<SJTU>

<交我赚——基金分析系统>
软件架构文档

版本 <1.1>

修订历史记录

日期	版本	说明	作者
<20/07/21>	<1.0>	<修订第一版基金分析系统软件架构文档>	<李逸岩，黄喆敏，汪逊杰，钱博闻>
<22/07/21>	<1.1>	<修改用例视图>	<黄喆敏>

目录

软件架构文档

1. 简介

1.1 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面做出的重要决策。

本文档面向软件开发者，在编码前确定软件的逻辑视图，架构视图，部署视图，实现试图和技术视图。同时给出数据视图和关键核心算法设计。

1.2 参考资料

[1] 沈备军，陈昊鹏，陈雨亭. 软件工程原理[M]. 高等教育出版社，2013.

[2] 徐嘉晨. “互联网+”背景下传统券商 APP 理财模块设计分析[J]. 无线互联科技，2020，017（004）：58-59.

[3] 吴志远. 需求引导的移动理财型 APP 交互设计研究[J]. 2017.

2. 用例视图

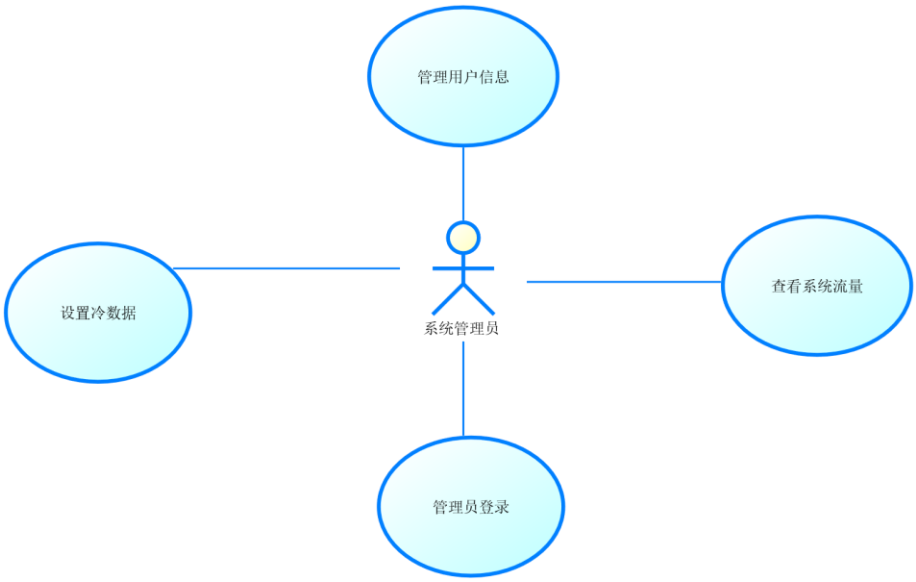


fig 1 系统管理员用例视图

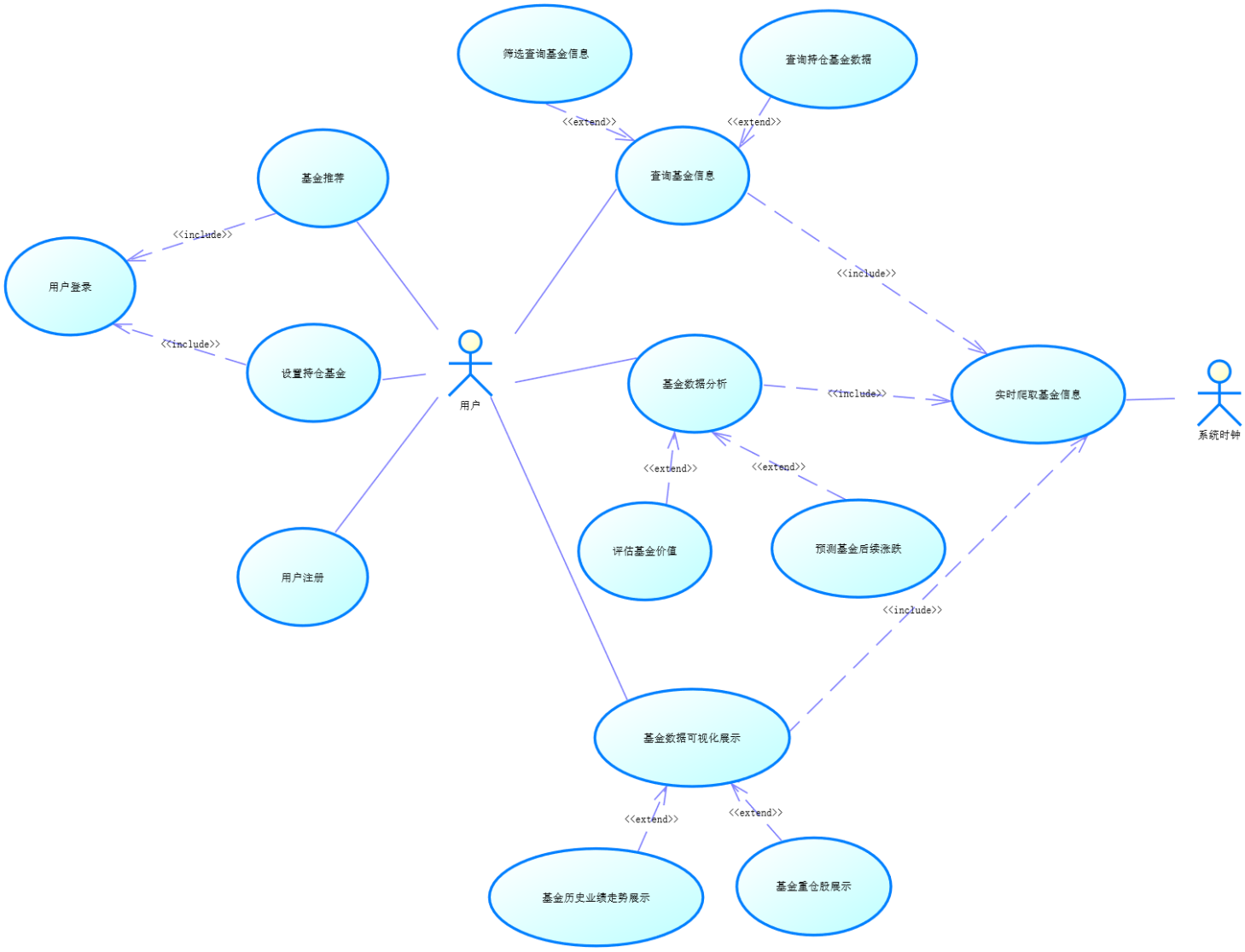


fig 2 用户用例视图

3. 逻辑视图

3.1 概述

采用层次架构（layered architecture）设计。

应用层提供了查看基金推荐、查看持仓基金、设置持仓基金、查询基金数据等功能。其中，查询基金数据包含查询历史净值、根据时间（过去 3 个月、6 个月等）查询基金排名、查询基金对应的基金经理、查询基金重仓股等。

特定业务层提供用户信息管理、冷数据管理、用户管理、基金数据爬取等功能。用户管理指用户更改自己的头像，电子邮箱，用户名，密码等；冷数据管理指管理员更改数据库中已有的信息；

用户管理指管理员对用户权限等进行管理；基金数据爬取指 Python 脚本**定时**从网站上爬取基金最新的信息，并更新数据库。

中间件层采用 Apache Tomcat，连接系统软件层与应用层。系统通过 Tomcat 访问本地的 MySQL 数据库；爬虫程序采用 MySQL-Python 连接脚本与数据库。

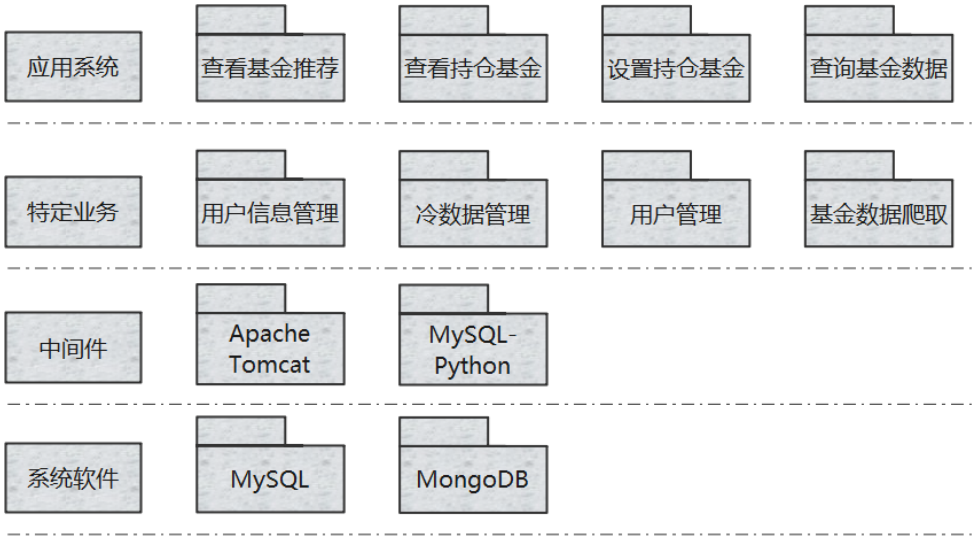


fig 3 **逻辑视图**

3.2 在构架方面具有重要意义的设计包

3.2.1 Config 包

Config 包储存各种配置，包括系统拦截器配置，Spring Security 配置，跨域配置，URL 前缀配置等。

3.2.2 Utils 包

Utils 包由各种工具类组成，如 Unix 时间戳的转换，基金信息转换等。

3.2.3 Entity 包

Entity 包储存 OR 映射的实体类。

3.2.4 Service 包

Service 包实现系统功能抽象得到的 Service 类，实现主要的业务逻辑。如用户登录，查询基金信息等。

3.2.5 DAO 包

DAO 包为数据库访问层，负责与不同的数据库交互。DAO 包分别从 MySQL，MongoDB 中调取信息，并进行合并。

3.2.6 Controller 包

Controller 包定义了各种接口，供前端调用。针对用户客户端与管理员客户端，定义了相应的接口。

4. 进程视图

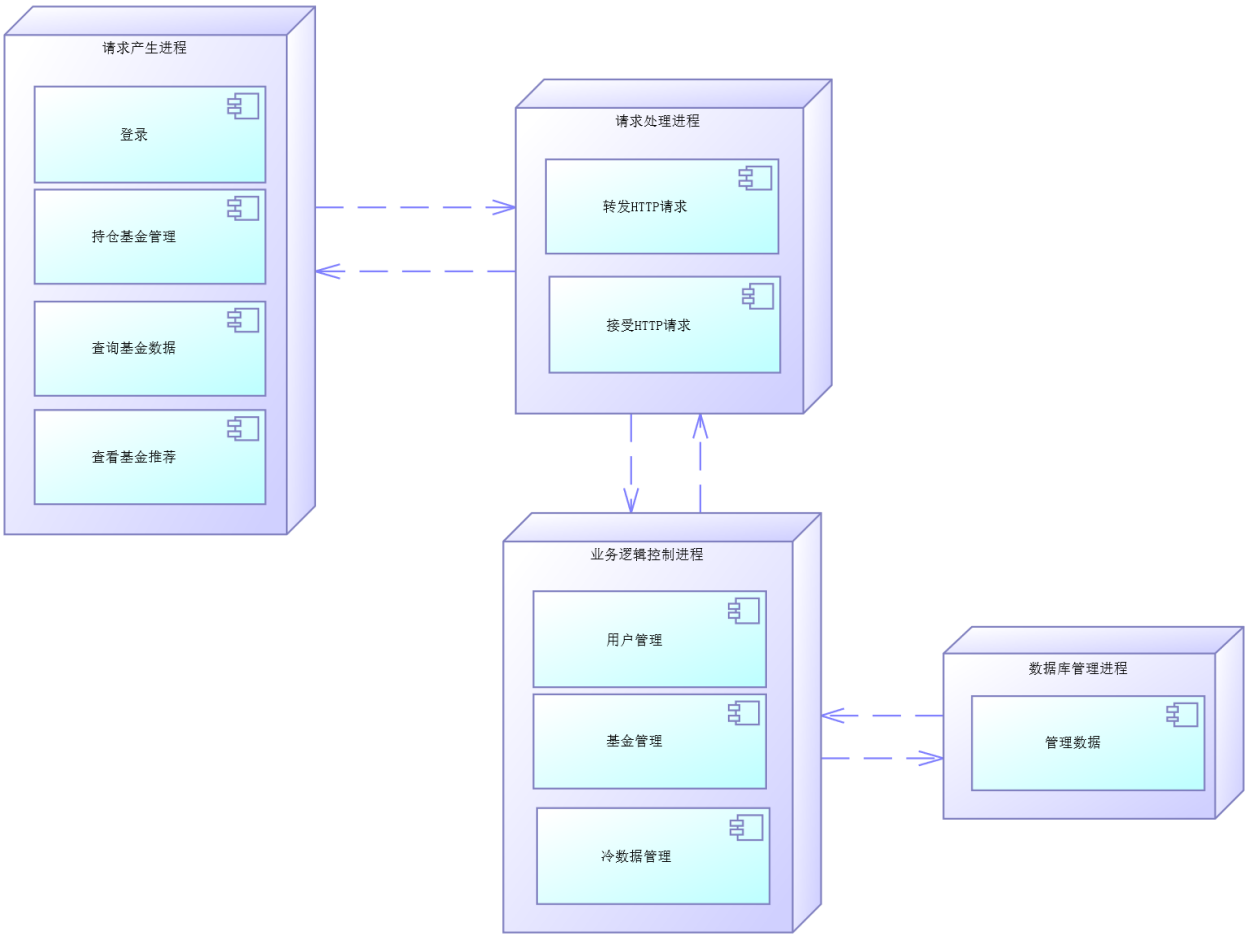
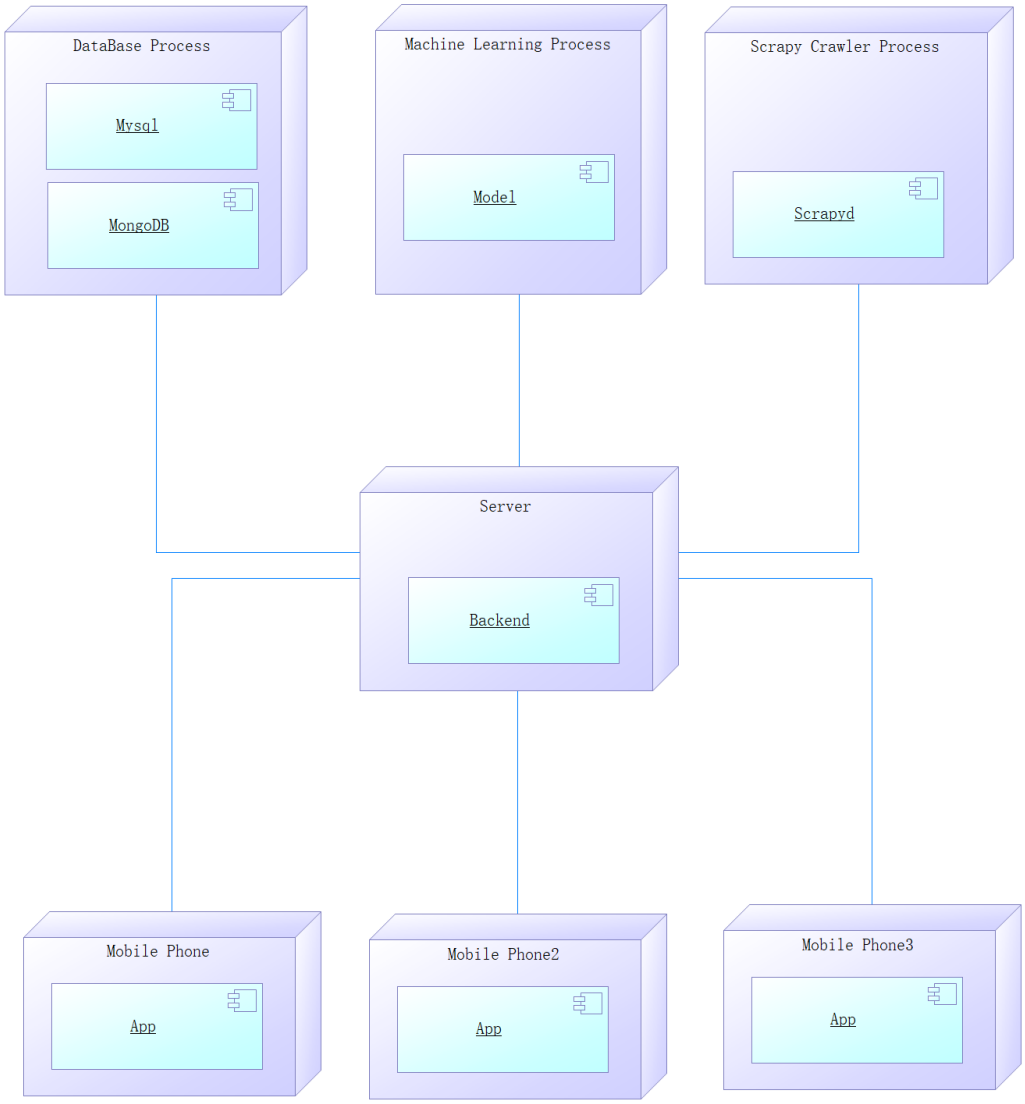


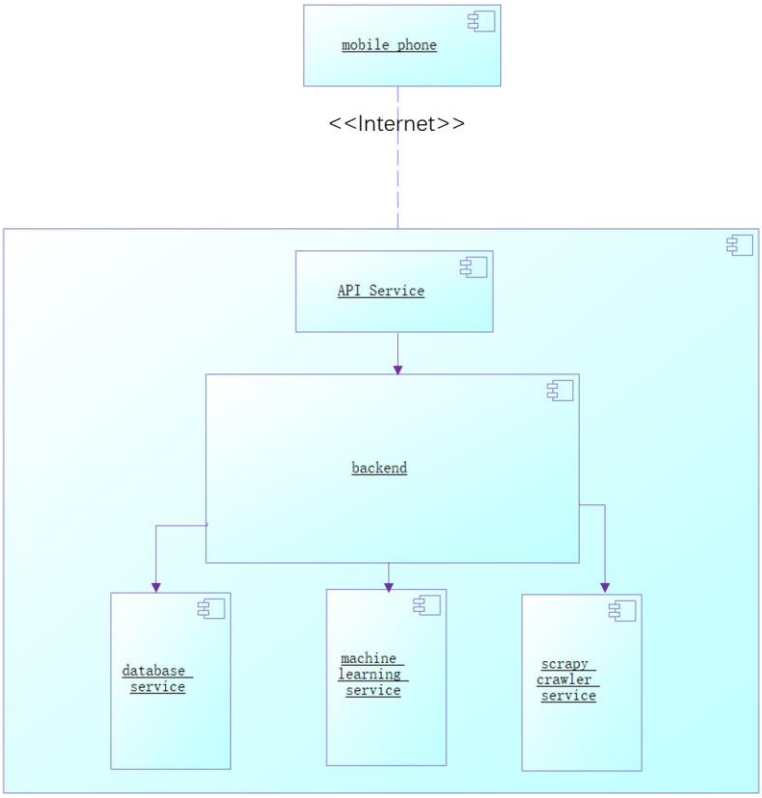
fig 4 进程视图

5. 部署视图

用户通过 App 访问服务器，后端程序部署在多台服务器上，这可以确保在一台服务器宕机时，有其他服务器提供服务。在每一台服务器上，都部署了机器学习进程、爬虫进程和数据库系统，他们与主要的后端程序通过一些协议进行交互调用。



6. 实现视图


mobile phone
<<Internet>>
API Service
backend
database service
machine learning service
scrapy crawler service

7. 技术视图

7.1 操作系统：Windows、Mac OS

7.2 数据库：MySQL、MongoDB

7.3 App 端技术栈：

7.3.1 编程语言：JavaScript、Java

7.3.2 前端框架：React-native

7.3.3 开发工具：IntelliJ IDEA、Android Studio、WebStorm、Git 等

7.3.4 测试工具：Appium

7.4 Server 端技术栈：

7.4.1 编程语言：Java、Python

7.4.2 后端框架：Spring Boot

7.4.3 爬虫框架：Scrapy

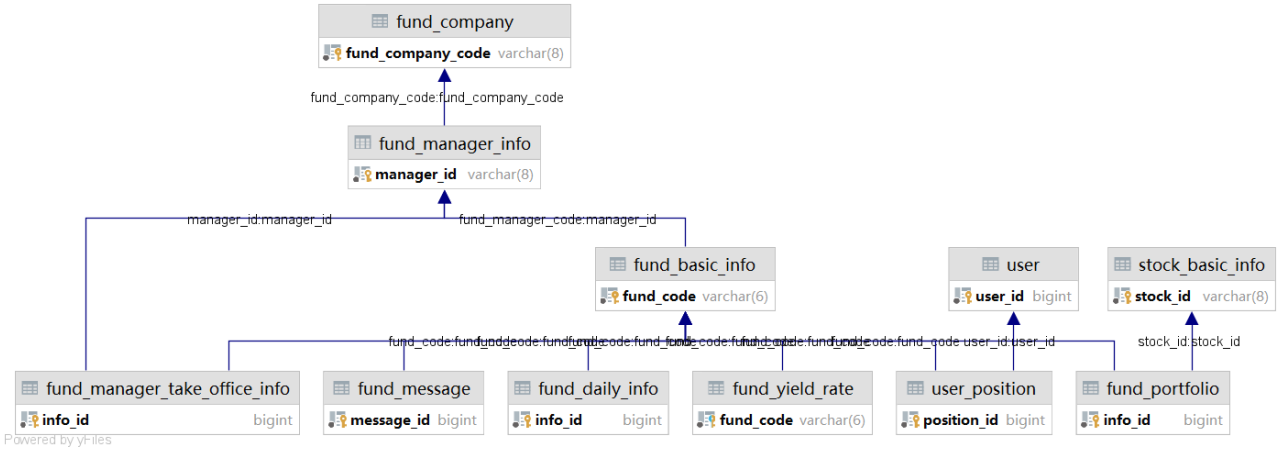
7.4.4 机器学习框架：RNN(LSTM)，使用 TensorFlow 框架

7.4.5 开发工具：IntelliJ IDEA、PyCharm、VSCode、Git

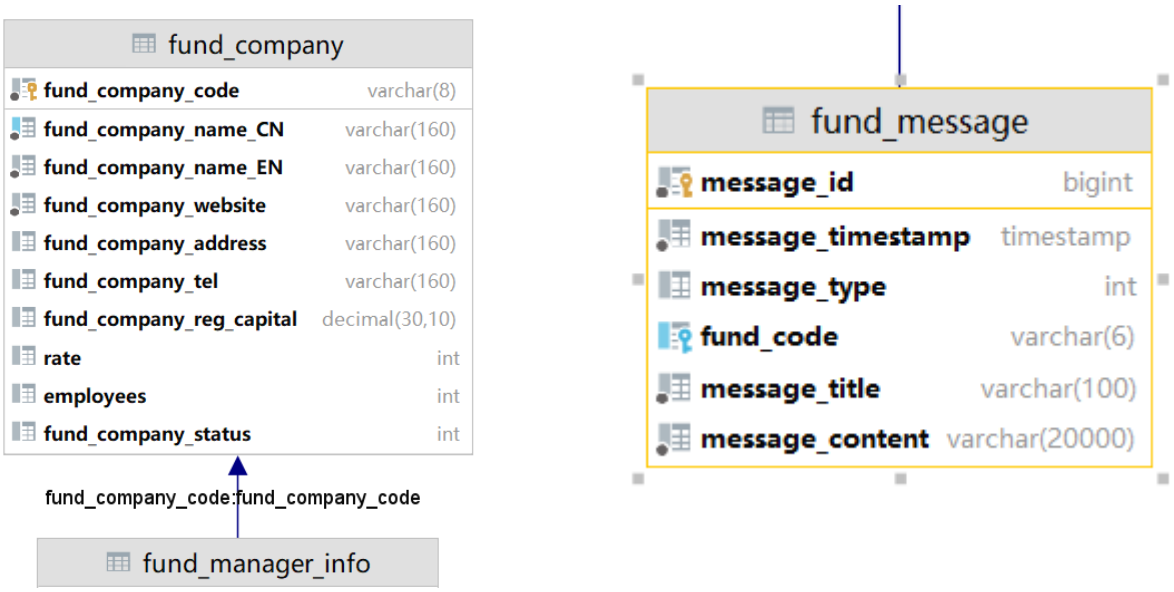
7.4.6 测试工具：Junit、JMeter、TensorBoard

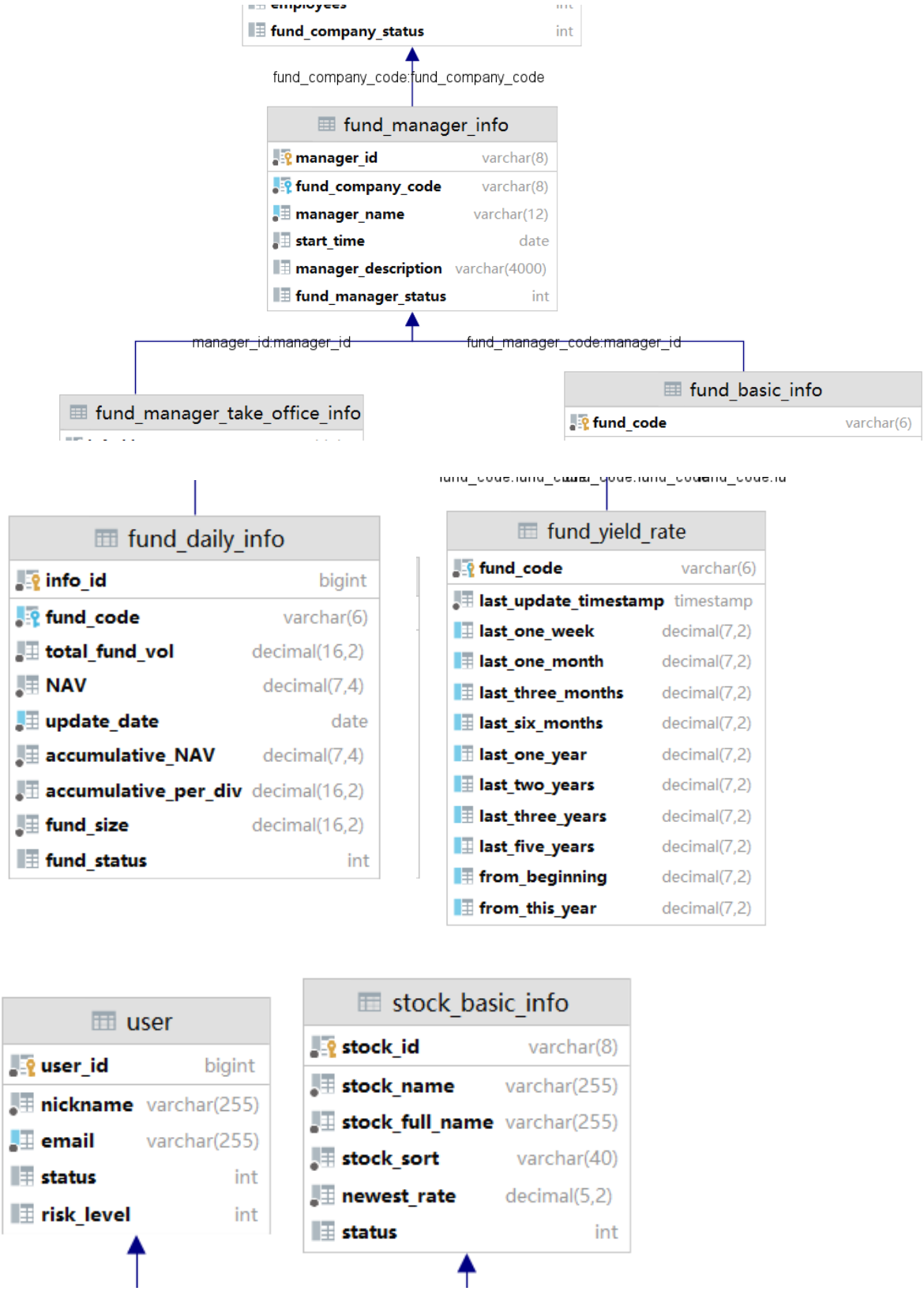
8. 数据视图

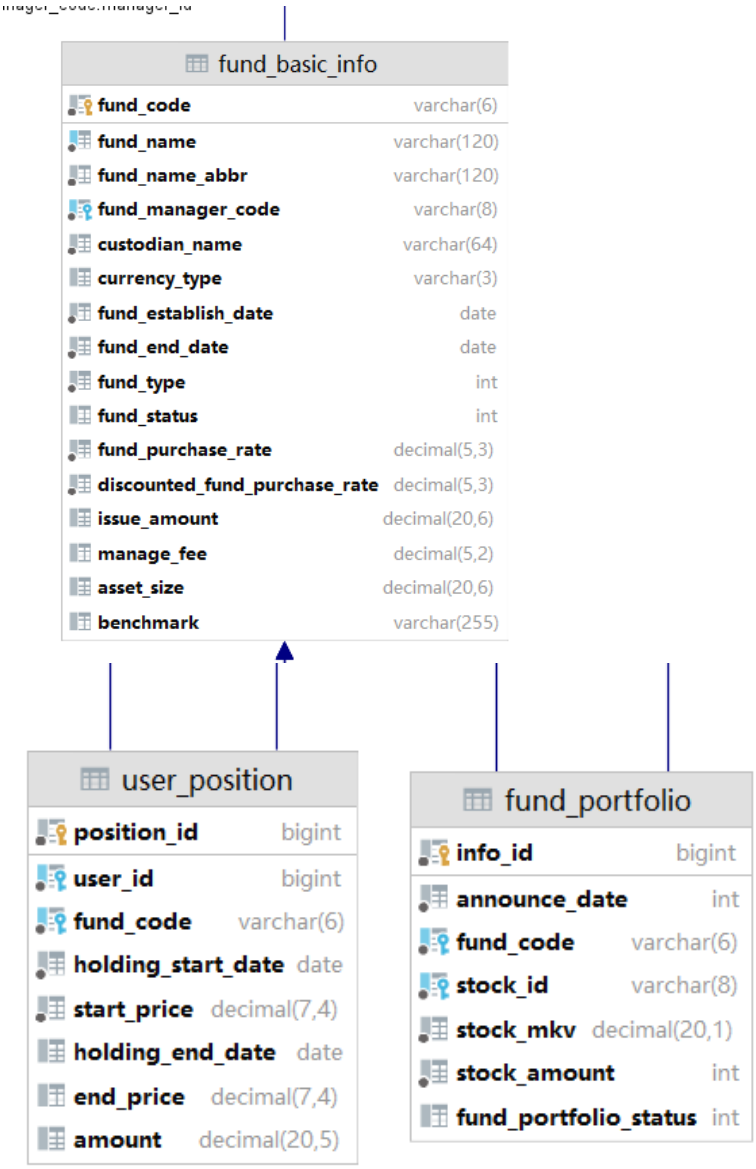
8.1 只含有主键的全局缩略图



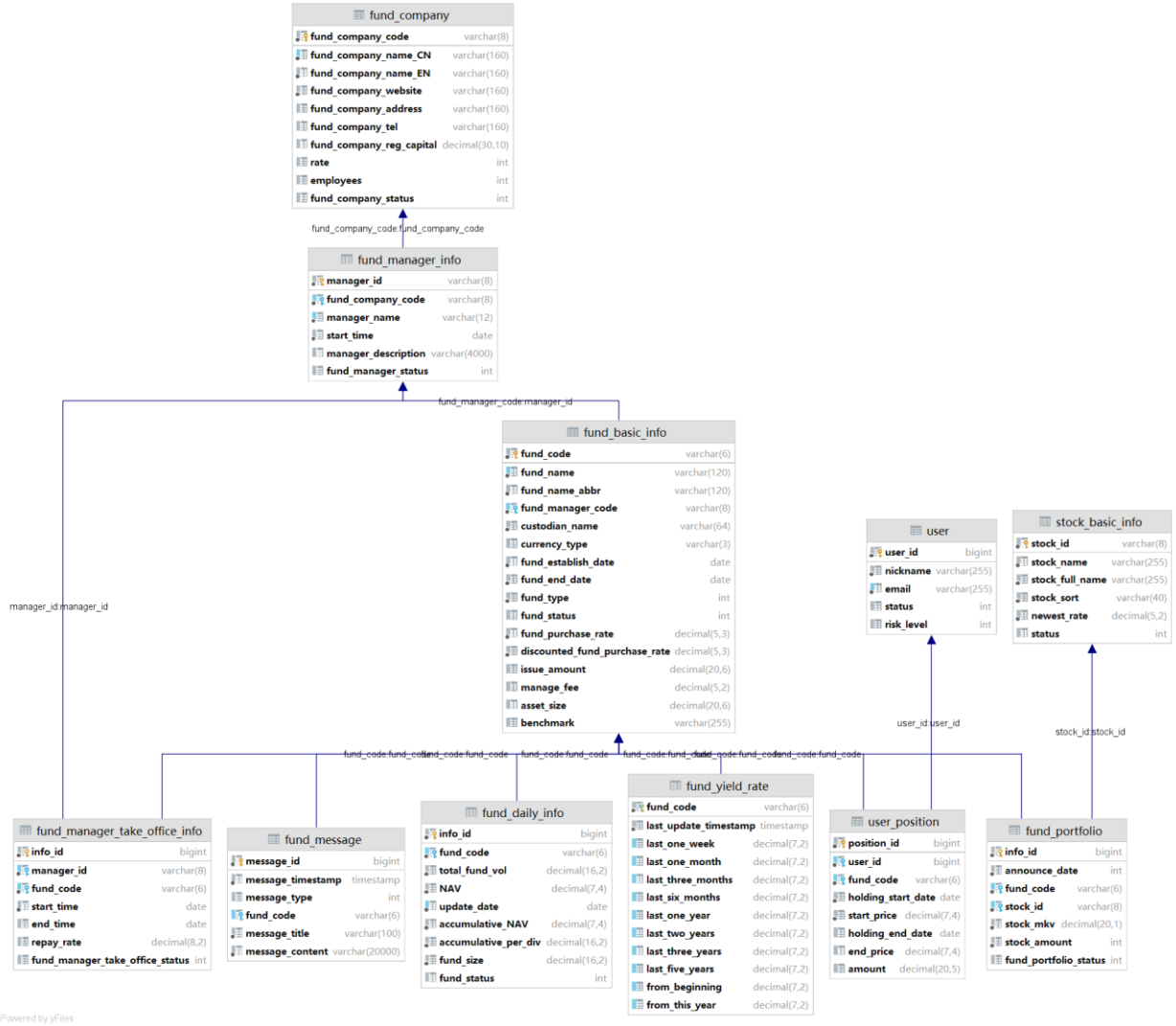
8.2 逐表审看的放大图



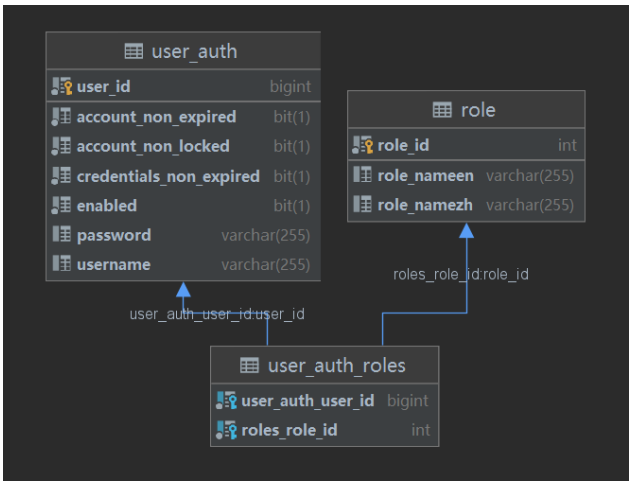

fund_company_status int
fund_company_code:fund_company_code
fund_manager_info
manager_id varchar(8)
fund_company_code varchar(8)
manager_name varchar(12)
start_time date
manager_description varchar(4000)
fund_manager_status int
manager_id:manager_id
fund_manager_code:manager_id
fund_manager_take_office_info
fund_basic_info
fund_code varchar(6)
fund_daily_info
info_id bigint
fund_code varchar(6)
total_fund_vol decimal(16,2)
NAV decimal(7,4)
update_date date
accumulative_NAV decimal(7,4)
accumulative_per_div decimal(16,2)
fund_size decimal(16,2)
fund_status int
fund_yield_rate
fund_code varchar(6)
last_update_timestamp timestamp
last_one_week decimal(7,2)
last_one_month decimal(7,2)
last_three_months decimal(7,2)
last_six_months decimal(7,2)
last_one_year decimal(7,2)
last_two_years decimal(7,2)
last_three_years decimal(7,2)
last_five_years decimal(7,2)
from_beginning decimal(7,2)
from_this_year decimal(7,2)
user
user_id bigint
nickname varchar(255)
email varchar(255)
status int
risk_level int
stock_basic_info
stock_id varchar(8)
stock_name varchar(255)
stock_full_name varchar(255)
stock_sort varchar(40)
newest_rate decimal(5,2)
status int


fund_basic_info
fund_code varchar(6)
fund_name varchar(120)
fund_name_abbr varchar(120)
fund_manager_code varchar(8)
custodian_name varchar(64)
currency_type varchar(3)
fund_establish_date date
fund_end_date date
fund_type int
fund_status int
fund_purchase_rate decimal(5,3)
discounted_fund_purchase_rate decimal(5,3)
issue_amount decimal(20,6)
manage_fee decimal(5,2)
asset_size decimal(20,6)
benchmark varchar(255)
user_position
position_id bigint
user_id bigint
fund_code varchar(6)
holding_start_date date
start_price decimal(7,4)
holding_end_date date
end_price decimal(7,4)
amount decimal(20,5)
fund_portfolio
info_id bigint
announce_date int
fund_code varchar(6)
stock_id varchar(8)
stock_mkv decimal(20,1)
stock_amount int
fund_portfolio_status int

8.3 全局图



8.4 spring security 数据库视图



8.5 mongo DB 数据视图


```
_id: ObjectId("60f66667926d6ecd08af0c78")
manager_id: "30282105"
avatar: Binary('LzlqLzRBQVFTa1pKUmdBQkFRQUFBUUFCQUFELzJ3QkRBQWdHQmdjR0JRZ0hCd2NKQ1FnS0RCUU5EQXNMREJrU0V3OFVIUm9mSGgw...', 0)
```

```
_id: ObjectId("60f66667926d6ecd08af0c79")
manager_id: "30275466"
avatar: Binary('LzlqLzRBQVFTa1pKUmdBQkFRQUFBUUFCQUFELzJ3QkRBQWdHQmdjR0JRZ0hCd2NKQ1FnS0RCUU5EQXNMREJrU0V3OFVIUm9mSGgw...', 0)
```


9. 核心算法设计

9.1 深度学习算法

循环神经网络（Recurrent neural network：RNN）是神经网络的一种例子，由于其在序列化数据处理相较于常见的 SVM 分类器，决策树等具有天然的巨大优势，我们采用 RNN 来进行基金走势预测。但是，单纯的 RNN 因为无法处理随着递归，权重指数级爆炸或梯度消失问题，难以捕捉长期时间关联；而结合不同的 LSTM 可以很好解决这个问题。在某些节点上还有可能使用 GRU 来代替 LSTM 节点，这将根据具体的训练结果来决定。

$$
\begin{aligned}
i_t &= \tanh(W_{\mathrm{xi}}x_t + W_{\mathrm{hi}}h_{t-1} + b_{\mathrm{i}}) \\
j_t &= \mathrm{sigm}(W_{\mathrm{xj}}x_t + W_{\mathrm{hj}}h_{t-1} + b_{\mathrm{j}}) \\
f_t &= \mathrm{sigm}(W_{\mathrm{xf}}x_t + W_{\mathrm{hf}}h_{t-1} + b_{\mathrm{f}}) \\
o_t &= \tanh(W_{\mathrm{xo}}x_t + W_{\mathrm{ho}}h_{t-1} + b_{\mathrm{o}}) \\
c_t &= c_{t-1} \odot f_t + i_t \odot j_t \\
h_t &= \tanh(c_t) \odot o_t
\end{aligned}
$$

fig 5 LSTM – 简单迭代计算式

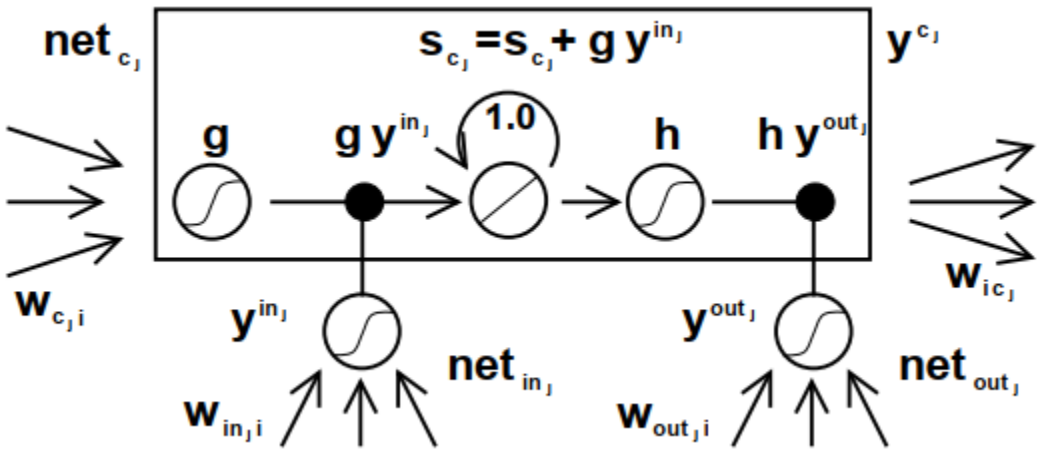


fig 6 LSTM 图示结构

10. 质量属性的设计

10.1 安全性

10.1.1 原因：

本系统的用户持仓基金功能只能为有权限的已登录普通用户提供服务，用户管理等管理员功能只能为有权限的已登录管理员用户提供服务，由于涉及个人信息，需要保证用户权限的安全性。

10.1.2 实现

后端使用 Spring Security 实现用户注册登录、用户授权等重要安全功能，实现后端接收信息的过滤。对于用户密码进行加密，而并非明文传输以应对数据库泄露等极端情况；此外严格权限管理，对所有能访问，操作数据库的请求都显式的要求权限，不获得后端 spring security 框架的权限认证不可能访问到数据库。

10.2 可修改性

10.2.1 原因

由于整个软件开发过程需要经历生产环境、测试环境、发布环境等，数据库、爬虫等模块的部署要具有可修改性。

10.2.2 实现

对这些模块的信息编码到配置文件中，而并非硬编码到模块内部，当环境发生改变时，只需要修改配置文件即可成功部署。

10.3 易用性

10.3.1 原因

基金分析系统面向的对象范围广，对于各类人群都需要较好的可用性。

10.3.2 实现

设计用户交互界面简洁并符合基本使用习惯，设置帮助手册；对于“没有数据”，“数据正在加载中”的情况必须显式的告诉用户，对于正在进行计算或数据加载，应该提供可靠的进度条，不能卡在某个地方或

者卡在 99%这样不可靠的情况出现。

10.4 性能

10.4.1 原因

软件要求在 2000 并发的场景下，保证用户体验。

10.4.2 实现

对于“一段时间内基金净值排名”等类似信息的排序，在数据库中建立索引，以达到快速排序。建立缓存表。服务器端负载均衡

10.5 可测试性

10.5.1 原因

保证系统的质量需要有足够的测试支撑。

10.5.2 实现

前端要分 View 和 Component，测试时对于每个 View 和 Component 进行测试。后端分 Controller, Service, DAO, Repository 等数层，测试时对每层分别测试，提高测试效率和准确度。

10.6 稳定性

10.6.1 原因

在任何情况下都不能让 App 端用户看到无法理解的错误，因为对于外行型用户来说，看到带有英文的报错信息是让人惊慌的，不可接受的。

10.6.2 实现

对于可以预估的错误应该制定错误跳转页面，如“权限不足”等页面，并显式的告诉用户错误发生原因，对于不可预估的错误，也应编写兜底的页面，提醒用户发生错误和可能的解决方案。