

🏠 / 开发指南 / 后端手册

👤 芋道源码 📅 2022-03-28

🔍 操作日志、访问日志、异常日志

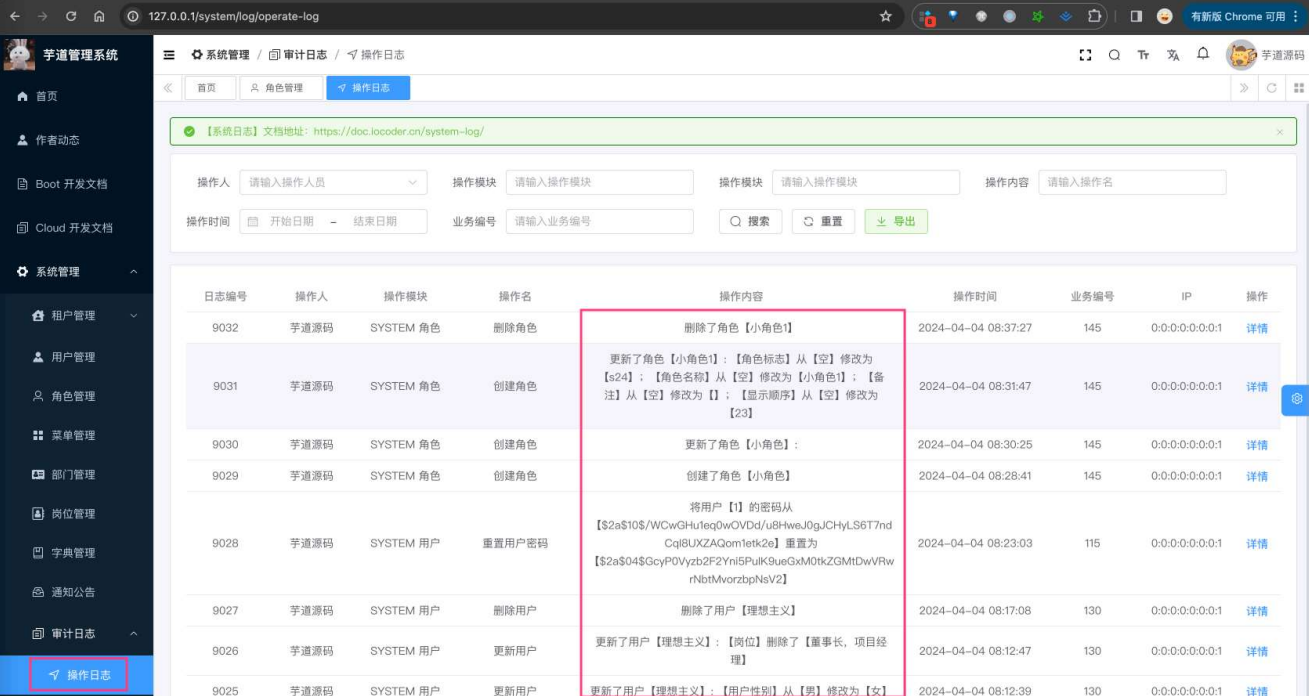
项目提供 2 类 4 种系统日志：

- 审计日志：用户的操作日志、登录日志
- API 日志：RESTful API 的访问日志、错误日志

1. 操作日志

操作日志，记录「谁」在「什么时间」对「什么对象」做了「什么事情」。

打开 [系统管理 -> 审计日志 -> 操作日志] 菜单，可以看到对应的列表，如下图所示：



1.1 操作日志组件

① 操作日志的记录，由 `yudao-spring-boot-starter-security` 技术组件的 `operatelog` 包提供，基于我老友开源的 <https://github.com/mouzt/mzt-biz-log> 实现，只需要添加 `@LogRecord` 注解，即可实现操作日志的记录。

- 小明 增加 自定义属性【分类-交互优化，稳定性/bug】 2021-01-28 21:37:31
 - 小明 优先级由 为空 变更为 中 2021-01-28 21:37:27
 - 小明 更新了 描述 至V1版本（上一版本快照：V0） 2021-01-28 21:37:02
- 【新增】2021-09-16 10:00 订单创建，订单号：NO.11089999，其中涉及变量订单号“NO.11089999”

- **【修改】** 2021-09-16 10:00 用户小明修改了订单的配送地址：从“金灿灿小区”修改到“银盏盏小区”

疑问：为什么不独立一个 `yudao-spring-boot-starter-operatelog` 组件呢？

早期，项目确实有 `operatelog` 组件，和 `mzt-biz-log` 组件一样，也是基于 Spring AOP + 注解实现。

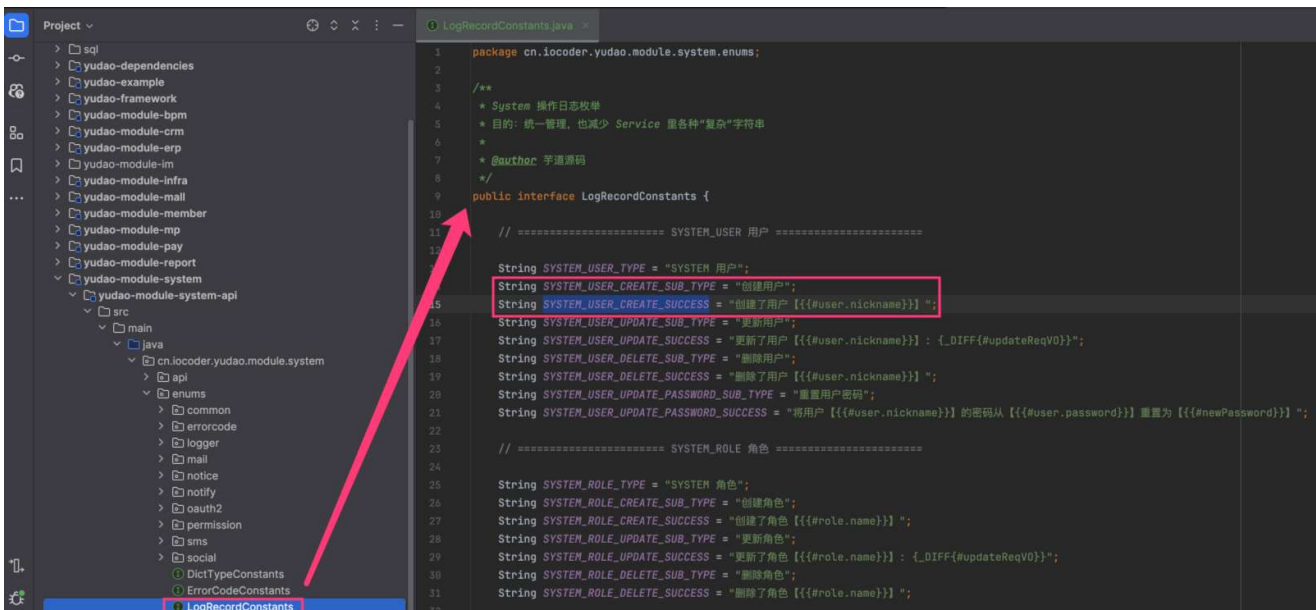
随着我老友开源了 `mzt-biz-log` 组件，再加上希望减少项目的 starter 数量，所以决定直接使用 `mzt-biz-log` 组件，放到 `security` 组件中，而不是自己再造一个轮子。

- ② 操作日志的存储，由 `yudao-module-system` 的 `OperateLog` 模块实现，记录到数据库的 `system_operate_log` 表。

下面，我们来看项目中的几个使用案例。在开始之前，希望你先简单通读下《`mzt-biz-log` 使用指南》文档。

1.2 场景一：创建用户

- ① 在 `LogRecordConstants` 类中，定义 `SYSTEM_USER_CREATE_SUB_TYPE`、`SYSTEM_USER_CREATE_SUCCESS` 变量。如下图所示：



疑问：为什么要在 `LogRecordConstants` 定义？

这个并非强制，只是一个使用建议。每个 `yudao-module-xxx` 模块下，建议都搞一个 `LogRecordConstants` 类，方便大家统一管理操作日志的常量，这样方便大家查找。

- ② 在 Service 方法上，添加 `@LogRecord` 注解，如下图所示：

Project ▾
yudao-module-system-biz
 src
 main
 java
 cn.iocoder.yudao.module.system
 api
 controller
 convert
 dal
 framework
 job
 mq
 service
 auth
 dept
 dict
 errorcode
 logger
 mail
 member
 notice
 notify
 oauth2
 permission
 sensitiveword
 sms
 social
 tenant
 user
 AdminUserService
 AdminUserServiceImpl
 util
 package-info.java

AdminUserServiceImpl.java
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
@Override
@Transactional(rollbackFor = Exception.class)
@LogRecord(type = SYSTEM_USER_TYPE, subType = SYSTEM_USER_CREATE_SUB_TYPE, bizNo = "{{user.id}}",
success = SYSTEM_USER_CREATE_SUCCESS)
public Long createUser(UserSaveReqVO createReqVO) {
 // 1.1 校验账户配合
 tenantService.handleTenantInfo(tenant -> {
 Long count = userMapper.selectCount();
 if (count >= tenant.getAccountCount()) {
 throw exception(USER_COUNT_MAX, tenant.getAccountCount());
 }
 });
 // 1.2 校验正确性
 validateUserForCreateOrUpdate(id: null, createReqVO.getUsername(),
 createReqVO.getMobile(), createReqVO.getEmail(), createReqVO.getDeptId(), createReqVO.getPostIds());
 // 2.1 插入用户
 AdminUserDO user = BeanUtils.toBean(createReqVO, AdminUserDO.class);
 user.setStatus(CommonStatusEnum.ENABLE.getStatus()); // 默认开启
 user.setPassword(encodePassword(createReqVO.getPassword())); // 加密密码
 userMapper.insert(user);
 // 2.2 插入关联岗位
 if (CollectionUtil.isNotEmpty(user.getPostIds())) {
 userPostMapper.insertBatch(convertList(user.getPostIds(),
 postId -> new UserPostDO().setUserId(user.getId()).setPostId(postId)));
 }
 // 3. 记录操作日志上下文
 LogRecordContext.putVariable(names: "user", user);
 return user.getId();
}

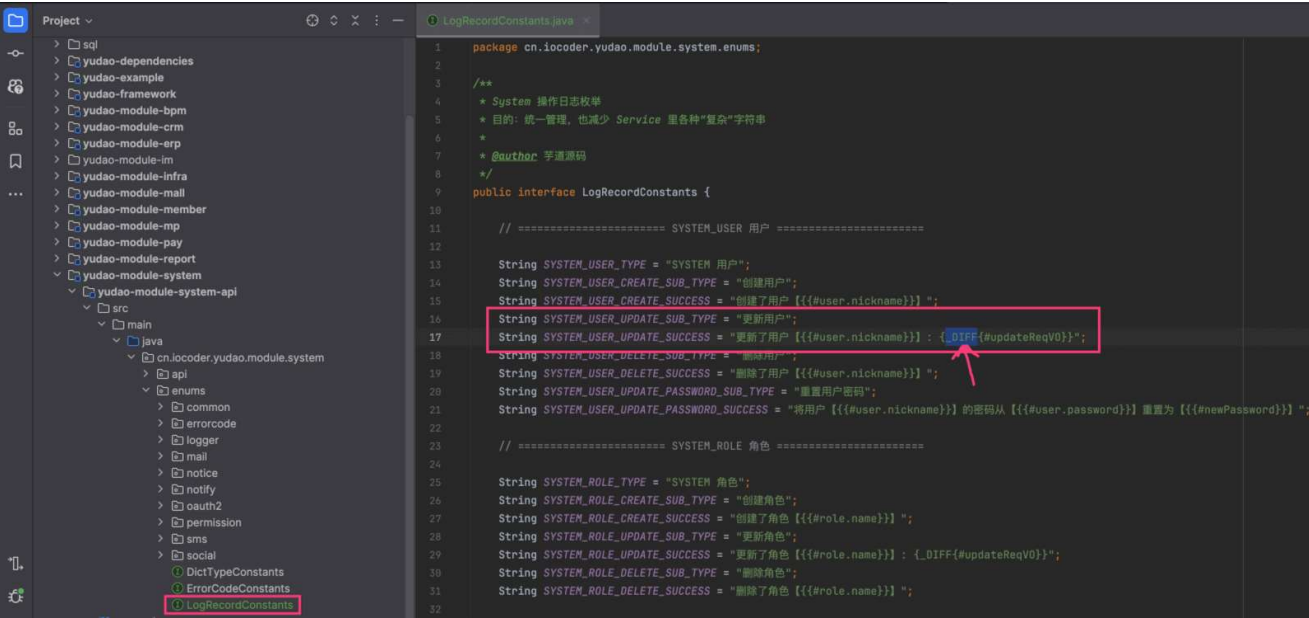
最终，我们记录操作日志的内容，如下图所示：

详情

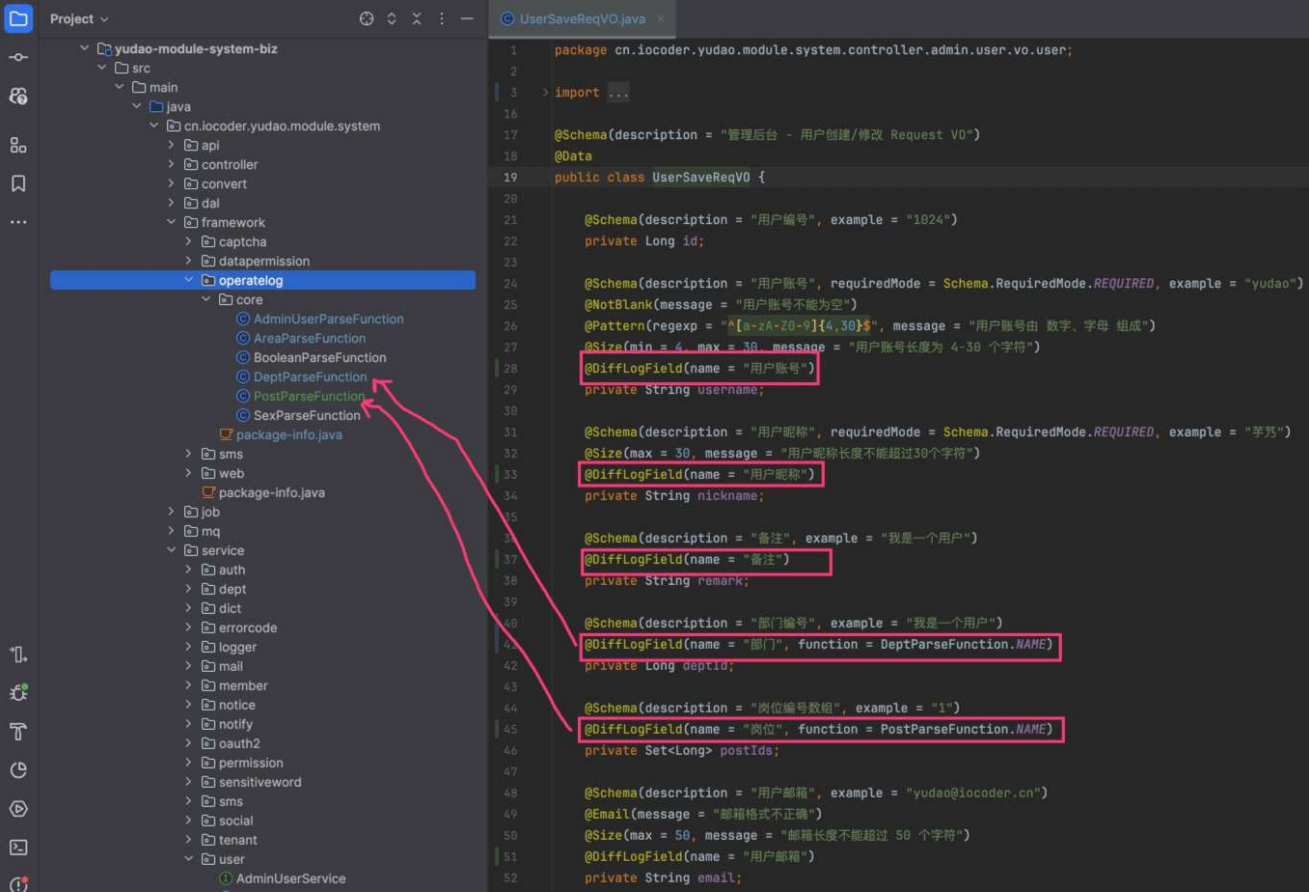
日志主键	9019
操作人编号	1
操作人名字	芋道源码
操作人 IP	0:0:0:0:0:0:1
操作人 UA	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
操作模块	SYSTEM 用户
操作名	创建用户
操作内容	创建了用户【理想主义】
请求 URL	POST /admin-api/system/user/create
操作时间	2024-04-04 07:49:22
业务编号	130

1.3 场景二：修改用户信息

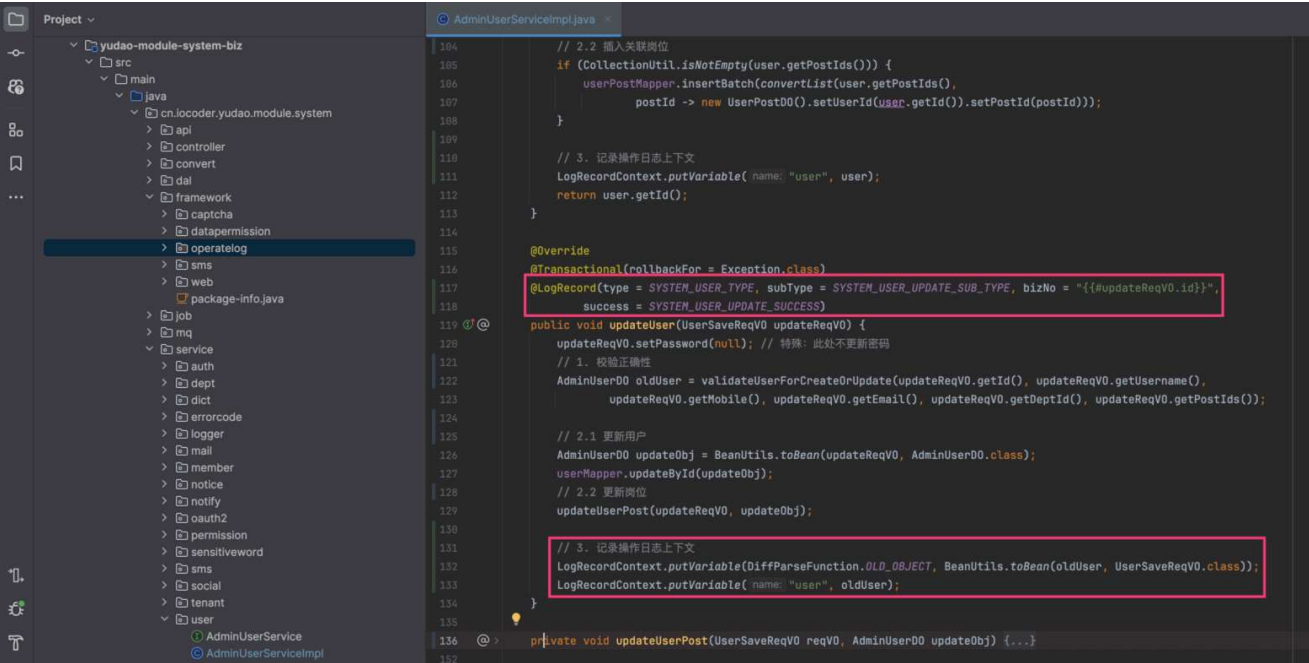
① 在 LogRecordConstants 类中，定义 SYSTEM_USER_UPDATE_SUB_TYPE 、
SYSTEM_USER_UPDATE_SUCCESS 变量。如下图所示：



这里我们使用了 `_DIFF` 函数，实现对象 diff 功能，即“【备注】从【132】修改为【1324】”。因此，我们需要在 `UserSaveReqVO` 类上添加 `@DiffLogField` 注解，如下图所示：



- 注解的 `name` 字段：字段的中文名，例如说：“【备注】”
- 注解的 `function` 字段：自定义函数，用于字段的值翻译，例如说：`PostParseFunction`
 岗位名、`DeptParseFunction` 部门名、`SexParseFunction` 性别等等
- ② 在 Service 方法上，添加 `@LogRecord` 注解，如下图所示：



最终，我们记录操作日志的内容，如下图所示：

详情	
操作人编号	1
操作人名字	芋道源码
操作人 IP	0:0:0:0:0:0:1
操作人 UA	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
操作模块	SYSTEM 用户
操作名	更新用户
操作内容	更新了用户【1】：【部门】从【深圳总公司】修改为【长沙分公司】；【用户邮箱】从【】修改为【7648@qq.com】；【手机号码】从【15601691238】修改为【15601691229】；【用户昵称】从【1】修改为【阿呆】；【岗位】添加了【董事长，项目经理】；【备注】从【11】修改为【11222】；【用户性别】从【男】修改为【女】
请求 URL	PUT /admin-api/system/user/update
操作时间	2024-04-04 09:37:14
业务编号	115

1.4 更多场景

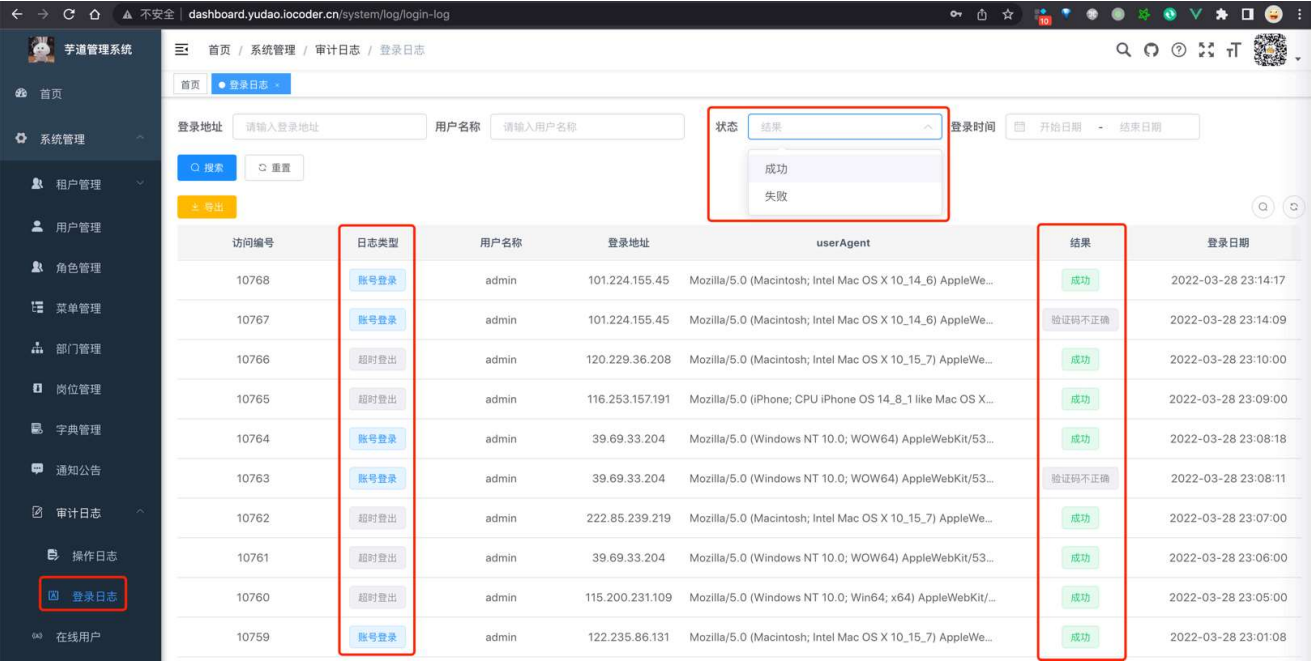
由于操作日志需要手动记录，再加上是新改造的，所以目前只有少数几个地方使用了。后续，我们会逐步完善，让操作日志更加完善。

目前，CRM 模块的操作日志是最全的，应该有大几十个使用案例，大家碰到不会使用的地方，可以参考下。

2. 登录日志

登录日志，记录用户的登录、登出行为，包括成功的、失败的。

打开 [系统管理 -> 审计日志 -> 登录日志] 菜单，可以看对应的列表，如下图所示：



登录日志的存储，由 `yudao-module-system` 的 `LoginLog` 模块实现，记录到数据库的 `system_login_log` 表。

登录类型通过 `LoginLogTypeEnum` 枚举，登录结果通过 `LoginResultEnum` 枚举，都可以自定义。代码如下：

```
6  /**
7   * 登录日志的类型枚举
8   */
9   @Getter
10  @AllArgsConstructor
11  public enum LoginLogTypeEnum {
12
13      LOGIN_USERNAME(100), // 使用账号登录
14      LOGIN_SOCIAL(101), // 使用社交登录
15      LOGIN MOCK(102), // 使用 Mock 登录
16      LOGIN_MOBILE(103), // 使用手机登陆
17      LOGIN_SMS(104), // 使用短信登陆
18
19      LOGOUT_SELF(200), // 自己主动登出
20      LOGOUT_TIMEOUT(201), // 超时登出
21      LOGOUT_DELETE(202), // 强制退出
22  };
23
24  /**
25   * 日志类型
26   */
27  private final Integer type;
28  }
```

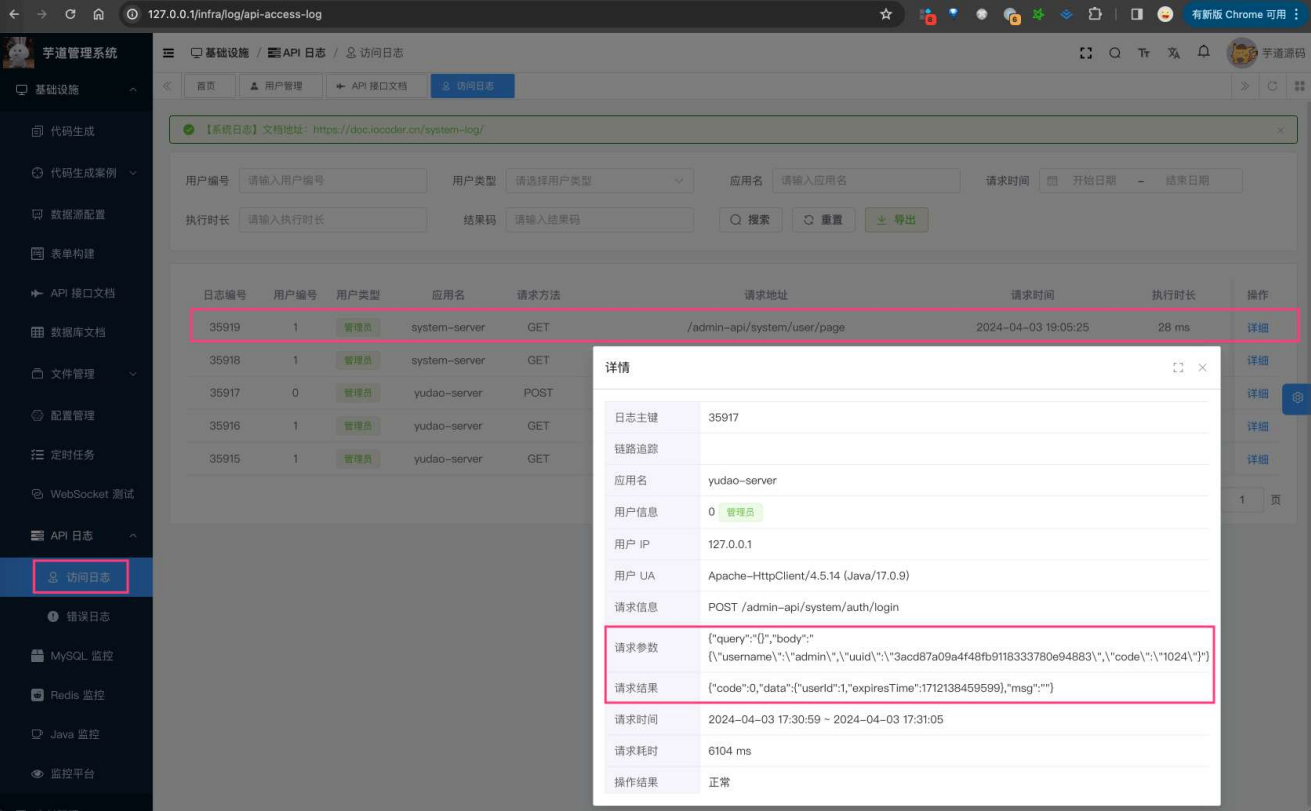
```
6  /**
7   * 登录结果的枚举类
8   */
9   @Getter
10  @AllArgsConstructor
11  public enum LoginResultEnum {
12
13      SUCCESS(0), // 成功
14      BAD_CREDENTIALS(10), // 账号或密码不正确
15      USER_DISABLED(20), // 用户被禁用
16      CAPTCHA_NOT_FOUND(30), // 图片验证码不存在
17      CAPTCHA_CODE_ERROR(31), // 图片验证码不正确
18
19      UNKNOWN_ERROR(100), // 未知异常
20  };
21
22  /**
23   * 结果
24   */
25  private final Integer result;
26
27  }
```

3. API 访问日志

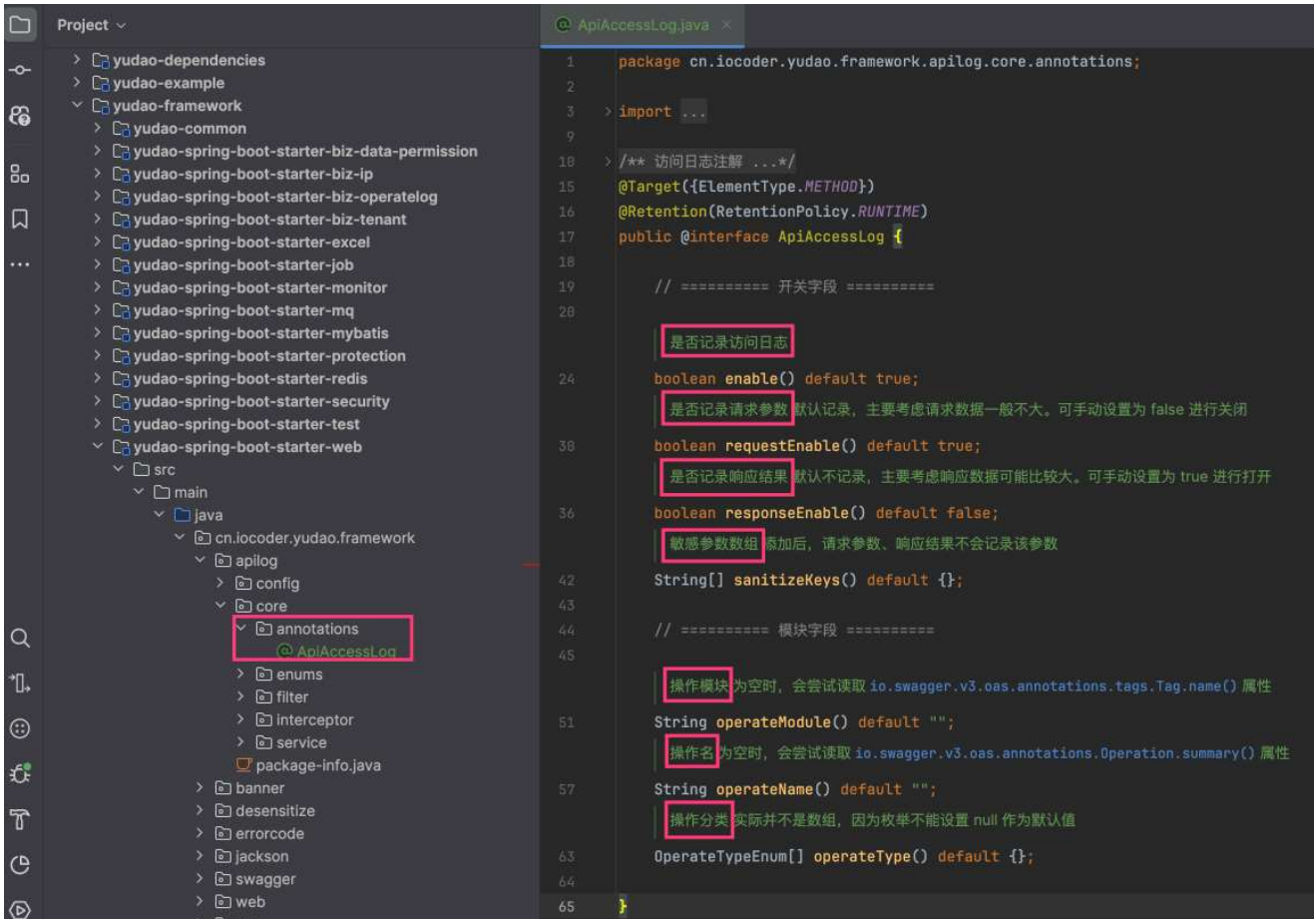
3.1 数据库记录

API 访问日志，记录 API 的每次调用，包括 HTTP 请求、用户、开始时间、时长等等信息。

打开 [基础设施 -> API 日志 -> 访问日志] 菜单，可以看对应的列表，如下图所示：



- ① 访问日志的记录，由 `yudao-spring-boot-starter-web` 技术组件实现，通过 `ApiAccessLogFilter` 过滤 RESTful API 请求，异步记录日志。
- ② 访问日志的存储，由 `yudao-module-infra` 的 `AccessLog` 模块实现，记录到数据库的 `infra_api_access_log` 表。
- ③ 可以通过 `@ApiAccessLog` 注解，自定义 API 访问日志的记录。如下图所示：



- `enable` 字段：是否记录日志，默认为 `true` 记录日志。如果你想某个接口不记录日志，可以设置为 `false`，例如说 `NotifyMessageController` 的 `#getUnreadNotifyMessageCount()` 接口
- `requestEnable` 字段：默认为 `true` 记录请求参数，主要考虑请求参数一般不大。如果你想某个接口不记录请求参数，可以设置为 `false`
- `sanitizeKeys` 字段：脱敏字段，例如说：密码、访问令牌等等。如果你想某个接口脱敏某个字段，可以设置为 `password`、`mobile` 等等。另外，`ApiAccessLogFilter` 默认有 `SANITIZE_KEYS` 全局配置，包括 `password`、`accessToken`、`refreshToken` 等，避免大家忘记~
- `responseEnable` 字段：默认为 `false` 不记录响应参数，主要考虑响应参数一般比较大，特别是 `GET` 列表请求。如果你想某个接口记录响应参数，可以设置为 `true`
- `operateModule` 字段：操作模块，例如说：用户、岗位、部门等等。为空时，默认会读取类上的 Swagger `@Tag` 注解的 `name` 属性
- `operateName` 字段：操作名，例如说：新增用户、修改用户等等。为空时，默认会读取方法的 Swagger `@Operation` 注解的 `summary` 属性
- `operateType` 字段：操作类型，操作类型，在 `OperateTypeEnum` 枚举。目前有 `GET` 查询、`CREATE` 新增、`UPDATE` 修改、`DELETE` 删除、`EXPORT` 导出、`IMPORT` 导入、`OTHER` 其它，可进行自定义

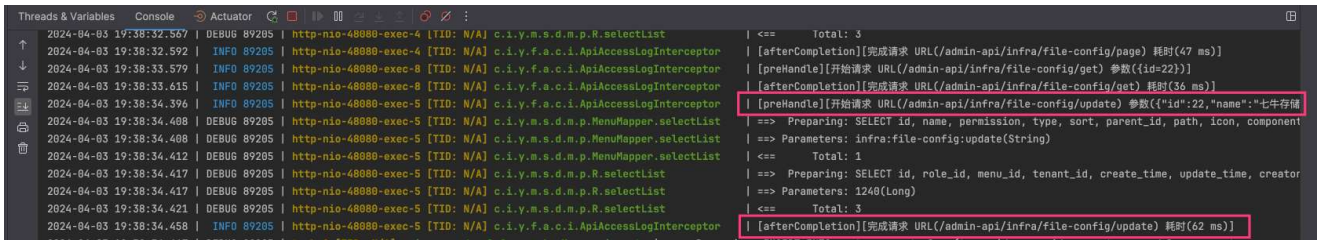
疑问：为什么要增加 ``operateModule``、``operateName``、``operateType`` 字段呢？

从感受上来说，它们应该属于“操作日志”，不应该记录到“访问日志”中。但是考虑到“操作日志”需要手动记录，可能大家会“偷懒”不想记录，但是业务人员（例如说：产品、运行）等看不懂“访问日志”，所以增加了这些字段，方便大家查看。

④ 我们在 `local` 本地环境下，一般做一些日常开发，使用不到“访问日志”，所以默认在 `application-local.yaml` 配置文件里，我们设置 `yudao.access-log.enable` 为 `false` 默认不记录，大家如果有需要，可以设置为 `true` 打开进行记录。

3.2 文件记录

项目还提供了 `ApiAccessLogInterceptor` 拦截器，打印 HTTP 请求、参数、耗时到文件（IDEA 控制台）中，方便大家进行调试。如下图所示：

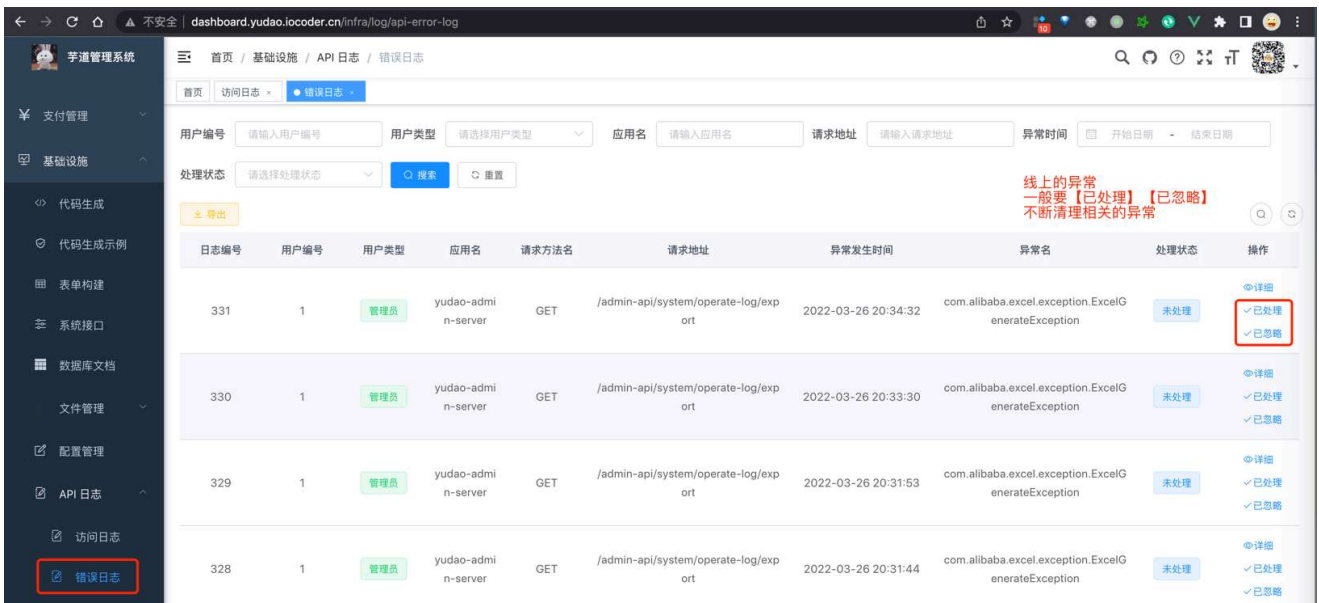


每次请求有两条：一条 `request` 【开始请求】包括请求 URL、请求参数；一条 `response` 【结束请求】只包括耗时。

另外，考虑到 `ApiAccessLogInterceptor` 的定位是开发调试，所以 `prod` 生产环境默认不开启噢，当然你也可以按照自己需要修改。

4. API 错误日志

API 错误日志，记录每次 API 的异常调用，包括 HTTP 请求、用户、异常的堆栈等等信息。打开 [基础设施 -> API 日志 -> 错误日志] 菜单，可以看对应的列表，如下图所示：



错误日志的记录，由 `yudao-spring-boot-starter-web` 技术组件实现，通过 `GlobalExceptionHandler` 拦截每次 RESTful API 的系统异常，异步记录日志。

```
GlobalExceptionHandler.java
187     log.warn("[accessDeniedExceptionHandler][userId({}) 无法访问 url({})]", WebFrameworkUtils.getLoginUserId(req),
188             req.getRequestURL(), ex);
189     return CommonResult.error(FORBIDDEN);
190 }
191
192 /** 处理业务异常 ServiceException ...*/
193 @ExceptionHandler(value = ServiceException.class)
194 public CommonResult<?> serviceExceptionHandler(ServiceException ex) {...}
195
196
197 /**
198  * 处理系统异常，兜底处理所有的一切
199  */
200 @ExceptionHandler(value = Exception.class)
201 public CommonResult<?> defaultExceptionHandler(HttpServletRequest req, Throwable ex) {
202     log.error("[defaultExceptionHandler]", ex);
203     // 插入异常日志
204     this.createExceptionLog(req, ex);
205     // 返回 ERROR CommonResult
206     return CommonResult.error(INTERNAL_SERVER_ERROR.getCode(), INTERNAL_SERVER_ERROR.getMsg());
207 }
208
209 private void createExceptionLog(HttpServletRequest req, Throwable e) {
210     // 插入错误日志
211     ApiErrorLogCreateReqDTO errorLog = new ApiErrorLogCreateReqDTO();
212     try {
213         // 初始化 errorLog
214         initExceptionLog(errorLog, req, e);
215         // 执行插入 errorLog
216         apiErrorLogFrameworkService.createApiErrorLogAsync(errorLog);
217     } catch (Throwable th) {
218         log.error("[createExceptionLog][url({}) log({}) 发生异常]", req.getRequestURI(), JsonUtils.toJsonString(errorLog), th);
219     }
220 }
221
222 private void initExceptionLog(ApiErrorLogCreateReqDTO errorLog, HttpServletRequest request, Throwable e) {...}
223
224
225
226
227
228
```

错误日志的存储，由 `yudao-module-infra` 的 `ErrorLog` 模块实现，记录到数据库的 `infra_api_error_log` 表。

← [Excel 导入导出](#)

[MyBatis 数据库](#) →



Theme by [Vdoing](#) | Copyright © 2019-2024 芋道源码 | MIT License