

**信息与软件工程学院**

**软件工程综合训练中期报告**

课程名称： 软件工程综合训练

课题名称： 多Linux系统间的数据传输

指导教师： 刘玓

所在系别： 软件工程国际班

执行学期： 大三上

学生信息：

|  |  |  |
| --- | --- | --- |
| 序号 | 学号 | 姓名 |
| 1（组长） | 2019091608028 | 卞加勉 |
| 2 | 2019091604015 | 石佳一 |
| 3 | 2019091604016 | 夏野 |
| 4 | 2019091201005 | 张运辰 |
| 5 | 2019091201020 | 彭冲 |
| 6 | 2019091605012 | 钟镇阳 |
|  |  |  |

目 录

第一章 综合设计的进展情况 1

1.1 针对工程问题的方案设计 1

1.2 针对工程问题的推理分析 1

1.3 针对工程问题的具体实现 1

1.4 知识技能学习情况 1

第二章 存在问题与解决方案 2

2.1 存在的主要问题 2

2.2 解决方案 2

第三章 前期任务完成度与后续实施计划 3

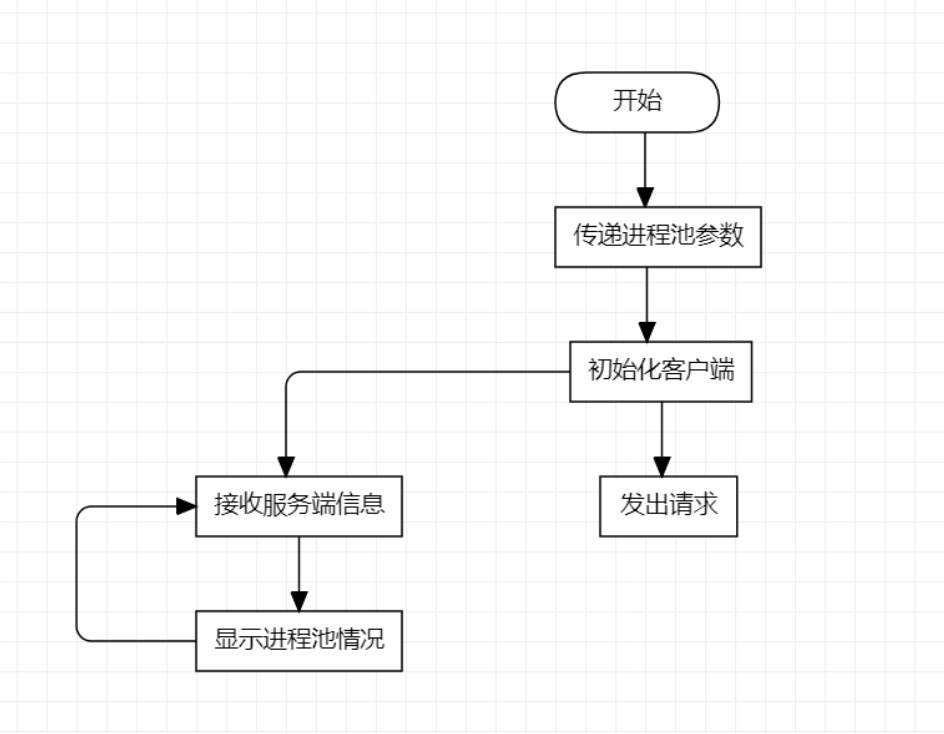
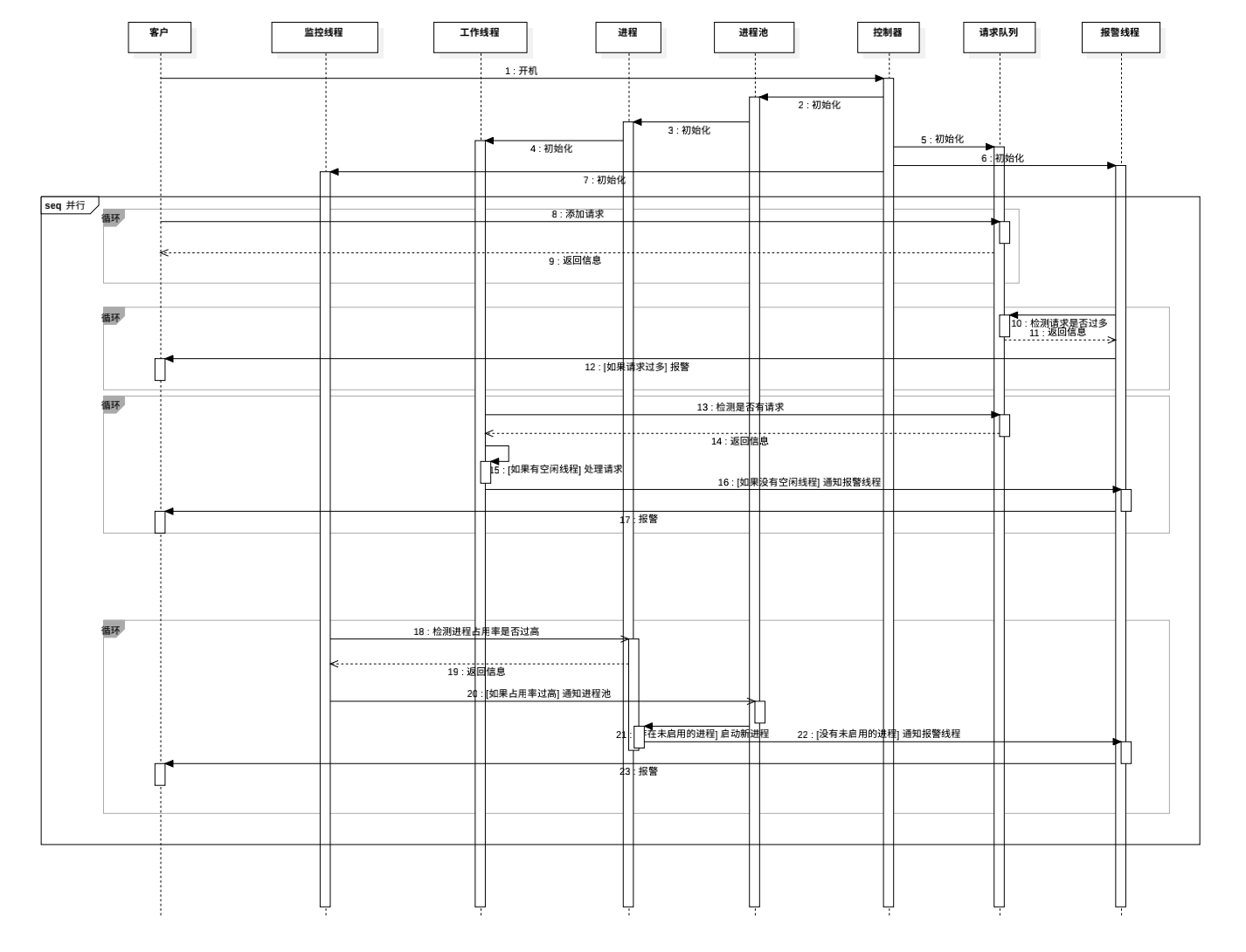
参考文献 4

**说明:**

1. **报告要求2000字以上。**
2. **本模板仅为基本参考，请各位同学根据个人情况进行目录结构扩展。**
3. **报告正文必须双面打印。**

# 第一章 综合设计的进展情况

1. 针对工程问题的方案设计

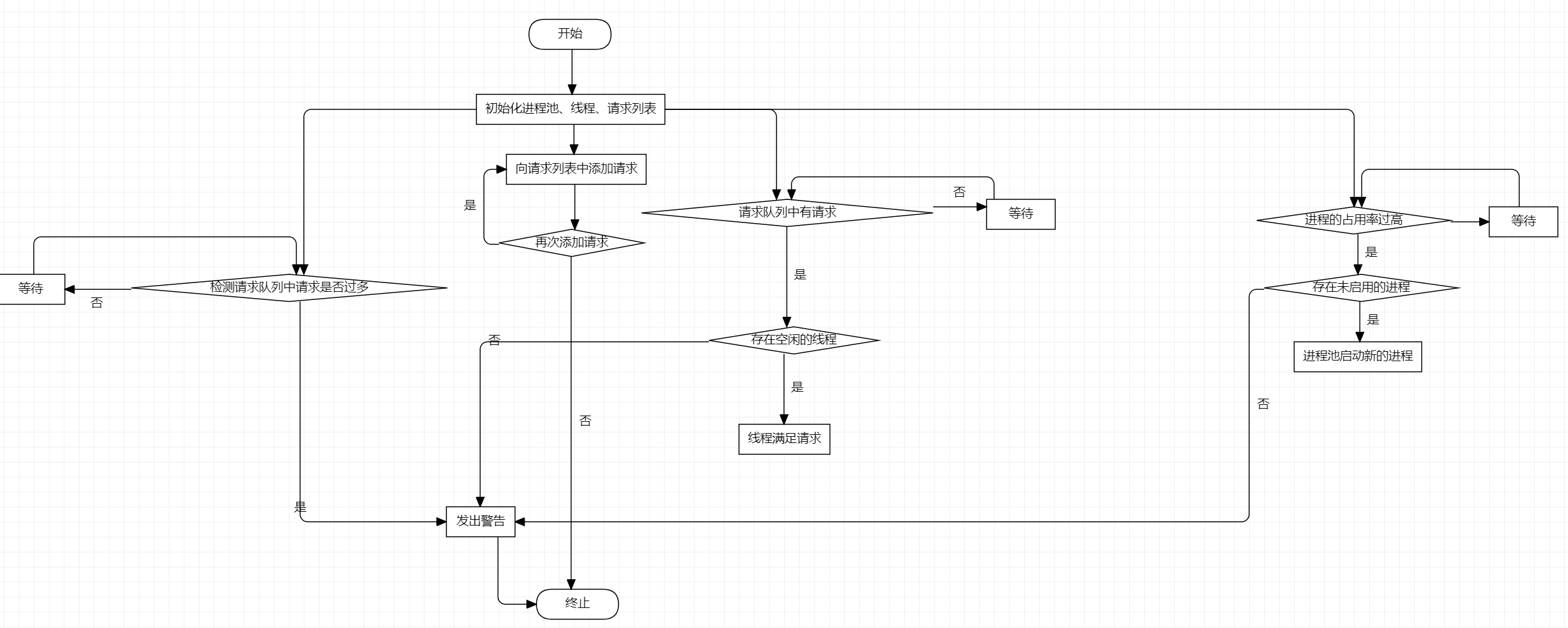
  


1. 针对工程问题的推理分析
2. 确定开发功能

首先这里有两个对象，一个是Client端，一个是Server端。用户能够通过图形化界面设置进程个数；其次，每个服务进程的可服务的请求数量是确定的，当请求数量不断增多时，逐渐启动进程池中的服务进程；当请求数量减少时，则降低服务进程的数量。当请求数量超过进程池中所有服务进程能够处理的最大数量时，则系统给出告警信息；最后，用户能够通过前端界面看到进程的个数变化，以及动态检测进程池的变化。

1. 设计开发

通过前面功能的定义，小组成员绘制了流程图，完成了概要设计。



1. 代码开发

使用python在本地IDE上进行后端的实现，使用微信小程序开发工具实现前端界面的展示。

1. 产品测试

第一阶段，前端设计好后，用户能够通过微信小程序设置进程的数量，同时界面能够显示出当前进程数；第二阶段，当用户不断申请进程后，如果进程池满了，那么用户将在界面上看到警告信息，并且用户无法再申请新的进程。

1. 针对工程问题的具体实现

本次课题主要是多进程的平衡调度问题，同时，我们想的还有更多。要求能够自行设置进程的数量以及进程池的动态变化，因此我们打算设计一个图形用户界面，形成一个窗口，而不仅仅是控制台显示的字符界面。在实现的时候，会遇到两个问题：一个是后端实现进程调度平衡的编程语言，一个是实现前端展示的编程语言，同时这两种语言间有很好的接口进行关联。因此，我们最后选择编程工具是Python和微信小程序前端设计语言WXML、WXSS。

Python编程语言最大的优点之一，是其具有伪代码的特质，它可以让我们在开发 Python 程序时，专注于解决问题，而不是搞明白语言本身。针对于本次课题，python中提供了大量的开发模块能够迅速的解决进程的创建和销毁，并且由于 Python 是开源的，它已经被移植到许多平台上。如果能够避免使用依赖系统的特性，那就意味着，所有 Python 程序都无需修改就可以在好多平台上运行，包括 Linux 、Windows、FreeBSD、Solaris 等等，甚至还有 PocketPC、Symbian 以及 Google 基于 Linux 开发的 Android 平台。解释型语言几乎天生就是跨平台的。Python 作为一门解释型的语言，它天生具有跨平台的特征，只要为平台提供了相应的 Python 解释器，Python 就可以在该平台上运行。

因此，鉴于python的优势，我们选择微信小程序这样的高级前端实现。WXML和WXSS的编程方式和HTML于CSS基本一致，甚至可以说是就是从他们演变过来的。同时，使用此开发平台能够让最终的UI界面更方便的展示出来。仅仅需要通过手机微信扫描二维码就能够进入小程序，完成进程的创建、进程数量的变化过程以及进程池的运行状态。

最后，对于技术标准，就是根据python的手册执行的代码撰写。

1. 知识技能学习情况

使用Python作为该课题的主要编程语言，编程工具为 Pycharm ， docker 作为容器为前端提供服务。同时，代码存放在 Github 上，能够让小组中每一个人都能在上面实时更新代码和数据。

1. Python学习

首先由于部分小组成员没有接触过 Python ，就阅读《Python基础教程》大致学习了 Python 的基础语法，同时由于已经了解进程和线程的概念，因此又学习了 Python 的进程和线程调度机制以及如何使用 Python 创建 thread 。

1. Docker

Docker 技术使用 Linux 和内核功能（例如 Cgroups 和 namespaces来分隔进程，以便各进程相互独立运行。这种独立性正是采用容器的目的所在；它可以独立运行多种进程、多个应用，更加充分地发挥基础设施的作用，同时保持各个独立系统的[安全性](https://www.redhat.com/zh/topics/security)。容器工具（包括 Docker ）可提供基于镜像的部署模式。这使得它能够轻松跨多种环境，与其依赖程序共享应用或服务组。Docker 还可在这一容器环境中自动部署应用（或者合并多种流程，以构建单个应用）。此外，由于这些工具基于 Linux 容器构建，使得 Docker 既易于使用，又别具一格——它可为用户提供前所未有的高度应用程访问权限、快速部署以及版本控制和分发能力。

1. Github 学习

GitHub是最大的开源代码托管平台，旨在促进在一个共同项目上工作的个人之间的代码托管、版本控制和协作。 通过该平台，无论何时何地，都可以对项目进行操作（托管和审查代码，管理项目和与世界各地的其他开发者共同开发软件），GitHub 平台为开源项目和私人项目都提供了项目处理功能。

1. logging模块

logging模块是Python内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志文件回滚等；相比print，具备如下优点：

1. 可以通过设置不同的日志等级，在release版本中只输出重要信息，而不必显示大量的调试信息；
2. print将所有信息都输出到标准输出中，严重影响开发者从标准输出中查看其它数据； logging 则可以由开发者决定将信息输出到什么地方，以及怎么输出。

因此我们采用 logging 模块而不是单纯进行 print 操作。

# 第二章 存在问题与解决方案

1. 存在的主要问题
2. Python实现线程池遭遇的性能瓶颈问题

GIL 是流程的 CPython 解释器（平常称为 Python）中的一个技术术语，即全局解释器锁，其本质上类似操作系统的 Mutex。GIL 的功能是：在 CPython 解释器中执行的每一个 Python 线程，都会先锁住自己，以阻止别的线程执行。

这导致了解释器被一个全局解释器锁保护着，它确保任何时候都只有一个Python线程执行。

GIL会影响到那些严重依赖CPU的程序（比如计算型密集的程序）。GIL会产生一些问题，因为如果一个线程长期持有GIL的话会导致其他非CPU型线程一直等待。

1. 前后端
2. 来自服务端的请求数量可能庞大，需要将请求Promise化，异步处理数据，减少页面阻塞；
3. 如何有效地可视化当前网段存在的服务器进程的分布情况以及实时数据的展示；
4. 具有多个服务器的局域网层级结构的展示方式如何设计，以及如何优化数据结构；
5. 前后端交互时需要拉通对齐JSON格式，规范RESTFUL。
6. 代码快速部署问题

我们希望实现的该基于Client/Server结构的多进程调度系统可能会需要部署在多个服务器，由于服务器间的配置存在差异，指令集，环境等，我们想要使得我们的代码具有可移植性，避免多次调试修改，因此我们想要实现一个可以快速构建生成的运行环境。

我们选择Docker ，将应用程序与基础架构分开，从而可以快速部署软件。借助 Docker，可以与管理应用程序相同的方式来管理基础架构。通过利用 Docker 的方法来快速交付，测试和部署代码，您可以大大减少编写代码和在生产环境中运行代码之间的延迟。

1. 解决方案
2. 进程池效率瓶颈方面
3. 为了避免GIL带来的多线程执行时伪并发问题，我们采取的解决方法是：  
   当一个线程想要执行CPU密集型工作时，会将任务发给进程池。 然后进程池会在另外一个进程中启动一个单独的Python解释器来工作。 当线程等待结果的时候会释放GIL。 并且，由于计算任务在单独解释器中执行，那么就不会受限于GIL了。 在一个多核系统上面，这个技术可以利用多CPU的优势；
4. 同时，我们也采取了C扩展编程技术。 主要思想是将计算密集型任务转移给C，跟Python独立，在工作的时候在C代码中释放GIL。
5. 前后端
6. 精简需求，优化页面渲染，采用Vue等框架，实现对后台的请求到的数据进行异步处理，减少阻塞时间；
7. 优化接口，异步进行数据处理和响应探测结果，实现非阻塞程序，降低系统响应时间；
8. 保持沟通，约定好接口并使用POSTMAN等接口测试工具测试后端接口正确性
9. 代码快速部署问题

使用Docker， 完成环境搭建时，通过Dockerfile构建镜像，将已经完成的应用程序部署在镜像中并发布。在其它服务器主机部署时，直接拉取并构建镜像，通过镜像生成容器并运行，省去了在各服务器上重新配置环境与构建部署代码。容器的使用也保证了持续集成和持续交付（CI / CD）工作流程。

开发的小组人员在本地编写代码，并使用 Docker 容器与小组成员共享他们的工作。使用 Docker将其应用程序推送到测试环境中，并执行自动或手动测试。当开发人员发现错误时，他们可以在开发环境中对其进行修复，然后将其重新部署到测试环境中，以进行测试和验证。

# 第三章 前期任务完成度与后续实施计划

目前张运宸和石佳一已经基本完成了课题任务书中Server端程序的功能，并做了单元测试。Server端程序能够正常响应多个Client端的请求。与Server端并行的还有报警线程和监控线程。当服务进程不足且进程池中有阻塞的进程时，Server端的监控线程能够线性的启动进程池中的进程，子进程能够自动打开服务线程，从而能够服务更多的进程；当服务进程不足且进程池中没有阻塞的进程时，报警线程将会给出告警信息。后续将进行后端程序的API封装，为前端提供可靠的API接口。除此之外，完善和改进后端程序的，提高程序的鲁棒性。

彭冲和卞加勉已经完成了前端界面的设计，菜单的跳转逻辑，后续将不断完善代码，并与后端同学制定接口设计规则，完成整个软件系统。夏野和钟镇阳同学完成了此次报告的编写，并在不断的跟进项目，编写了用于软件测试的工具，节省了单元测试和后端压力测试的时间。

# 第四章 前端界面设计与代码实现

一、前端界面设计

二、前端代码实现

在设计前端页面时，我们将我们的小程序一共分为了5个页面，分别是：

1. display页面：实时展示当前运行程序的总体情况：一共有多少线程、一共有多少进程

2. history页面：用于展示我们对于线程分配修改的简单历史记录

3. historyDetail 页面：用于展示我们对于线程分配修改的详细历史记录

4. index页面：主界面，用于设置你想设置的线程数以及进程数。并且你也可以通过这个页面跳转到其他别的页面之中去。

5．record页面：点击更改进入调整参数的界面.

A picture containing table

Description automatically generated

图4.1 前端页面代码目录

4.2.1 common代码实现

common目录下代码：

Graphical user interface, text

Description automatically generated

图4.2 common代码目录详情

在common目录中只有一个文件`weui.wxss`这个目录中主要存放的是一些常用的wxss的类。这样使用单独的一个包来管理便于其他包的引用，以及管理各个类的指标。

4.2.2 display代码实现

display目录下代码：

Text

Description automatically generated

图4.3 display代码目录详情

4.2.3 history代码实现

history目录下代码：

Graphical user interface, text, application, chat or text message

Description automatically generated

图4.3 history代码目录详情

由于我们已经在前面的章节详细解释了我们对于小程序页面的设计，所以我们这里抛开`history.wxml`以及`history.wxss`两个页面设计的文件。重点介绍该页面逻辑交互部分的代码`history.js`

在`history.js`文件中主要实现了几个函数：

1. onLoad: 页面初始化

2. onShow: 展示我们所得到的历史数据

3. historyToggle：触发是否展示我们的历史记录

4. recordStart：准备开始记录我们的修改记录

5. recordMove：随着我们记录的增加修改、移动我们的显示

4.2.4 historyDetail代码实现

historyDetail目录下代码：

Text

Description automatically generated

图4.4. historyDetail代码目录详情

由于我们已经在前面的章节详细解释了我们对于小程序页面的设计，所以我们这里抛开`historyDetail.wxml`以及`historyDetail.wxss`两个页面设计的文件。重点介绍该页面逻辑交互部分`historyDetail.js`的代码：

1. onLoad：页面初始化

2. recordStart：准备开始记录我们的修改记录

3. recordMove：随着我们记录的增加修改、移动我们的显示

4. recordEnd：如果记录数量显示到一定数目，则不显示最早加进来的数据

5. deleteRecordEvent：删除我们想要删除的记录事件

5.2.5 index代码实现

Index目录下代码：

Graphical user interface, application

Description automatically generated

图4.5 index代码目录详情

由于我们已经在前面的章节详细解释了我们对于小程序页面的设计，所以我们这里抛开`index.wxml`以及`index.wxss`两个页面设计的文件。重点介绍该页面逻辑交互部分`index.js`的代码：

1. onLoad：页面初始化

2. onShow：这里我们使用了POST方法，来获取我们想要的信息。并集中用于展示我们的相关信息。

5.2.6 record代码实现

record目录下代码：

Graphical user interface, text, application

Description automatically generated with medium confidence

图4.6 record目录代码详情

由于我们已经在前面的章节详细解释了我们对于小程序页面的设计，所以我们这里抛开`record.wxml`以及`record.wxss`两个页面设计的文件。重点介绍该页面逻辑交互部分`record.js`的代码：

大部分和前面代码功能类似，这里不再赘述。