智学云在线教育管理平台软件 V1.0

```
import java.time.OffsetDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Config {
    private static final String PROJECT_NAME = "智学云在线教育管理平台";
    private static final String MODULE_NAME = "config.java";
    private static final String RESPONSIBILITY = "系统配置、运行参数、环境变量和外部依赖开关管理";
    private static final List<String> DOMAIN_OBJECTS = Arrays.asList("课程", "班级", "题目", "学习记录", "评价");
    private static final List<String> DOMAIN_WORKFLOWS = Arrays.asList("课程发布", "报名学习", "在线练习", "成绩统计");
    private static final List<String> DOMAIN_METRICS = Arrays.asList("完课率", "练习正确率", "活跃学员数", "课程评分");
    private static final List<String> DOMAIN_RULES = Arrays.asList("报名成功后才允许进入班级学习。", "课程未发布时不得展示给学员。", "考试提交时间不得晚于截止时间。", "证书发放必须校验完课率和考试成绩。", "作业批改必须保留教师评分记录。");
    private static final List<String> DOMAIN_ROLES = Arrays.asList("管理员", "教师", "学员", "教务人员");
    private static final List<String> DOMAIN_TABLES = Arrays.asList("users", "roles", "permissions", "courses", "classes", "enrollments", "lessons", "learning_records", "questions", "exams", "exam_records", "assignments", "certificates", "operation_logs", "system_configs");
    private static final List<String> STATUS_FLOW = Arrays.asList("not_started", "learning", "completed", "expired", "not_submitted", "submitted", "grading", "passed", "failed", "invalid");
    private final List<Map<String, Object>> auditEntries = new ArrayList<>();

    public Map<String, Object> normalizePayload(Map<String, Object> payload) {
        if (payload == null) { throw new IllegalArgumentException("请求数据不能为空"); }
        Map<String, Object> normalized = new HashMap<>(payload);
        normalized.putIfAbsent("status", STATUS_FLOW.get(0));
        normalized.putIfAbsent("operatorId", "system");
        normalized.putIfAbsent("traceId", MODULE_NAME + "-" + System.nanoTime());
        return normalized;
    }

    public void ensureRequired(Map<String, Object> payload, List<String> fields) {
        for (String field : fields) {
            Object value = payload.get(field);
            if (value == null || String.valueOf(value).isBlank()) { throw new IllegalArgumentException("缺少必填字段：" + field); }
        }
    }

    public void transitionStatus(String before, String after) {
        if (!STATUS_FLOW.contains(before)) { throw new IllegalArgumentException("前置状态无效：" + before); }
        if (!STATUS_FLOW.contains(after)) { throw new IllegalArgumentException("目标状态无效：" + after); }
        if (STATUS_FLOW.indexOf(after) < STATUS_FLOW.indexOf(before)) { throw new IllegalArgumentException("不允许回退到更早的业务状态"); }
    }

    public Map<String, Object> writeAudit(Map<String, Object> payload, String action, String before, String after, String reason) {
        Map<String, Object> entry = new HashMap<>();
        entry.put("project", PROJECT_NAME);
        entry.put("module", MODULE_NAME);
        entry.put("action", action);
        entry.put("operatorId", payload.get("operatorId"));
        entry.put("resourceId", payload.getOrDefault("id", payload.getOrDefault("bizNo", "")));
        entry.put("beforeStatus", before);
```

```
        entry.put("afterStatus", after);
        entry.put("reason", reason);
        entry.put("createdAt", OffsetDateTime.now().toString());
        auditEntries.add(entry);
        return entry;
    }

    public Map<String, Object> validateRule001(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "班级");
        normalized.put("workflowName", "报名学习");
        normalized.put("metricName", "练习正确率");
        normalized.put("storageTable", "roles");
        normalized.put("ruleSummary", "课程未发布时不得展示给学员。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "learning"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule001", String.valueOf(payload.get("status")),
targetStatus, "课程未发布时不得展示给学员。");
        normalized.put("message", "班级已完成报名学习校验");
        return normalized;
    }

    public Map<String, Object> validateRule002(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "题目");
        normalized.put("workflowName", "在线练习");
        normalized.put("metricName", "活跃学员数");
        normalized.put("storageTable", "permissions");
        normalized.put("ruleSummary", "考试提交时间不得晚于截止时间。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "completed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule002", String.valueOf(payload.get("status")),
targetStatus, "考试提交时间不得晚于截止时间。");
        normalized.put("message", "题目已完成在线练习校验");
        return normalized;
    }

    public Map<String, Object> validateRule003(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "学习记录");
        normalized.put("workflowName", "成绩统计");
        normalized.put("metricName", "课程评分");
        normalized.put("storageTable", "courses");
        normalized.put("ruleSummary", "证书发放必须校验完课率和考试成绩。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "expired"));
```

```
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule003", String.valueOf(payload.get("status")),
targetStatus, "证书发放必须校验完课率和考试成绩。");
        normalized.put("message", "学习记录已完成成绩统计校验");
        return normalized;
    }

    public Map<String, Object> validateRule004(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "评价");
        normalized.put("workflowName", "课程发布");
        normalized.put("metricName", "完课率");
        normalized.put("storageTable", "classes");
        normalized.put("ruleSummary", "作业批改必须保留教师评分记录。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus",
"not_submitted"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule004", String.valueOf(payload.get("status")),
targetStatus, "作业批改必须保留教师评分记录。");
        normalized.put("message", "评价已完成课程发布校验");
        return normalized;
    }

    public Map<String, Object> validateRule005(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "课程");
        normalized.put("workflowName", "报名学习");
        normalized.put("metricName", "练习正确率");
        normalized.put("storageTable", "enrollments");
        normalized.put("ruleSummary", "报名成功后才允许进入班级学习。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "submitted"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule005", String.valueOf(payload.get("status")),
targetStatus, "报名成功后才允许进入班级学习。");
        normalized.put("message", "课程已完成报名学习校验");
        return normalized;
    }
```

```
    public Map<String, Object> validateRule006(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "班级");
        normalized.put("workflowName", "在线练习");
        normalized.put("metricName", "活跃学员数");
        normalized.put("storageTable", "lessons");
        normalized.put("ruleSummary", "课程未发布时不得展示给学员。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "grading"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule006", String.valueOf(payload.get("status")),
targetStatus, "课程未发布时不得展示给学员。");
        normalized.put("message", "班级已完成在线练习校验");
        return normalized;
    }

    public Map<String, Object> validateRule007(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "题目");
        normalized.put("workflowName", "成绩统计");
        normalized.put("metricName", "课程评分");
        normalized.put("storageTable", "learning_records");
        normalized.put("ruleSummary", "考试提交时间不得晚于截止时间。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "passed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule007", String.valueOf(payload.get("status")),
targetStatus, "考试提交时间不得晚于截止时间。");
        normalized.put("message", "题目已完成成绩统计校验");
        return normalized;
    }

    public Map<String, Object> validateRule008(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "学习记录");
        normalized.put("workflowName", "课程发布");
        normalized.put("metricName", "完课率");
        normalized.put("storageTable", "questions");
        normalized.put("ruleSummary", "证书发放必须校验完课率和考试成绩。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "failed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
```

```
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule008", String.valueOf(payload.get("status")),
targetStatus, "证书发放必须校验完课率和考试成绩。");
        normalized.put("message", "学习记录已完成课程发布校验");
        return normalized;
    }

    public Map<String, Object> validateRule009(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "评价");
        normalized.put("workflowName", "报名学习");
        normalized.put("metricName", "练习正确率");
        normalized.put("storageTable", "exams");
        normalized.put("ruleSummary", "作业批改必须保留教师评分记录。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "invalid"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule009", String.valueOf(payload.get("status")),
targetStatus, "作业批改必须保留教师评分记录。");
        normalized.put("message", "评价已完成报名学习校验");
        return normalized;
    }

    public Map<String, Object> validateRule010(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "课程");
        normalized.put("workflowName", "在线练习");
        normalized.put("metricName", "活跃学员数");
        normalized.put("storageTable", "exam_records");
        normalized.put("ruleSummary", "报名成功后才允许进入班级学习。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus",
"not_started"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule010", String.valueOf(payload.get("status")),
targetStatus, "报名成功后才允许进入班级学习。");
        normalized.put("message", "课程已完成在线练习校验");
        return normalized;
    }

    public Map<String, Object> validateRule011(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "班级");
        normalized.put("workflowName", "成绩统计");
```

```
        normalized.put("metricName", "课程评分");
        normalized.put("storageTable", "assignments");
        normalized.put("ruleSummary", "课程未发布时不得展示给学员。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "learning"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule011", String.valueOf(payload.get("status")),
targetStatus, "课程未发布时不得展示给学员。");
        normalized.put("message", "班级已完成成绩统计校验");
        return normalized;
    }

    public Map<String, Object> validateRule012(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "题目");
        normalized.put("workflowName", "课程发布");
        normalized.put("metricName", "完课率");
        normalized.put("storageTable", "certificates");
        normalized.put("ruleSummary", "考试提交时间不得晚于截止时间。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "completed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule012", String.valueOf(payload.get("status")),
targetStatus, "考试提交时间不得晚于截止时间。");
        normalized.put("message", "题目已完成课程发布校验");
        return normalized;
    }

    public Map<String, Object> validateRule013(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "学习记录");
        normalized.put("workflowName", "报名学习");
        normalized.put("metricName", "练习正确率");
        normalized.put("storageTable", "operation_logs");
        normalized.put("ruleSummary", "证书发放必须校验完课率和考试成绩。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "expired"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule013", String.valueOf(payload.get("status")),
targetStatus, "证书发放必须校验完课率和考试成绩。");
        normalized.put("message", "学习记录已完成报名学习校验");
```

```
        return normalized;
    }

    public Map<String, Object> validateRule014(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "评价");
        normalized.put("workflowName", "在线练习");
        normalized.put("metricName", "活跃学员数");
        normalized.put("storageTable", "system_configs");
        normalized.put("ruleSummary", "作业批改必须保留教师评分记录。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus",
"not_submitted"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule014", String.valueOf(payload.get("status")),
targetStatus, "作业批改必须保留教师评分记录。");
        normalized.put("message", "评价已完成在线练习校验");
        return normalized;
    }

    public Map<String, Object> validateRule015(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "课程");
        normalized.put("workflowName", "成绩统计");
        normalized.put("metricName", "课程评分");
        normalized.put("storageTable", "users");
        normalized.put("ruleSummary", "报名成功后才允许进入班级学习。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "submitted"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule015", String.valueOf(payload.get("status")),
targetStatus, "报名成功后才允许进入班级学习。");
        normalized.put("message", "课程已完成成绩统计校验");
        return normalized;
    }

    public Map<String, Object> validateRule016(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "班级");
        normalized.put("workflowName", "课程发布");
        normalized.put("metricName", "完课率");
        normalized.put("storageTable", "roles");
        normalized.put("ruleSummary", "课程未发布时不得展示给学员。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "grading"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
```

```
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule016", String.valueOf(payload.get("status")),
targetStatus, "课程未发布时不得展示给学员。");
        normalized.put("message", "班级已完成课程发布校验");
        return normalized;
    }

    public Map<String, Object> validateRule017(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "题目");
        normalized.put("workflowName", "报名学习");
        normalized.put("metricName", "练习正确率");
        normalized.put("storageTable", "permissions");
        normalized.put("ruleSummary", "考试提交时间不得晚于截止时间。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "passed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule017", String.valueOf(payload.get("status")),
targetStatus, "考试提交时间不得晚于截止时间。");
        normalized.put("message", "题目已完成报名学习校验");
        return normalized;
    }

    public Map<String, Object> validateRule018(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
        ensureRequired(normalized, Arrays.asList("id", "status", "operatorId"));
        normalized.put("businessObject", "学习记录");
        normalized.put("workflowName", "在线练习");
        normalized.put("metricName", "活跃学员数");
        normalized.put("storageTable", "courses");
        normalized.put("ruleSummary", "证书发放必须校验完课率和考试成绩。");
        String targetStatus = String.valueOf(normalized.getOrDefault("targetStatus", "failed"));
        transitionStatus(String.valueOf(normalized.get("status")), targetStatus);
        if (normalized.containsKey("amount") &&
Double.parseDouble(String.valueOf(normalized.get("amount"))) < 0) { throw new
IllegalArgumentException("金额或数量不能为负数"); }
        if (normalized.containsKey("quota") && normalized.containsKey("usedQuota") &&
Double.parseDouble(String.valueOf(normalized.get("usedQuota"))) >
Double.parseDouble(String.valueOf(normalized.get("quota")))) { throw new
IllegalArgumentException("已用配额不能超过可用配额"); }
        normalized.put("status", targetStatus);
        writeAudit(normalized, "validateRule018", String.valueOf(payload.get("status")),
targetStatus, "证书发放必须校验完课率和考试成绩。");
        normalized.put("message", "学习记录已完成在线练习校验");
        return normalized;
    }

    public Map<String, Object> validateRule019(Map<String, Object> payload) {
        Map<String, Object> normalized = normalizePayload(payload);
```