**Gitmining设计文档**

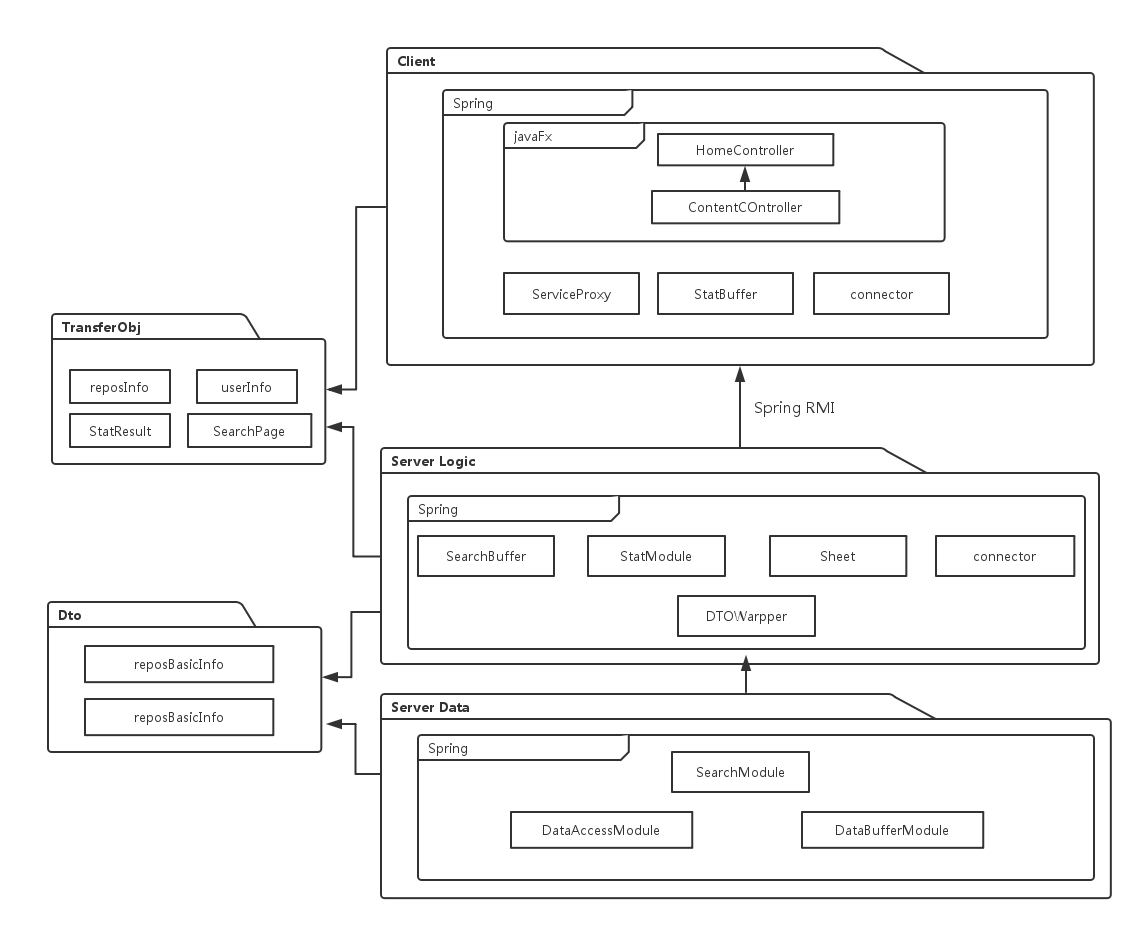
Gitmining当前功能主要分为搜索和数据统计，下面列出设计时面临的最主要问题：

1. 仓库总数为3000余个，用户总数为50000余个，如何根据关键字快速进行搜索。
2. 如何快速在搜索结果上分析数据，或者在全体仓库/用户列表中分析数据。

最终的解决方案是复杂的，可以概括为：

* 在客户端使用延迟加载来处理repos对象和user对象，用户点击特定对象才加载其具体信息。这对UI层是透明的。
* 搜索采用分页查询，page对象内部调用服务器。因此访问服务器对UI层是透明的。
* 建立统计service的代理，将统计结果缓存在本地。
* 服务器建立对单个关键字搜索结果的缓存，这大大提高了之后对搜索结果统计的速度，和搜索同一关键字的速度以及分页查询的速度。
* 数据层在内存中保存所有user和repos的基本信息对象，提高了对基本信息统计的速度。

体系架构简图



下面为项目设计中的重要设计决策：

## 在迭代二中，除了标准需求外，为项目增加哪些需求？

对于单个repos

* 增加了对不同语言代码行数的统计的饼图。

对于单个user

* 增加了对一周内活动的统计的折线图，如commit、pull等。

对于user搜索结果

* 增加了对最常用语言比例的统计的饼图。
* 增加了fellower，fellower，contributed repos均值的统计的柱状图。
* 增加了某一用户与均值的比较结果统计。

对于repos搜索结果

* 增加了使用语言比例的统计。
* 增加了star，fork，contributors均值的统计。
* 增加了单个repos与均值的比较结果统计。

对于repos总体

* 增加了star-fork的散点分布统计。
* 增加了stat-contributor的散点分布统计。

## 解析json是使用jackson还是json-lib？

jackson，对比和查阅资料后发现，jackson的体积更小，maven上usage更多，jar包更少。而在个别json属性过多，如repostory的属性有一百多个的情况下，它提供的基于annotation的功能非常的好得处理，我们可以选择是丢弃某些属性还是将他们保留在map里。

## java如何调用python？

利用python脚本语言的特性，我们让java程序通过命令行调用.py文件，并通过输入流调用特定方法，通过输出流获得输出。java和python之间的数据交换依据json格式，图片交换考虑使用python保存本地文件然后java读取文件的格式。

## 是用gitmining.api还是api.github？

迭代1使用gitmining.api，之后可能会使用api.github。

gitmining优势在数据总量可控，一些api调用起来很方便，但是也存在一些问题：大部分repos的创建者是organization，不同规模的organization可能可比性不强。repos中查询contributor一般是user，经常出现查无此人的情况。这些信息不全可能在展示时带来一些问题。

api.github的数据是完整的，但是量实在太大。它提供了内置的搜索和排序功能，而且已经有封装好的github api for java可以直接调用。一些方面的api没有gitmining.api方便。

## 全部gitmining数据存入内存会占用多大空间?

大约70M到80M。

## 加载gitmining数据，并且建立数据结构要多长时间？

大约1到1.5秒，实际操作中采用延迟加载策略。

## 需不需要用分页查询？

逻辑层接口采用分页，数据对逻辑不采用分页。考虑到用户看到的信息可能是已经经过关键字搜索，或者经过tag过滤过的，所以让数据层返回全部的搜索结果。而界面上一次显示的结果十分有限，所以logic层接口采用分页形式。

## 关键字模糊查找的算法用在哪里？

数据层。假如关键字搜索算法写在逻辑层，那么必然要将所有信息都读入逻辑层进行排序，可能占用大量内存。假如写在数据层，可能会避免读入大量信息，从而节省内存。

## 排序算法用在哪里？

逻辑层。因为搜索结果的数量相对较少，而且用户对排序的切换可能比较频繁，多次读取数据层可能造成性能损失。所以采用在逻辑层排序的方法。

## 是否对list属性延迟加载？ 比如repos的贡献者，合作者，user的创建过的项目等属性。

是。因为其中大部分信息在搜索结果上只显示数量，不显示具体。当用户想查看具体信息时，再去加载它们，这样能节省内存。

## 逻辑层是否应该缓存某一关键字的搜索结果？

是。原因有 1.目前数据不会动态变化，2.访问数据层开销较大 3.当用户对搜索结果用不同方式排序时，只需要重新在缓存中排序即可 3.对于客户每次的搜索，服务器要进行一定的数据统计。缓存使得搜索的逻辑和统计的逻辑可以不必连续完成，可以分开进行。

## 何时清理数据层缓存？

客户端服务器建立缓存刷新机制：客户端每5秒刷新一次服务器的缓存，服务器每十秒钟清理一次未被刷新的缓存数据。这样首先能及时清理不需要的缓存数据，另一方面若两个客户端搜索相同关键字，一个关闭时，缓存不会被误删。

## 当使用者在repos详细信息中点击user信息该如何做？

逻辑层增加接口，可以精确根据login查询用户。

## 当使用者在user详细信息中点击repos信息该如何做？

逻辑层增加接口，可以精确根据ownername和reposname返回repos。

## 界面如何布局？

顶上是logo和搜索栏，需求变更是在上边增加选项卡。

左侧信息栏，用来显示搜索结果的缩略统计或user,repos的详细信息。右侧是标签栏+竖向scroll列表，用来显示搜索结果或者详细信息中的列表。若增加新的功能也可在右侧增加标签栏页。

## 如何处理http访问速度慢的问题？

imageProxy和chartProxy未加载完时显示loading动画。客户端网络层建立缓存，缓存总体统计数据。

## 逻辑层如何进行数据统计？

数据层提供getAll方法，返回所有repos或者user的基本信息list，逻辑层进行计算。

## 对于总体情况的统计耗时过长，如何解决？

考虑到repos和user总体情况是很少发生变更的，因此客户端内存中缓存总体情况统计结果，第二次访问时从内存中读取。

## 目前数据存在文件里，如何应对将来更换为数据库的变更？

数据层内建立数据访问接口，若更换数据库，将数据访问接口的实现换为数据库实现即可，其余不用变动。