

Gitmining迭代二详细设计文档

第27组



|  |  |  |  |
| --- | --- | --- | --- |
| **变更历史** | | | |
| **版本号** | **更新时间** | **更新人** | **更新摘要** |
| **V1.0** | 2016/3/19 | 董本超 | 初次创建详细设计文档 |

# 1.引言

## 1.1编制目的

本报告对系统设计中的一些关键部分进行详细说明，从而可以更好的指导后续开发工作。详细设计文档仅对设计文档中的一些关键部分进行补充说明，欲了解全面的设计描述请观看体系结构设计文档。

本报告面向开发人员、测试人员和最终用户编写，是了解系统的导航。

## 1.2参考资料

1.IEEE标准

2.需求规格说明文档

3.体系结构设计文档

# 2.产品概述

Gitmining系统是针对github上的用户和仓库数据进行分析统计，从而获得一些直观的数据展示和分析结果的软件。

利用该软件可以对github上的项目和用户进行排序、筛选等操作，从而找到自己感兴趣的一些项目和用户的信息。

# 3.体系结构设计概述

Gitmining项目开发总体采用螺旋模型，分为3个迭代过程。细节上，本系统采用分层模型，客户端分为展示层，业务逻辑层，并通过RMI和服务器相连；服务器分为数据层和集成层，数据层负责通过RMI为逻辑层提供数据服务，集成层负责不定期从互联网更新本地数据。

# 4.结构视角

# 5.数据视角

由于迭代二阶段不允许使用数据库，而运行时直接从gitmining或github 的结合口获取数据太慢无法接受，我们将所有需要的数据都事先放在服务器端，有客户端在运行时调用。

为了向迭代三过度，不至于在迭代三阶段修改太多关于数据组织方面的设计，我们存在服务器端的数据也尽可能的模拟数据库表的形式，存储数据如下：

1. repobyname.txt，List<RepoBriefPO>序列化文件，存放服务器中所有已存的仓库索引，搜索时就从该列表中进行检索并将附符合求的RepoBriefPO转化成RepoBriefVO再放进一个list里放回客户端。

2. userbyname.txt，与repobyname.txt功能类似，用于存储用户简略信息的索引。

3. nameToRepo.txt，Map<String, RepoPO>序列化文件，存放每个仓库名到该仓库详细信息RepoPO的映射，在需要获取单个仓库的详细信息的时候访问该map。在将来迭代三这个map将成为数据库中的一张存储仓库详细信息的表。

4. nameToUser.txt，Map<String, UserPO>序列化文件，与上一条nameToRepo.txt类似，存放用户详细信息。

以下的5—11都是map<String, List<String> >，它们都是repo和user之间关系的映射，是为了消除RepoPO和UserPO中的一对多的树状结构而设计的（因为在迭代三时数据库不允许一列对应多个值）。

5. repoToCollab.txt，一个仓库的所有合作者的姓名

6. repoToContributor.txt，一个仓库的所有贡献者的姓名

7. repoToSubscriber.txt，一个仓库的所有订阅者的姓名

8. userToCollabRepo.txt，一个用户所有参加合作的仓库的名字

9. userToContribute.txt，一个用户所有贡献的仓库的名字

10. userToOwnRepo.txt，一个用户所有拥有的仓库的名字

11. userToSubscribeRepo.txt，一个用户的所有订阅的仓库的名字

## 5.1 Repository模块的数据分析

### 5.1.1 repository文字信息界面

仓库信息需要显示的非图表信息总共包括：拥有者姓名，仓库名称，大小，语言，github地址，描述，创建时间，更新时间，点赞数，fork数，订阅数，合作者数，贡献者数。

其中搜索列表中的简略信息包括：拥有者姓名，仓库名称，描述，点赞数，fork数，订阅数，更新时间。

仓库文字信息页面中需要显示一张雷达图和一张活跃曲线图。

**一．雷达图**

仓库的雷达图共有5个维度，分别是仓库大小（size）、仓库被引用次数（fork）、仓库受欢迎程度（popular）、仓库协作复杂度（complexity）和仓库内容的更新活跃度（avtive）。

size维度根据仓库中文件总量的大小来生成，fork维度根据仓库被别的用户fork的次数生成，popular维度根据仓库的点赞数num\_stars和被订阅数num\_subscrebers加权算出，complexity维度根据仓库的贡献者数num\_contributors和协作者数num\_collaborators加权算出，active维度根据仓库的commit数、issue数量以及pull数量加权算出。

雷达图中的所有数据都需要将所有仓库的对应数据进行归一化处理，生成0到1之间的浮点数来表示每个维度的值。

**二．活跃曲线图**

仓库的活跃曲线图将根据仓库自创建时间（create\_at）以来到最后更新时间（update\_at）之间每天的commit数、issue数、pull数来统计仓库每天的活跃度，并将活跃度放在折线图上用来显示活跃度随天数的变化。

### 5.1.2 repository图表信息界面

**一．创建时间柱状图**

根据仓库创建时间的分布，显示不同年份创建仓库数量的不同和变化趋势。

**二．fork量柱状图**

根据不同仓库fork量的大小，将fork量分为多个区间，显示不同区间内对应仓库的数量。

**三．star量柱状图**

根据不同仓库star量的大小，将star量分为多个区间，显示不同区间内对应仓库的数量。

**四．language饼图**

统计所有仓库的编程语言使用情况，将不同语言的使用比例用饼图显示出来。

## 5.2 User模块的数据分析

### 5.2.1 user文字信息界面

用户信息需要显示的非图表信息总共包括：登录名，用户类型，姓名，公司，博客地址，居住地址，email，个人简介(bio)，被关注数，关注数，创建时间，更新时间，拥有仓库数，gist数，订阅项目数，贡献项目数，参与项目数，头像url。

其中搜索列表中的简略信息包括：都登录名，关注数，被关注数，拥有仓库数，创建时间，更新时间，公司。

用户文字信息界面还需要一张雷达图用来显示用户相关属性在全体user中占的地位。

**一．雷达图**

用户雷达图总共包含5个维度：用户积极性（enthusiasm）、用户的粉丝数（followers）、用户价值（value）、、repo数量（repos）和gist数量（gists）。

用户积极性根据用户contribute的、collaborate的和subscribe的项目数量加权生成；粉丝数根据follower生成；用户价值的生成比较复杂，首先要遍历该用户所有own的和collaborate的项目仓库，然后再计算这些项目的被点赞数（star）、fork数以及被订阅数（subscribe），根据这些项目的这三个属性来判定这些项目价值，再把这些项目的价值加起来后适当处理得到该用户的价值。最后，repo数量和gist数量都可以从用户的基本属性中直接得到，比较简单。

这5个维度的值都需要归一化为0到1之间的浮点数。

### 5.2.2 user图表信息界面

**一．用户类型饼图**

公司或个人所占的比例。

**二．账户注册时间折线图**

统计不同年份中新注册用户量，用折线图反映出用户注册量的变化趋势。

**三．参加项目数柱状图**

统计用户参加项目数的不同区间内的人数，比如参加0-49个项目的有多少人，参加50-99个项目的有多少人。

**四．创建项目数柱状图**

统计用户创建项目数的不同区间内的人数，比如创建0-9个项目的有多少人，创建10-19个项目的有多少人。

**五．不同公司用户数量**

统计几个用户量最多的大公司的用户数量，对于没有公司信息的用户不计入统计，对于一些只有非常少的用户的公司放入“其他”分类。