import { Context, Schema, Session, Tables, User, $, Row } from 'koishi'

import dayjs from 'dayjs'

import zhCN from './locales/zh-CN'

// -------------------------- 1. 核心类型定义 --------------------------

export interface SleepRecord {

id: number // 自增主键（由Koishi模型选项自动处理）

uid: number // 关联用户ID

sleepTime: string // 入睡时间（ISO字符串）

wakeTime?: string // 起床时间（ISO字符串，可选）

duration?: number // 睡眠时长（分钟，可选）

date: string // 记录日期（YYYY-MM-DD）

}

// -------------------------- 2. 扩展Koishi内置类型 --------------------------

declare module 'koishi' {

interface User {

sleepTotal: number // 累计睡眠时长（分钟）

sleepCount: number // 睡眠记录总次数

lastSleepId?: number // 最后一条未完成睡眠记录ID（可选）

}

interface Tables {

sleep\_record: SleepRecord // 睡眠记录表

}

}

// -------------------------- 3. 插件配置与元属性 --------------------------

export interface Config {

time: {

morningStart: number // 早安开始小时（0-23）

morningEnd: number // 早安结束小时（0-23）

eveningStart: number // 晚安开始小时（0-23）

eveningEnd: number // 晚安结束小时（0-23）

}

tips: {

manyEveningThreshold: number // 重复晚安阈值（2-10）

emptyRecord: string // 无记录提示语

successEvening: string // 晚安成功提示语（{{time}} 变量）

repeatEvening: string // 重复晚安提示语（{{time}} 变量）

manyEvening: string // 超阈值提示语（{{count}} 变量）

dbError: string // 数据库异常提示语

noUnfinished: string // 无未完成记录提示语

}

rank: {

defaultTop: number // 排行榜默认条数（1-50）

showAverage: boolean // 是否显示平均时长

}

}

export const name = 'sleep-shiyi'

export const usage = '记录用户睡眠作息，支持晚安/早安识别、排行榜与个人记录查询'

export const inject = { required: ['database'], optional: ['i18n'] }

export const reusable = false

// 配置Schema

export const Config: Schema<Config> = Schema.object({

time: Schema.object({

morningStart: Schema.number().min(0).max(23).default(6).description('早安开始小时（24小时制）'),

morningEnd: Schema.number().min(0).max(23).default(12).description('早安结束小时（24小时制）'),

eveningStart: Schema.number().min(0).max(23).default(21).description('晚安开始小时（24小时制）'),

eveningEnd: Schema.number().min(0).max(23).default(3).description('晚安结束小时（24小时制，支持跨天）'),

}).description('时段配置'),

tips: Schema.object({

manyEveningThreshold: Schema.number().min(2).max(10).default(3).description('重复晚安提示阈值'),

emptyRecord: Schema.string().default('❌ 你还没有睡眠记录～发送"晚安"开始吧！').description('无记录提示语'),

successEvening: Schema.string().default('🌙 晚安！已记录入睡时间：{{time}}').description('晚安成功提示语'),

repeatEvening: Schema.string().default('⚠️ 你今天已记录入睡时间：{{time}}').description('重复晚安提示语'),

manyEvening: Schema.string().default('😱 你已说{{count}}次晚安！快休息吧～').description('多次晚安提示语'),

dbError: Schema.string().default('❌ 数据库操作异常，请稍后再试').description('数据库异常提示语'),

noUnfinished: Schema.string().default('❌ 未找到你的未完成睡眠记录～请先发送「晚安」记录入睡时间').description('无未完成记录提示语'),

}).description('提示语配置'),

rank: Schema.object({

defaultTop: Schema.number().min(1).max(50).default(10).description('排行榜默认显示条数'),

showAverage: Schema.boolean().default(true).description('是否显示平均睡眠时长'),

}).description('排行榜配置'),

}) as Schema<Config>

// -------------------------- 4. 工具函数 --------------------------

const sleepTableName = 'sleep\_record' as const

/\*\*

\* 检查当前小时是否在目标时段内（支持跨天）

\*/

function isInSpan(hour: number, config: Config, type: 'morning' | 'evening'): boolean {

const { morningStart, morningEnd, eveningStart, eveningEnd } = config.time

const [start, end] = type === 'morning' ? [morningStart, morningEnd] : [eveningStart, eveningEnd]

return start <= end ? (hour >= start && hour <= end) : (hour >= start || hour <= end)

}

/\*\*

\* 替换提示语中的变量

\*/

function replaceTipVars(tip: string, vars: Record<string, string | number>): string {

return Object.entries(vars).reduce((res, [key, val]) => res.replace(`{{${key}}}`, String(val)), tip)

}

/\*\*

\* 计算睡眠时长（支持跨天）

\*/

function calcSleepDuration(sleepTime: string, wakeTime: Date): number {

const sleepMs = new Date(sleepTime).getTime()

const wakeMs = wakeTime.getTime()

const durationMs = wakeMs >= sleepMs ? wakeMs - sleepMs : wakeMs + 24 \* 60 \* 60 \* 1000 - sleepMs

return Math.round(durationMs / 60000)

}

/\*\*

\* 安全获取用户ID（适配多平台）

\*/

function getSafeUserId(session: Session, ctx: Context): number | null {

// 1. 优先使用Koishi标准uid

if (typeof session.uid === 'number' && session.uid > 0) {

ctx.logger.debug(`[getSafeUserId] 从session.uid获取: ${session.uid}`)

return session.uid

}

// 2. 适配OneBot平台（QQ/微信等）

if (session.platform === 'onebot') {

const event = session.event as any

if (event?.user\_id) {

const uid = Number(event.user\_id)

if (uid > 0) return uid

}

}

// 3. 适配QQ频道

if (session.platform === 'qqguild') {

const event = session.event as any

if (event?.member?.user?.id) {

const uid = Number(event.member.user.id)

if (uid > 0) return uid

}

}

// 4. 适配Discord

if (session.platform === 'discord') {

const event = session.event as any

if (event?.author?.id) {

const uid = Number(event.author.id)

if (uid > 0) return uid

}

}

// 5. 兜底从session.user获取

if (session.user) {

const userId = (session.user as any).id

if (typeof userId === 'number' && userId > 0) return userId

}

// 6. 记录日志便于排查

ctx.logger.error(`[getSafeUserId] 无法识别用户ID，会话信息:`, {

platform: session.platform,

userId: session.userId,

hasUser: !!session.user

})

return null

}

/\*\*

\* 校验并清理无效的lastSleepId

\*/

async function validateLastSleepId(ctx: Context, uid: number, lastSleepId?: number): Promise<number | null> {

if (!lastSleepId) return null

try {

// 用ORM查询，确保记录存在且属于当前用户

const records = await ctx.database.get(sleepTableName, { id: lastSleepId, uid })

if (records.length > 0) {

ctx.logger.debug(`[validateLastSleepId] 有效 - uid: ${uid}, lastSleepId: ${lastSleepId}`)

return lastSleepId

}

// 无效则清空

await ctx.database.set('user', { id: uid }, { lastSleepId: null })

ctx.logger.warn(`[validateLastSleepId] 无效ID已清空 - uid: ${uid}, lastSleepId: ${lastSleepId}`)

return null

} catch (error) {

ctx.logger.error(`[validateLastSleepId] 异常 - uid: ${uid}`, error)

return null

}

}

// -------------------------- 5. 数据库模型初始化（完全符合Koishi规范） --------------------------

function initModels(ctx: Context) {

// 扩展User表（使用默认主键）

ctx.model.extend('user', {

sleepTotal: { type: 'integer', initial: 0 },

sleepCount: { type: 'integer', initial: 0 },

lastSleepId: { type: 'unsigned', nullable: true }, // 允许为NULL，适配SQLite

})

// 定义睡眠记录表（关键修复：移除autoInc属性，在模型选项中配置自增）

ctx.model.extend(sleepTableName, {

id: { type: 'unsigned', nullable: false }, // 仅保留合法属性：类型+非空

uid: { type: 'unsigned', nullable: false }, // 关联用户ID（非空）

sleepTime: { type: 'string', nullable: false }, // 入睡时间（非空）

wakeTime: { type: 'string', nullable: true }, // 起床时间（允许NULL，适配SQLite）

duration: { type: 'integer', nullable: true }, // 时长（允许NULL）

date: { type: 'string', nullable: false }, // 日期（非空）

}, {

primary: 'id', // 模型选项：配置主键

autoInc: true, // 模型选项：配置主键自增（符合Koishi规范）

indexes: [

['uid'], // 按用户ID索引，优化查询

['uid', 'date'], // 按用户+日期索引，优化晚安重复检查

['uid', 'wakeTime'], // 按用户+起床时间索引，优化早安查询

],

})

ctx.logger.info(`[${name}] 数据库模型初始化完成（完全符合Koishi规范）`)

}

// -------------------------- 6. 核心功能实现 --------------------------

function registerFeatures(ctx: Context, config: Config) {

const { tips, rank } = config

// 功能1：晚安记录（确保创建NULL的wakeTime）

ctx.middleware(async (session: Session, next) => {

const content = session.content?.trim().toLowerCase()

const now = new Date()

const nowHour = now.getHours()

const today = dayjs(now).format('YYYY-MM-DD')

const eveningKeywords = ['睡觉', '晚安', '睡了', '休息', '我要睡了']

// 前置校验：关键词+时段

if (!content || !eveningKeywords.some(k => content.includes(k)) || !isInSpan(nowHour, config, 'evening')) {

return next()

}

// 获取用户ID

const uid = getSafeUserId(session, ctx)

if (uid === null) {

return '❌ 无法识别用户信息，请稍后再试'

}

try {

// 检查用户是否存在，不存在则创建

let users = await ctx.database.get('user', { id: uid })

if (!users.length) {

const userName = session.author?.nickname || `用户${uid.toString().slice(-4)}`

await ctx.database.create('user', { id: uid, name: userName })

ctx.logger.info(`[晚安] 创建新用户 - uid: ${uid}`)

}

const user = users[0] || { sleepCount: 0 }

// 统计当天记录次数

const todayRecords = await ctx.database.get(sleepTableName, { uid, date: today })

const currentCount = todayRecords.length

const newCount = currentCount + 1

// 【关键】检查今日未完成记录（仅匹配wakeTime为NULL，适配SQLite）

const unfinishedRecords = await ctx.database.get(sleepTableName, (row: Row<SleepRecord>) =>

$.and(

$.eq(row.uid, uid),

$.eq(row.date, today),

$.eq(row.wakeTime, null) // 仅查NULL，与创建时状态一致

)

) as SleepRecord[]

if (unfinishedRecords.length > 0) {

const sleepTimeStr = dayjs(unfinishedRecords[0].sleepTime).format('HH:mm')

return replaceTipVars(tips.repeatEvening, { time: sleepTimeStr })

}

// 事务创建记录+更新用户（显式设置wakeTime为NULL）

const newRecord = await ctx.database.transact(async (tx) => {

// 创建睡眠记录，wakeTime明确设为NULL（id由自增自动生成）

const record = await tx.create(sleepTableName, {

uid,

sleepTime: now.toISOString(),

date: today,

wakeTime: null, // 关键：确保写入NULL，适配SQLite

duration: null,

}) as SleepRecord

// 更新用户的记录数和最后一条未完成记录ID

await tx.set('user', { id: uid }, {

lastSleepId: record.id,

sleepCount: user.sleepCount + 1,

})

return record

})

// 验证创建结果（日志输出实际状态）

ctx.logger.debug(`[晚安] 创建记录验证 - id: ${newRecord.id}, wakeTime: ${newRecord.wakeTime}, 类型: ${typeof newRecord.wakeTime}`)

// 返回提示语

const timeStr = dayjs(now).format('HH:mm')

return newCount >= tips.manyEveningThreshold

? replaceTipVars(tips.manyEvening, { count: newCount })

: replaceTipVars(tips.successEvening, { time: timeStr })

} catch (error) {

ctx.logger.error(`[晚安] 异常 - uid: ${uid}`, error)

return tips.dbError

}

})

// 功能2：早安记录（核心修复：精准查询未完成记录）

ctx.middleware(async (session: Session, next) => {

const content = session.content?.trim().toLowerCase()

const now = new Date()

const nowHour = now.getHours()

const morningKeywords = ['早安', '早上好', '早', '我醒了', '起床了']

// 前置校验：关键词+时段

if (!content || !morningKeywords.some(k => content.includes(k)) || !isInSpan(nowHour, config, 'morning')) {

return next()

}

// 获取用户ID

const uid = getSafeUserId(session, ctx)

if (uid === null) {

return '❌ 无法识别用户信息，请稍后再试'

}

try {

// 检查用户是否存在

const users = await ctx.database.get('user', { id: uid })

if (!users.length) {

return '❌ 未找到你的用户记录～先发送「晚安」创建记录吧'

}

const user = users[0]

// 计算日期范围（覆盖跨天场景）

const today = dayjs(now).format('YYYY-MM-DD')

const yesterday = dayjs(now).subtract(1, 'day').format('YYYY-MM-DD')

ctx.logger.debug(`[早安] 日期范围 - 今天: ${today}, 昨天: ${yesterday}`)

// 【关键】统一查询条件（仅匹配wakeTime为NULL，覆盖当天+昨天）

const baseCondition = (row: Row<SleepRecord>) => $.and(

$.eq(row.uid, uid),

$.eq(row.wakeTime, null), // 与晚安创建的状态一致

$.or($.eq(row.date, today), $.eq(row.date, yesterday)) // 跨天兜底

)

let sleepRecords: SleepRecord[] = []

// 策略1：按lastSleepId精准查询（最快）

const validLastSleepId = await validateLastSleepId(ctx, uid, user.lastSleepId)

if (validLastSleepId) {

sleepRecords = await ctx.database.get(sleepTableName, (row: Row<SleepRecord>) =>

$.and(baseCondition(row), $.eq(row.id, validLastSleepId))

) as SleepRecord[]

ctx.logger.debug(`[早安] 策略1结果 - 记录数: ${sleepRecords.length}, 目标ID: ${validLastSleepId}`)

}

// 策略2：查询当天+昨天的未完成记录（容错）

if (!sleepRecords.length) {

sleepRecords = await ctx.database.get(sleepTableName, baseCondition, {

sort: { sleepTime: 'desc' }, // 最新记录优先

limit: 1

}) as SleepRecord[]

ctx.logger.debug(`[早安] 策略2结果 - 记录数: ${sleepRecords.length}`)

// 同步lastSleepId（确保下次查询更快）

if (sleepRecords.length) {

await ctx.database.set('user', { id: uid }, { lastSleepId: sleepRecords[0].id })

}

}

// 策略3：查询用户所有未完成记录（彻底兜底）

if (!sleepRecords.length) {

sleepRecords = await ctx.database.get(sleepTableName, (row: Row<SleepRecord>) =>

$.and($.eq(row.uid, uid), $.eq(row.wakeTime, null))

, { sort: { sleepTime: 'desc' }, limit: 1 }) as SleepRecord[]

ctx.logger.debug(`[早安] 策略3结果 - 记录数: ${sleepRecords.length}, 无日期限制`)

}

// 打印记录详情（调试关键）

if (sleepRecords.length) {

const record = sleepRecords[0]

ctx.logger.debug(`[早安] 找到记录 - id: ${record.id}, date: ${record.date}, wakeTime: ${record.wakeTime}`)

} else {

ctx.logger.warn(`[早安] 未找到记录 - uid: ${uid}, 已查日期: ${today}, ${yesterday}`)

return tips.noUnfinished

}

// 记录完整性校验

const sleepRecord = sleepRecords[0]

if (!sleepRecord.sleepTime) {

ctx.logger.error(`[早安] 记录损坏 - id: ${sleepRecord.id}`)

return '❌ 睡眠记录不完整，无法计算时长'

}

// 计算睡眠时长并更新记录

const duration = calcSleepDuration(sleepRecord.sleepTime, now)

const wakeTimeISO = now.toISOString()

// 更新睡眠记录（标记为已完成）

await ctx.database.set(sleepTableName, { id: sleepRecord.id }, {

wakeTime: wakeTimeISO,

duration,

})

// 更新用户累计数据

await ctx.database.set('user', { id: uid }, {

sleepTotal: (user.sleepTotal || 0) + duration,

lastSleepId: null, // 清空已完成的记录ID

})

// 返回结果

const sleepTimeStr = dayjs(sleepRecord.sleepTime).format('HH:mm')

const wakeTimeStr = dayjs(now).format('HH:mm')

const durationText = `${Math.floor(duration / 60)}小时${duration % 60}分钟`

return `☀️ 早上好！睡眠时长：${durationText}（${sleepTimeStr}→${wakeTimeStr}）`

} catch (error) {

ctx.logger.error(`[早安] 异常 - uid: ${uid}`, error)

return tips.dbError

}

})

// 功能3：睡眠排行榜

ctx.middleware(async (session: Session, next) => {

const content = session.content?.trim()

if (!content || !content.startsWith('sleep.rank ')) return next()

const uid = getSafeUserId(session, ctx)

if (uid === null) return '❌ 无法识别用户信息，请稍后再试'

try {

const args = content.split(' ').filter(Boolean)

const timeType = args[1] || 'week'

const topIndex = args.findIndex(arg => arg.startsWith('-top'))

const top = Math.min(

topIndex > -1 ? (Number(args[topIndex + 1]) || rank.defaultTop) : rank.defaultTop,

50

)

// 计算时间范围

let [startDate, endDate, title] = [

dayjs().subtract(1, 'week').format('YYYY-MM-DD'),

dayjs().format('YYYY-MM-DD'),

'本周'

]

switch (timeType) {

case 'day': startDate = endDate; title = '今日'; break

case 'month': startDate = dayjs().subtract(1, 'month').format('YYYY-MM-DD'); title = '本月'; break

}

// 查询有效记录（仅完成的记录）

const records = await ctx.database.get(sleepTableName, (row: Row<SleepRecord>) =>

$.and(

$.ne(row.duration, null), // 排除未完成记录

$.gte(row.date, startDate),

$.lte(row.date, endDate)

)

) as SleepRecord[]

if (!records.length) return tips.emptyRecord

// 统计用户数据

const userIds = Array.from(new Set(records.map(r => r.uid))) as number[]

const users = await ctx.database.get('user', { id: userIds })

const userMap = new Map<number, string>(users.map(u => [u.id, u.name]))

const userStats: Record<number, { total: number; count: number; name: string }> = {}

records.forEach(record => {

const uid = record.uid

const name = userMap.get(uid) || `用户${uid.toString().slice(-4)}`

if (!userStats[uid]) userStats[uid] = { total: 0, count: 0, name }

userStats[uid].total += record.duration!

userStats[uid].count += 1

})

// 生成排行榜

const sortedStats = Object.values(userStats)

.sort((a, b) => b.total - a.total)

.slice(0, top)

let rankText = `🏆 ${title} 睡眠时长排行榜（TOP${top}）：`

sortedStats.forEach((stat, idx) => {

const total = `${Math.floor(stat.total / 60)}小时${stat.total % 60}分钟`

const avg = rank.showAverage

? `（平均：${Math.floor((stat.total / stat.count) / 60)}小时${Math.floor((stat.total / stat.count) % 60)}分钟）`

: ''

rankText += `\n${idx + 1}. ${stat.name} - 总时长：${total}${avg}`

})

return rankText.trim()

} catch (error) {

ctx.logger.error(`[排行榜] 异常 - uid: ${uid}`, error)

return tips.dbError

}

})

// 功能4：个人睡眠记录查询

ctx.middleware(async (session: Session, next) => {

const content = session.content?.trim()

if (!content || !['我的睡眠', '睡眠记录', '我睡了多久'].some(k => content.includes(k))) {

return next()

}

const uid = getSafeUserId(session, ctx)

if (uid === null) return '❌ 无法识别用户信息，请稍后再试'

try {

const sevenDaysAgo = dayjs().subtract(7, 'day').format('YYYY-MM-DD')

const today = dayjs().format('YYYY-MM-DD')

const records = await ctx.database.get(sleepTableName, (row: Row<SleepRecord>) =>

$.and(

$.eq(row.uid, uid),

$.gte(row.date, sevenDaysAgo),

$.lte(row.date, today)

)

, { sort: { date: 'desc' } }) as SleepRecord[]

if (!records.length) return tips.emptyRecord

// 生成记录文本

let recordText = `📊 你的近${records.length}天睡眠记录：`

let totalDuration = 0

let completedCount = 0

records.forEach(record => {

const dateStr = dayjs(record.date).format('MM-DD')

const sleepTimeStr = dayjs(record.sleepTime).format('HH:mm')

const wakeTimeStr = record.wakeTime ? dayjs(record.wakeTime).format('HH:mm') : '未记录'

const durationText = record.duration

? `${Math.floor(record.duration / 60)}小时${record.duration % 60}分钟`

: '未完成'

recordText += `\n${dateStr}：入睡${sleepTimeStr} → 起床${wakeTimeStr}（时长：${durationText}）`

if (record.duration) {

totalDuration += record.duration

completedCount += 1

}

})

// 统计信息

if (completedCount > 0) {

const avg = Math.round(totalDuration / completedCount)

const avgText = `${Math.floor(avg / 60)}小时${avg % 60}分钟`

recordText += `\n📈 统计：平均睡眠时长 ${avgText}（共${completedCount}次记录）`

}

return recordText

} catch (error) {

ctx.logger.error(`[个人记录] 异常 - uid: ${uid}`, error)

return tips.dbError

}

})

// 功能5：帮助指令

ctx.middleware(async (session: Session, next) => {

const content = session.content?.trim()

if (content !== 'sleep.help' && content !== '睡眠帮助') return next()

return `

📋 睡眠记录插件使用指南

1. 记录入睡：发送「晚安」「睡觉」「睡了」「休息」「我要睡了」

- 生效时段：${config.time.eveningStart}-${config.time.eveningEnd}点（支持跨天）

2. 记录起床：发送「早安」「早上好」「早」「我醒了」「起床了」

- 生效时段：${config.time.morningStart}-${config.time.morningEnd}点

3. 查看排行：发送「sleep.rank <类型> [-top 条数]」

- 类型：day=今日，week=本周（默认），month=本月

- 示例：sleep.rank day / sleep.rank week -top 5

4. 我的记录：发送「我的睡眠」「睡眠记录」「我睡了多久」

5. 帮助指令：发送「sleep.help」或「睡眠帮助」

`.trim()

})

}

// -------------------------- 7. 插件入口 --------------------------

export function apply(ctx: Context, config: Config) {

if (ctx.i18n) {

ctx.i18n.define('zh-CN', zhCN)

}

initModels(ctx)

registerFeatures(ctx, config)

ctx.logger.info(`[${name}] 插件加载完成（最终稳定版）`)

}