDelete();
    //$actions->disableEdit();
    $actions->disableView();
    $actions->add(new OrderAction);
});
$grid->disableExport();

$grid->batchActions(function ($batch) {
    //$batch->disableDelete();
    $batch->add(new BatchOrderAction);
});

return $grid;
}

```

模型：使用 Active Record 设计模式，实现数据对象到关系数据库的映射

```

class OrderModel extends Model
{
    use SoftDeletes;

    protected $table = 'hero_orders';
    public $timestamps = true;
    protected $fillable = [];

    protected static function boot()
    {
        parent::boot();
        //limit user_id
        static::addGlobalScope('user_limit', function (Builder $builder) {
            $builder->where('user_id', Admin::user()->id);
        });
    }
}

```

```
}

public function book()
{
    return $this->belongsTo(BookModel::class, 'book_id', 'id');
}

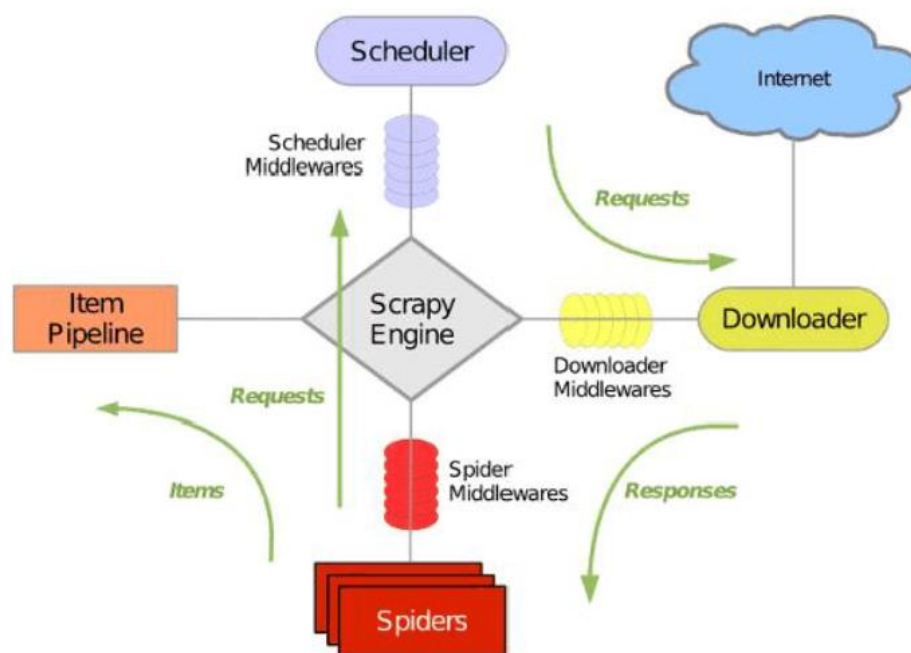
}
```

2. 爬虫设计与实现（基于 Scrapy 的爬虫设计）

Scrapy 框架：Scrapy 框架是用 Python 语言实现可爬去网页提取结构化数据的应用框架。Scrapy 框架包括 Scapy Engine（Scrapy 引擎）、Scheduler（调度器）、Downloader（下载器）、Spiders（爬虫）、Item Pipeline（管道）、Downloader Middlewares（下载器中间件）等组件。

2.1 爬取流程

- （1）、引擎从调度器中取出一个链接(URL)用于接下来的抓取
- （2）、引擎把 URL 封装成一个请求(Request)传给下载器
- （3）、下载器把资源下载下来，并封装成应答包(Response)
- （4）、爬虫解析 Response
- （5）、解析出实体（Item），则交给实体管道进行进一步的处理
- （6）、解析出的是链接（URL），则把 URL 交给调度器等待抓取



Scrapy Engine

这是 Scrapy 的爬虫核心，主要负责控制系统各个部件之间的 data flow，以及当特定事件发生的时候触发 event。

Scheduler

接受 engine 发来的 requests 放入队列中,当 engine 要求的时候再提供给 engine。

Downloader

负责拉取下载 page 并返给 engine，engine 之后再传递给 Spiders

Spider

自行编写的代码类，这部分的代码主要完成解析 response 并提取 item，或者是跟进页面中获取的额外的 link url。

1. 以初始的 URLRequest，并设置回调函数，当该 request 下载完毕并返回时，将生成 response，并作为参数传递给回调函数。spider 中初始的 request 是通过 start_requests() 来获取的。start_requests() 获取 start_urls 中的 URL，并以 parse 以回调函数生成 Request
2. 在回调函数内分析返回的网页内容，可以返回 item 对象或者 Request，返回的 Request 对象之后会经过 Scrapy 处理，下载相应的内容，并调用设置的 callback 函数。
3. 在回调函数，通过 xpath，css 等方法获取我们想要的内容生成 item
4. 最后将 item 传送给 pipeline 处理

Item Pipeline

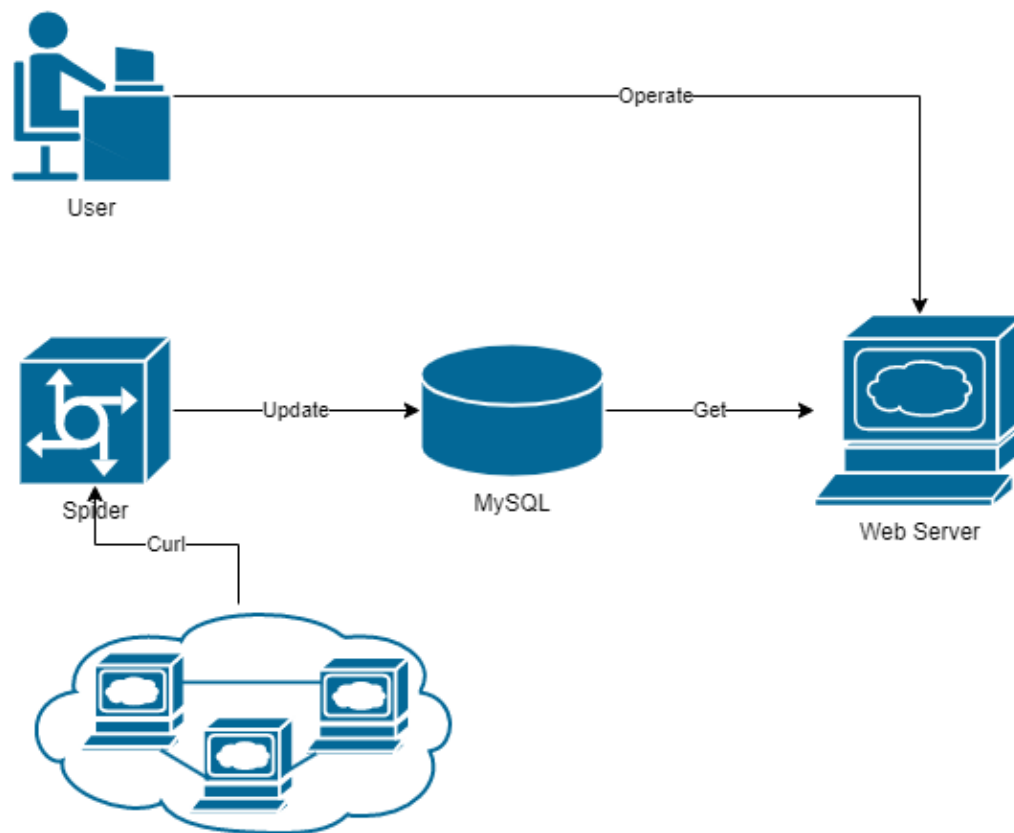
当 Item 在 Spider 中被收集之后，就会被传递到 Item Pipeline 中进行处理。每个 item pipeline 组件是实现了简单的方法的 python 类，负责接收到 item 并通过它执行一些行为，同时也决定此 Item 是否继续通过 pipeline，或者被丢弃而不再进行处理

item pipeline 的主要作用：

清理 html 数据

验证爬取的数据

去重并丢弃
讲爬取的结果保存到数据库中或文件中



导入数据库:

爬取后的数据存入 MySQL，Web 从 MySQL 中取出数据呈现给用户，用户可在 Web 端对数据进行操作。

3. 推荐系统设计

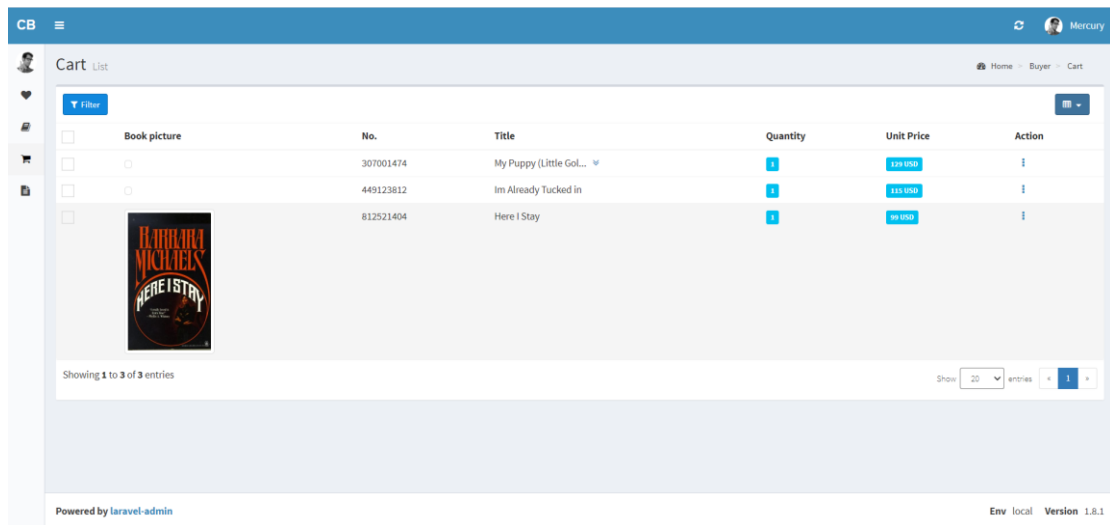
3.1 程序描述

根据用户历史行为，结合 hadoop 大数据处理技术，发现用户与图书的相关性，为用户推荐可能感兴趣的图书。本项目所用到的是基于物品的协同过滤推荐。

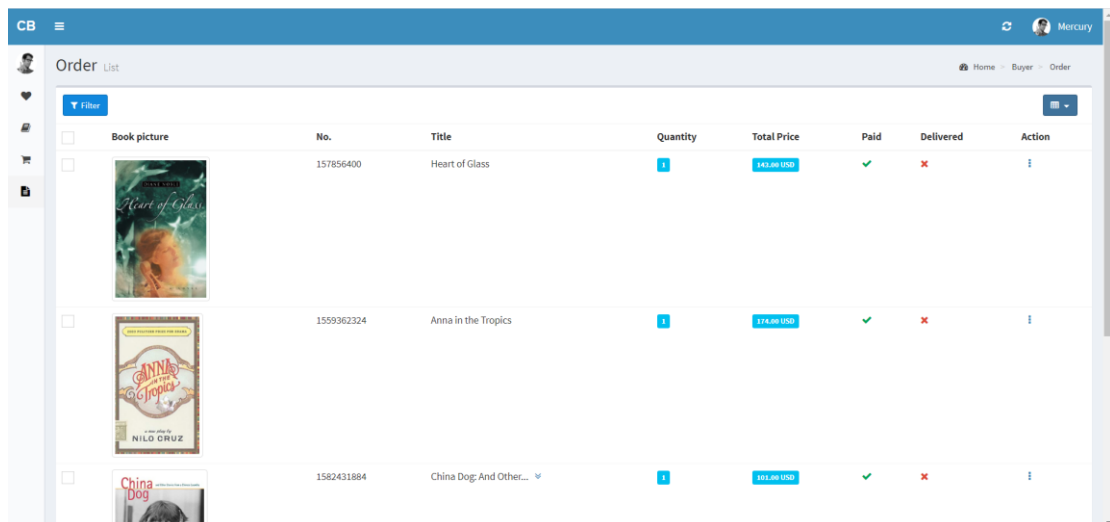
3.2 功能

根据用户行为，如添加购物车、购买图书等，在推荐页面为已注册用户推荐相关分类的图书。

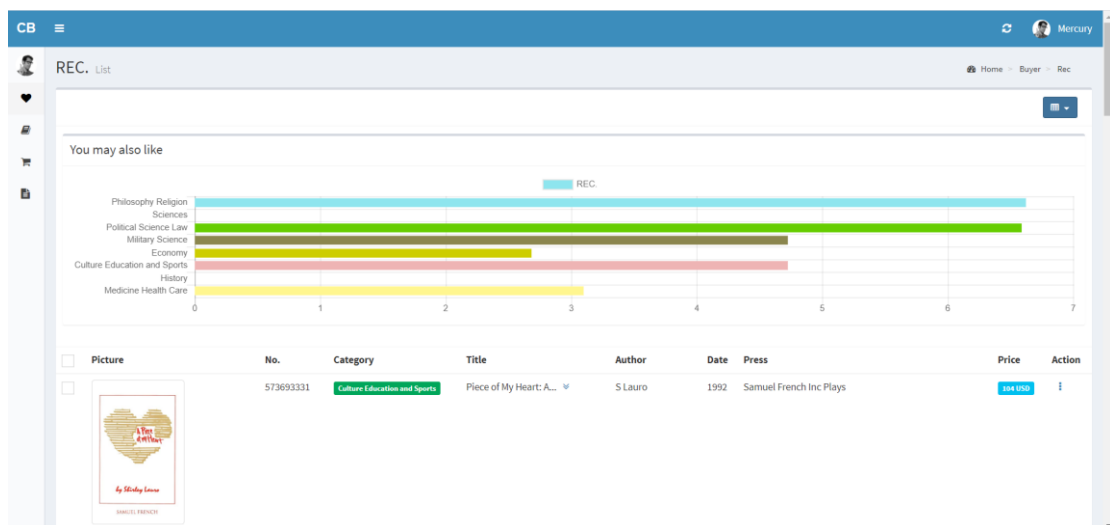
功能实现如下：
在购物车中添加图书。



在订单页进行支付



点击推荐页，会根据已购或加入购物车的图书，推荐相应的分类以及相应分类的图书。



3.3 输入项

鲲鹏云搭建的 web 向推荐系统输入用户行为 json 数据。

数据名称	数据类型	取值范围
用户行为表	Json	--

3.4 输出项

Json 数据经过循环处理后得到用户偏好表，并上传至 hadoop 集群，经过推荐算法处理得到最终的推荐结果。

数据名称	数据类型	取值范围
用户偏好表	Csv	--
结果表	Json	--

3.5 算法

对用户的行为进行分析得到用户的偏好后，可以根据用户的偏好计算相似用户和物品，然后可以基于相似用户或物品进行推荐，即基于用户的协同过滤和基于物品的协同过滤。

关于相似度的计算，现有的几种方法都是基于向量(Vector)的，其实也就是计算两个向量的距离，距离越近相似度越大。在推荐场景中，在用户-物品偏好的二维矩阵中，我们可以将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，或者将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度。

详细的代码如下：

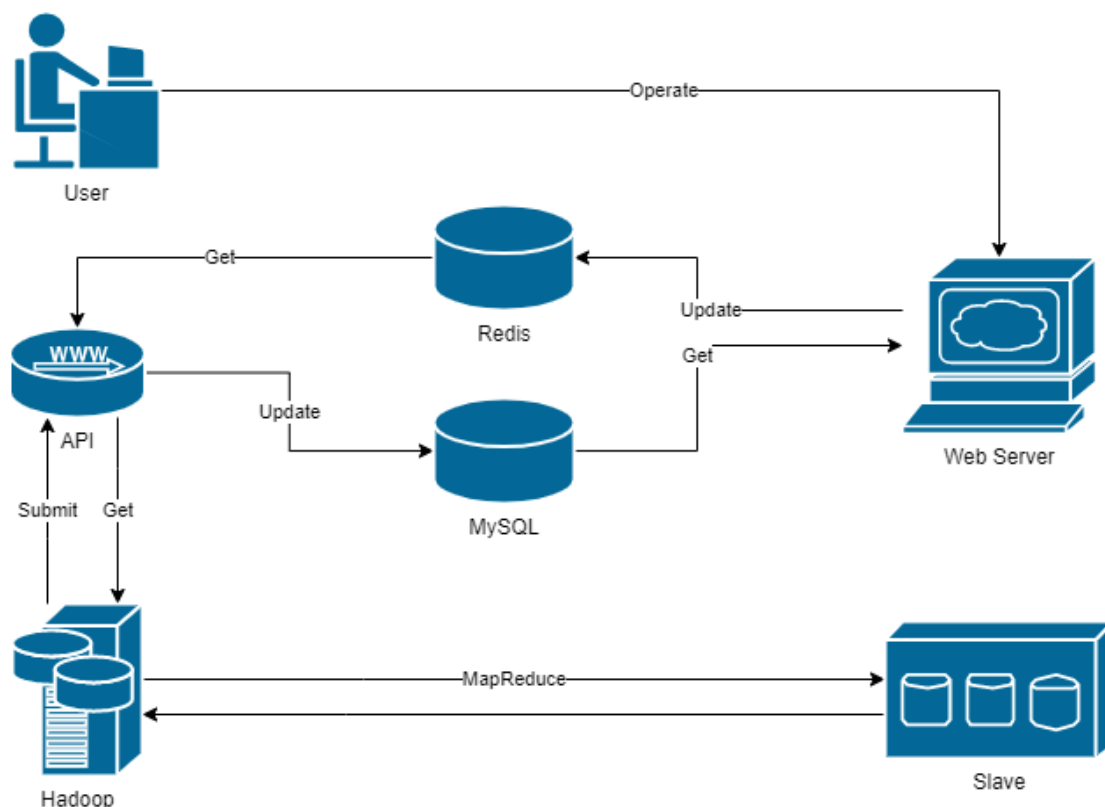
```
# prepare total and relation data
for user, categories in user_category.items():
    # category total count by user (once per user)
    for category in categories:
        if category not in total_category:
            total_category[category] = 0
        total_category[category] += 1
    # category relation count by user (once per user)
    for cat1 in categories:
        for cat2 in categories:
            if cat1 == cat2:
                continue
            sim_matrix.setdefault(cat1, {})
            sim_matrix[cat1].setdefault(cat2, 0)
            sim_matrix[cat1][cat2] += 1

# calculate similarity matrix
for cat1, related_cat2 in sim_matrix.items():
    for cat2, count in related_cat2.items():
        sim_matrix[cat1][cat2] = count / math.sqrt(total_category[cat1] * total_category[cat2])
# print(sim_matrix)

# recommend
for user in user_set:
    for category, rating in user_category[user].items():
        for related_category, sim in sorted(sim_matrix[category].items(), key=itemgetter(1), reverse=True):
            if related_category in user_category[user]:
                continue
            rec_category.setdefault(user, {})
            rec_category[user].setdefault(related_category, 0)
            rec_category[user][related_category] += sim * rating
```

3.6 流程逻辑

用户在 web 进行操作生成用户行为数据，并提交到 redis 中存储，通过 API 获取 redis 中的数据，并在 hadoop 集群中实现分布式推荐算法，然后将计算得出的推荐结果，通过 API 更新到 MySQL 中，最终返回至 web 页面显示推荐内容。



3.7 具体代码

首先取数据，并将数据格式化处理成 map 和 reduce 所需要的格式，然后是 hadoop 中的处理，把数据上传到 hdfs，删除之前的 output，并通过 hadoop streaming 提交 mapreduce，最后从 hdfs 中取出 hadoop 的运算结果。

```

base_url = r"http://kunpeng-web.uranme.com/interest/"
item_path = r"/root/Python_Hadoop/itemcf/item.csv"
result_path = r"/root/Python_Hadoop/itemcf/part-00000"

# get api data
api_data = requests.get(base_url+"pull")
api_data = api_data.json()

# Loop api data
for user_id in api_data:
    for category_id, books in api_data[user_id].items():
        category_times = 0;
        for book_id, times in books.items():
            category_times += int(times)
        # append to text
        with open(item_path, 'a+') as f:
            f.write(user_id+'_'+category_id+'_'+str(category_times)+'\n')

# hadoop
HADOOP_BIN = r"/home/hadoop/hadoop-3.1.3/bin/hadoop"
HADOOP_STREAMING = r"/home/hadoop/hadoop-3.1.3/share/hadoop/tools/lib/hadoop-streaming-3.1.3.jar"
# put hdfs
os.system(HADOOP_BIN+" fs -put -f /root/Python_Hadoop/itemcf/item.csv /python/itemcf/input")
# remove previous output
os.system(HADOOP_BIN+" fs -rm -r /python/itemcf/output")
# itemcf
os.system(HADOOP_BIN+" jar "+HADOOP_STREAMING+" -mapper /root/Python_Hadoop/itemcf/mapper.py -reducer /root/Python_Hadoop/itemcf/reducer.py -input /python/itemcf/in")
# get result
os.system(HADOOP_BIN+" fs -get -f /python/itemcf/output/part-00000 /root/Python_Hadoop/itemcf")

```

读取 hadoop 的运行结果，提交到 web 系统。

```
# submit
def read_input(file):
    with open(file, 'r') as f:
        for i, line in enumerate(f):
            yield line.strip('\r\n')

result = {}
for line in read_input(result_path):
    user, category_id, rating = line.split(',')
    result.setdefault(user, {})
    result[user].setdefault(category_id, rating)

JSON_RESULT = json.dumps(result)

response = requests.post(
    url = base_url+"recommend",
    headers = {'Content-Type': 'application/json'},
    # headers = {'Content-Type': 'text/xml'},
    data = JSON_RESULT,
)
```

3.8 注释设计

- (1) 模块首部的注释，要大体上说明模块要实现的功能
- (2) 各个语句后面的注释要说明语句实现的功能
- (3) 注释要规范，命名要标准

3.9 测试计划

测试名称	测试进度安排	测试目的	测试内容
基本数据输入	系统完成后	系统功能是否达到要求	输入简单的一致性的数据
非法数据输入	基本数据测试后	系统对非法数据的反应	输入一些特殊字符和字符串、边界值的数据
空字符串	非法测试以后	系统对空字符串的反应	在信息上输入空字符

3.10 尚未解决的问题

可以结合应用 UserCF 进一步提高推荐针对性，增强用户体验

4. 系统测试

4.1 测试目标

- (1) 测试系统功能的完整性和正确性。
- (2) 测试系统的健壮性。
- (3) 测试系统需求和设计的完整性和正确性。

4.2 测试策略

模拟真实的运行环境，对系统的预期的各个功能模块进行操作测试，以及对整个系统性能也进行测试。

4.3 测试结果

输入功能测试用例

编号	描述	输入	期待输出
1	登陆和注册	~	用户名只包括字母和数字，用户名在 3 到 20 个字
		12	用户名在 3 到 20 个字
		~12	用户名只包括字母和数字
2	商品搜索	No: 155853086	展示出序号为 155853086 的商品
		Title: Dandelion Wine (Grand Master Editions)	展示这本书的信息
		Category: history	展示出 history 类的所有图书
3	购物车商品搜索	No: 155853086	展示出序号为 155853086 的商品数量价格

		Title: Faulkner's Oxford: Recollections and Reflections	展示出该商品的 数量价格等信息
4	功能模块搜索	Rec	展示出 Rec 模块

峡谷书店系统基本功能测试

基本操作	测试情况	测试结果
注册、登录	登陆已有账号和注册新的账号	测试通过
个人信息设置	设置昵称、头像和修改密码	测试通过
添加商品到购物车之后购物车的显示	显示结果	测试通过
从购物车添加到订单	将商品从购物车删除，在订单添加	测试通过
新用户添加商品到购物车之后推荐显示	显示结果	过了一会儿在推荐界面上显示推荐结果
图书商店显示	正常显示	测试通过
图书搜索和筛选	输入正确的信息显示商品信息	测试通过
图书详情	正确和完整显示商品信息	测试通过
订单管理	订单可以支付和删除	测试通过
订单支付	订单支付成功之后有绿色弹窗提示	测试通过
销售者店铺管理	添加店铺的图书，修改、查看、删除图书	测试通过
销售者订单管理	对订单进行配送或删除	测试通过
管理员用户管理	可以查看、添加、删除、修改、查看用户信息	测试通过
管理员角色管理	可以对角色进行添加、删除、修改、查看	测试通过
管理员权限管理	可以对权限进行添加、删除、修改、查看	测试通过

峡谷书店推荐系统测试评估

功能模块名称	功能测试用例	测试结果
日志收集	用户加入购物车行为和加入订单	可以查询
推荐功能	查看不同用户推荐结果	推荐结果有效

系统性能测试







评估方面	细节	开发系统
系统安全性和稳定性	系统容错性	系统基本上具有容错功能
	系统安全性	系统安全性好
	系统保密性	系统保密性高
	系统稳定性	运行基本稳定
系统界面	界面显示功能	正常

5. Git 提交记录截图

Web

Branch: master
Commit Graph






6 Commits (master)
Search commits...
All Branches
Search

Author	SHA1	Message	Date
 Abyss Cong	6b8657bb3f	cold start	1 day ago
 Abyss Cong	339f7e6030	beta version	1 day ago
 Abyss Cong	3569034dec	seller side implement	3 days ago
 Abyss Cong	dd431da682	implement store, cart and redis api	4 days ago
 Abyss Cong	2523eb90af	second commit	5 days ago
 Abyss Cong	604016bfdb	first commit	6 days ago

大数据 1

Branch: master Commit Graph



5 Commits (master) Search commits... All Branches Search

Author	SHA1	Message	Date
 千秋星垂野	db583a6317	7.12	1 day ago
 千秋星垂野	5744b153b4	7.12	1 day ago
 千秋星垂野	903a96dffd	7.10	3 days ago
 千秋星垂野	02f066beab	7.8	5 days ago
 千秋星垂野	1175642b14	初始化	6 days ago

大数据 2

Branch: master Commit Graph



2 Commits (master) Search commits... All Branches Search

Author	SHA1	Message	Date
 Abyss	59ae0f0b41	beta	2 days ago
 abyss	4097421249	Initial commit	2 days ago

爬虫

Branch: master Commit Graph






2 Commits (master) Search commits... All Branches Search

Author	SHA1	Message	Date
 zyx	12936f315a	second commit	1 day ago
 zyx	bd44ba10c7	first commit	6 days ago

项目仓库

Repository Management (Total: 6)

Search... Search Sort

ID	Owner	Name	Private	Watches	Stars	Forks	Issues	Size	Created	Op.
4	zyx	spider	<input checked="" type="checkbox"/>	1	0	0	0	62 MiB	Jul 07, 2020	
6	zyn	2020summer	<input checked="" type="checkbox"/>	1	0	0	0	10 MiB	Jul 08, 2020	
3	Mercury	Recommend	<input checked="" type="checkbox"/>	1	0	0	0	90 MiB	Jul 07, 2020	
1	abyss	Web	<input checked="" type="checkbox"/>	1	0	0	0	3.0 MiB	Jul 07, 2020	
7	abyss	REC	<input checked="" type="checkbox"/>	1	0	0	0	21 KiB	Jul 12, 2020	
5	abyss	Test	<input checked="" type="checkbox"/>	1	0	0	0	0 B	Jul 07, 2020	