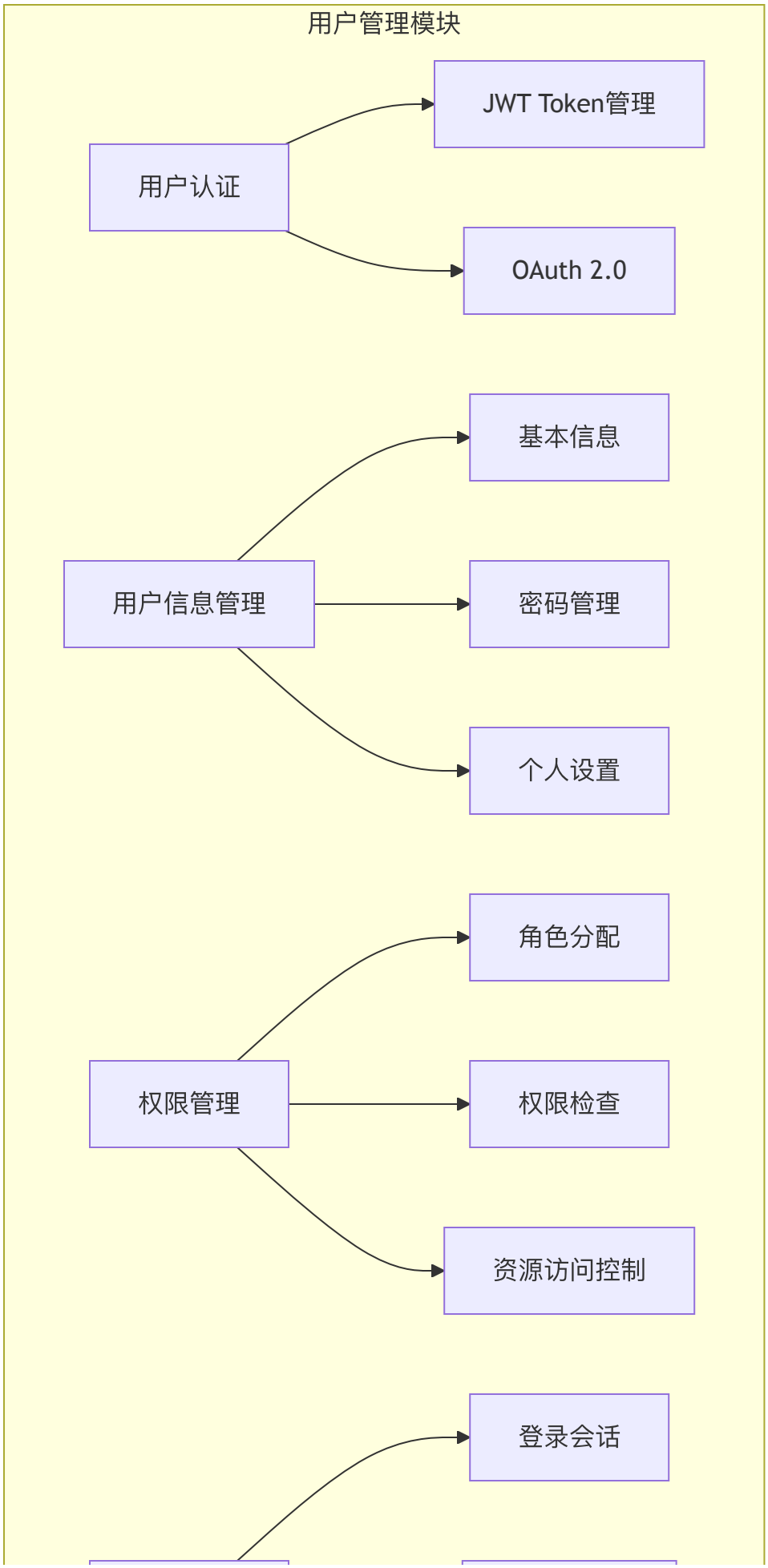
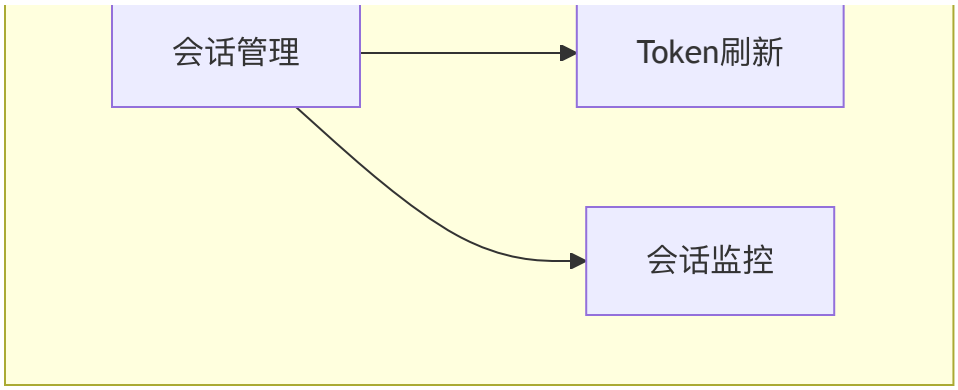
用户管理功能文档

1. 功能概述

用户管理模块是AI Proxy Service的核心基础模块，负责系统用户的全生命周期管理，包括用户注册、认证、授权、信息管理等功能。该模块采用RBAC（基于角色的访问控制）模型，支持多租户架构下的用户隔离和权限管理。

2. 功能架构


用户管理模块
用户认证
JWT Token管理
OAuth 2.0
用户信息管理
基本信息
密码管理
个人设置
权限管理
角色分配
权限检查
资源访问控制
登录会话



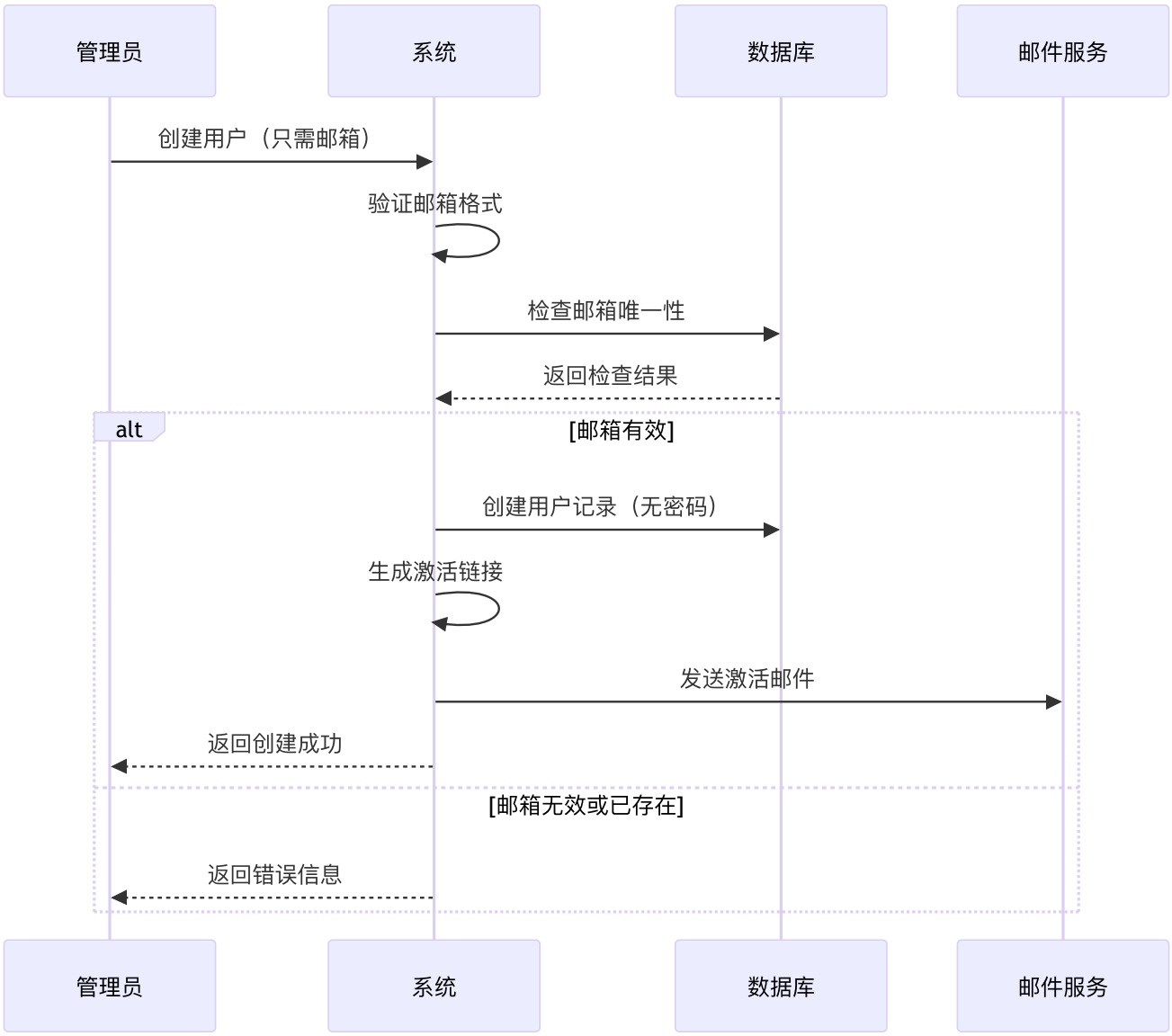
3. 详细功能说明

3.1 用户创建与首次登录

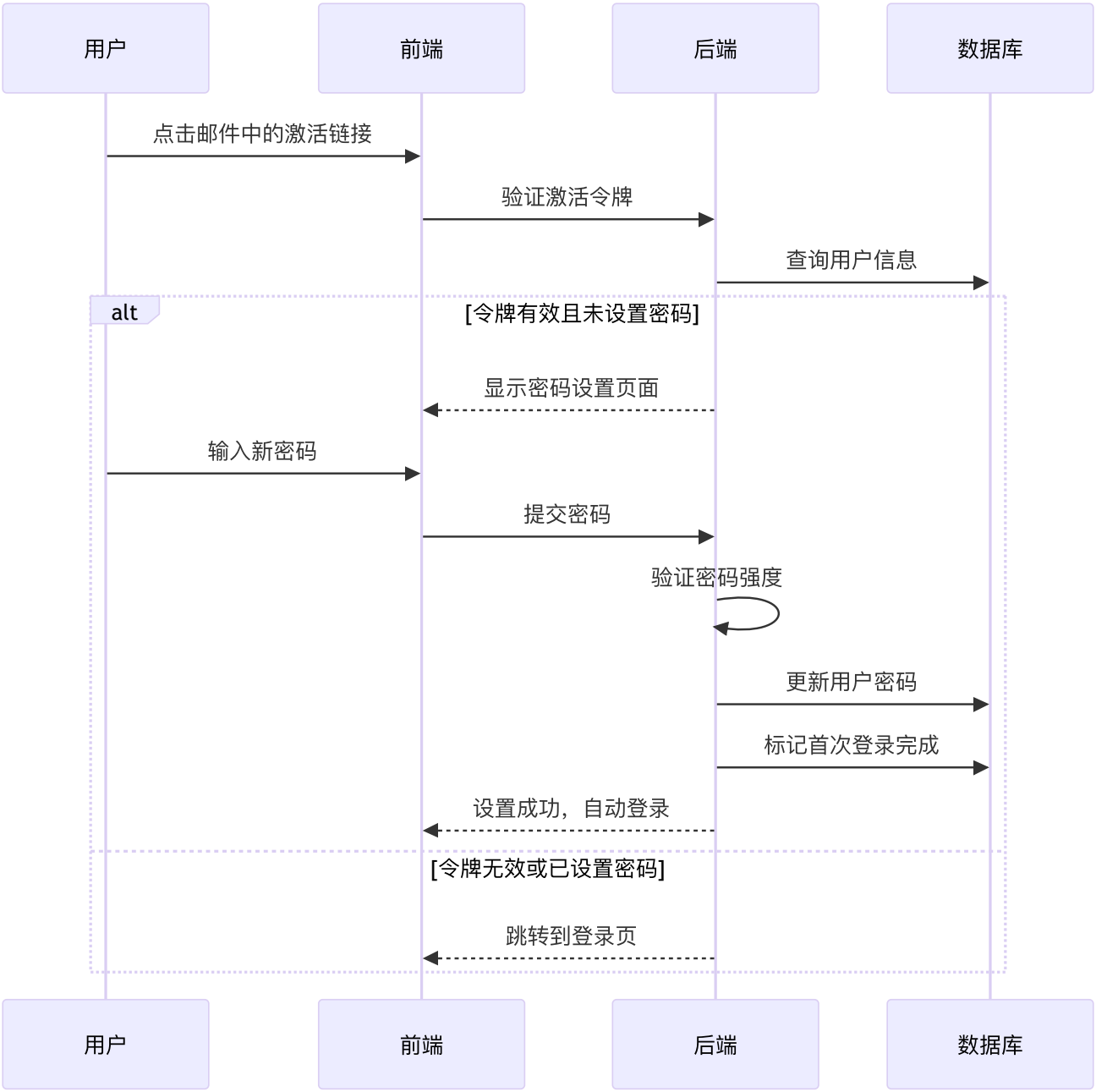
3.1.1 功能描述

用户由管理员创建，创建时只需提供邮箱，不设置密码。用户首次登录时必须设置密码才能使用系统。

3.1.2 用户创建流程（管理员操作）



3.1.3 首次登录设置密码流程



3.1.4 数据验证规则

字段	规则	说明
email	有效邮箱格式	邮箱全局唯一
password	最少8位，包含大小写字母和数字	首次登录时设置
isFirstLogin	布尔值	标识是否首次登录

3.2 用户认证

3.2.1 登录认证

支持的认证方式

1. **邮箱密码登录**
   - 使用邮箱 + 密码进行登录
   - 支持记住登录状态
   - 用于Web端和管理后台访问
2. **API Key认证**
   - 用于API调用场景
   - 无状态认证方式
   - 适用于程序化访问和集成

邮箱密码登录流程

```
// 登录请求结构
type LoginRequest struct {
    Email string `json:"email" v:"required|email"`
    Password string `json:"password"` // 首次登录可为空
    TenantCode string `json:"tenantCode"`
    RememberMe bool `json:"rememberMe"`
}

// 登录响应结构
type LoginResponse struct {
    AccessToken string `json:"accessToken"`
    RefreshToken string `json:"refreshToken"`
    ExpiresIn int64 `json:"expiresIn"`
    User UserInfo `json:"user"`
    RequireSetPassword bool `json:"requireSetPassword"` // 是否需要设置密码
}

// 首次设置密码请求
type SetPasswordRequest struct {
    Token string `json:"token" v:"required"` // 激活令牌
    Password string `json:"password" v:"required|length:8,32"`
    ConfirmPassword string `json:"confirmPassword" v:"required|same:password"`
}
```

API Key认证流程

```
// API Key认证方式
// 在请求头中添加 API Key
Headers: {
    "Authorization": "Bearer cr_your_api_key_here"
}

// 或者在请求参数中添加
Query: {
    "api_key": "cr_your_api_key_here"
}

// API Key验证逻辑
func ValidateAPIKey(key string) (*User, error) {
    // 验证Key格式
    if !strings.HasPrefix(key, "cr_") {
        return nil, ErrInvalidAPIKey
    }

    // 查询并验证Key
    apiKey := GetAPIKeyFromDB(key)
    if apiKey == nil || !apiKey.IsActive {
        return nil, ErrAPIKeyNotFound
    }

    // 检查过期时间
    if apiKey.ExpiresAt != nil && apiKey.ExpiresAt.Before(time.Now()) {
        return nil, ErrAPIKeyExpired
    }

    // 返回关联用户
    return GetUserByID(apiKey.UserID)
}
```

3.2.2 Token管理

JWT Token结构

```
{
  "sub": "user_id",
  "iss": "aiproxy",
  "aud": "aiproxy-api",
  "exp": 1234567890,
  "iat": 1234567890,
  "jti": "token_id",
  "user": {
    "id": "user_id",
    "username": "username",
    "email": "email",
    "tenantId": "tenant_id",
    "role": "role_name"
  }
}
```

Token刷新机制

- Access Token有效期：2小时
- Refresh Token有效期：7天
- 支持自动刷新和手动刷新

3.3 角色权限管理

3.3.1 角色定义

角色	代码	权限级别	说明
超级管理员	super_admin	100	系统最高权限，管理所有资源
租户管理员	tenant_admin	80	租户内最高权限，管理租户资源
普通用户	user	20	基础使用权限，仅管理个人资源

3.3.2 权限详细说明

超级管理员权限

```
SuperAdmin:
  租户管理:
    - 创建租户
    - 查看所有租户
    - 修改租户信息
    - 删除租户
    - 设置租户配额
  用户管理:
    - 创建用户
    - 查看所有用户
    - 修改用户信息
    - 删除用户
    - 分配用户角色
  AI账号管理:
    - 添加AI平台账号（Claude、OpenAI等）
    - 查看所有AI账号
    - 修改AI账号配置
    - 删除AI账号
    - 分配账号给租户
  系统管理:
    - 系统配置
    - 审计日志查看
    - 系统监控
```

租户管理员权限

```
TenantAdmin:
  用户管理:
    - 创建租户内用户
    - 查看租户内所有用户
    - 修改租户内用户信息
    - 删除租户内用户
    - 分配用户角色（仅普通用户角色）
  AI账号授权:
    - 查看租户分配的AI账号
    - 授权用户使用AI账号
    - 撤销用户AI账号授权
    - 设置用户使用配额
  租户管理:
    - 查看租户信息
    - 修改租户基础设置
    - 查看租户使用统计
    - 管理API Key
```

普通用户权限

```
User:
  个人信息:
    - 查看个人信息
    - 修改个人信息
    - 修改密码
    - 查看个人使用记录
  AI服务使用:
    - 使用被授权的AI账号
    - 查看使用配额
    - 查看使用历史
  账户充值:
    - 查看账户余额
    - 发起充值请求
    - 查看充值记录
    - 查看消费记录
  配额管理:
    - 查看个人配额规则
    - 查看配额使用情况
    - 查看配额使用历史
    - 查看配额状态
  API管理:
    - 创建个人API Key
    - 查看个人API Key
    - 删除个人API Key
```

3.3.3 权限模型

```
Permission:
  - resource: 资源类型（user/tenant/ai_account/apikey/billing）
  - action: 操作类型（create/read/update/delete/authorize）
  - scope: 作用范围（system/tenant/self）
  - role: 所需角色（super_admin/tenant_admin/user）
```

3.3.4 权限检查实现

```
// 权限检查中间件
func PermissionMiddleware(requiredRole string) gin.HandlerFunc {
    return func(c *gin.Context) {
        user := GetCurrentUser(c)

        // 超级管理员拥有所有权限
        if user.Role == "super_admin" {
            c.Next()
            return
        }

        // 检查角色权限
        if !HasPermission(user, requiredRole) {
            c.JSON(403, gin.H{
                "code": 403,
                "message": "权限不足",
            })
            c.Abort()
            return
        }

        c.Next()
    }
}

// 资源权限检查
func CheckResourcePermission(user *User, resource string, action string) bool {
    switch user.Role {
    case "super_admin":
        // 超级管理员可以操作所有资源
        return true

    case "tenant_admin":
        // 租户管理员权限检查
        switch resource {
        case "user":
            // 可以管理租户内用户
            return action != "delete" || !isSystemUser(resource)
        case "ai_account":
            // 可以授权AI账号给用户
            return action == "authorize" || action == "read"
```

```
        case "tenant":
            // 可以查看和修改租户基础信息
            return action == "read" || action == "update"
        default:
            return false
        }

    case "user":
        // 普通用户权限检查
        switch resource {
        case "user":
            // 只能操作自己的信息
            return action == "read" || action == "update" && isSelf(resource)
        case "ai_account":
            // 只能使用被授权的账号
            return action == "use" && isAuthorized(user, resource)
        case "billing":
            // 可以查看和充值
            return action == "read" || action == "recharge"
        case "quota":
            // 可以查看自己的配额
            return action == "read" && isSelf(resource)
        case "apikey":
            // 可以管理自己的API Key
            return isSelf(resource)
        default:
            return false
        }
    }

    return false
}

// 租户隔离检查
func CheckTenantIsolation(user *User, targetTenantId string) bool {
    // 超级管理员可以访问所有租户
    if user.Role == "super_admin" {
        return true
    }

    // 其他用户只能访问自己所在租户
    return user.TenantId == targetTenantId
}
```

3.4 用户信息管理

3.4.1 个人信息

用户基础信息

- **邮箱**（email）- 登录账号，不可修改
- **密码**（password）- 登录密码，可修改
- **钱包**（wallet）- 账户余额和充值管理

钱包信息结构

```
type UserWallet struct {
    UserId      string    `json:"userId"`
    Balance     float64   `json:"balance"`       // 账户余额
    Currency    string    `json:"currency"`      // 货币类型（CNY/USD）
    Status      string    `json:"status"`        // 状态（正常/冻结）
    UpdatedAt   time.Time `json:"updatedAt"`
}

// 用户配额信息
type UserQuotaInfo struct {
    UserId          string  `json:"userId"`
    HasQuotaRules   bool    `json:"hasQuotaRules"`      // 是否有配额规则
    CurrentHourLimit float64 `json:"currentHourLimit"`  // 当前小时限额
    TodayLimit      float64 `json:"todayLimit"`         // 今日限额
    MonthLimit      float64 `json:"monthLimit"`         // 本月限额
    TodayUsage      float64 `json:"todayUsage"`         // 今日已用
    MonthUsage      float64 `json:"monthUsage"`         // 本月已用
}
```

3.4.2 密码管理

密码修改流程

1. 验证原密码
2. 检查新密码强度
3. 更新密码（bcrypt加密）
4. 强制重新登录

密码重置流程

1. 请求重置（输入邮箱）

2. 发送重置链接
3. 验证重置令牌
4. 设置新密码

3.4.3 钱包管理

钱包功能

```
// 查看余额
GET /api/v1/users/wallet

Response:
{
  "balance": 1000.00,
  "currency": "CNY",
  "status": "normal"
}

// 充值
POST /api/v1/users/wallet/recharge
{
  "amount": 100.00,
  "paymentMethod": "alipay"
}

// 查看交易记录
GET /api/v1/users/wallet/transactions

Response:
{
  "transactions": [
    {
      "id": "tx_123",
      "type": "recharge",  // recharge/consume
      "amount": 100.00,
      "balance": 1100.00,
      "description": "账户充值",
      "createdAt": "2025-08-11T10:00:00Z"
    }
  ]
}
```

3.5 会话管理（Redis实现-简化版）

3.5.1 会话存储设计

```
// SessionService Redis会话管理服务（简化版）
type SessionService struct {
    redis *gredis.Redis
    ctx   context.Context
}

// SessionData 会话数据（只保留核心信息）
type SessionData struct {
    UserId   string `redis:"userId"`
    Email    string `redis:"email"`
    Role     string `redis:"role"`
    TenantId string `redis:"tenantId"`
}

// Redis Key设计（简化）
// session:{token} -> Hash（会话数据）
// 无需维护用户会话集合，不支持查询和踢出功能
```

3.5.2 核心功能实现

创建会话

```
func (s *SessionService) CreateSession(user *SessionUser) (string, error) {
    // 生成安全的随机Token
    token := generateSecureToken()
    sessionKey := fmt.Sprintf("session:%s", token)

    // 存储会话数据（只保留必要信息）
    sessionData := map[string]interface{}{
        "userId":   user.Id,
        "email":    user.Email,
        "role":     user.Role,
        "tenantId": user.TenantId,
    }

    // 存储到Redis
    err := s.redis.HSet(ctx, sessionKey, sessionData)
    if err != nil {
        return "", err
    }

    // 设置2小时过期
    s.redis.Expire(ctx, sessionKey, 7200)

    return token, nil
}
```

验证会话

```
func (s *SessionService) ValidateSession(token string) (*SessionData, error) {
    if token == "" {
        return nil, errors.New("token不能为空")
    }

    sessionKey := fmt.Sprintf("session:%s", token)

    // 检查会话是否存在
    exists, err := s.redis.Exists(ctx, sessionKey).Result()
    if err != nil || exists == 0 {
        return nil, errors.New("会话不存在或已过期")
    }

    // 获取会话数据
    data, err := s.redis.HGetAll(ctx, sessionKey).Result()
    if err != nil {
        return nil, err
    }

    // 刷新过期时间（滑动窗口）
    s.redis.Expire(ctx, sessionKey, 7200)

    // 返回会话数据
    return &SessionData{
        UserId:   data["userId"],
        Email:    data["email"],
        Role:     data["role"],
        TenantId: data["tenantId"],
    }, nil
}
```

刷新Token

```go
func (s *SessionService) RefreshToken(oldToken string) (string, error) {
    // 验证旧Token
    session, err := s.ValidateSession(oldToken)
    if err != nil {
        return "", err
    }

    // 创建新Token
    user := &SessionUser{
        Id:       session.UserId,
        Email:    session.Email,
        Role:     session.Role,
        TenantId: session.TenantId,
    }

    newToken, err := s.CreateSession(user)
    if err != nil {
        return "", err
    }

    // 删除旧Token
    oldSessionKey := fmt.Sprintf("session:%s", oldToken)
    s.redis.Del(ctx, oldSessionKey)

    return newToken, nil
}
```

登出

```
func (s *SessionService) Logout(token string) error {
    if token == "" {
        return nil
    }

    sessionKey := fmt.Sprintf("session:%s", token)

    // 删除会话（不关心是否存在）
    s.redis.Del(ctx, sessionKey)

    return nil
}
```

3.5.3 API接口

登出接口

```
POST /api/v1/auth/logout
Authorization: Bearer {token}

Response:
{
  "code": 0,
  "message": "登出成功"
}
```

刷新Token

```
POST /api/v1/auth/refresh
Content-Type: application/json

Request:
{
  "refreshToken": "old_token_here"
}

Response:
{
  "code": 0,
  "data": {
    "accessToken": "new_token_here",
    "expiresIn": 7200
  }
}
```

3.6 安全机制

3.6.1 登录安全

防暴力破解

- 登录失败次数限制（5次/10分钟）
- 验证码机制（可选）
- IP黑名单
- 账号锁定机制

异常检测

- 异地登录提醒
- 新设备登录验证
- 异常行为分析

3.6.2 数据安全

密码安全

- bcrypt加密存储（cost=10）
- 密码历史记录（防止重复使用）
- 定期密码更新提醒

敏感信息保护

- 敏感字段加密存储
- 传输层TLS加密
- 日志脱敏处理

4. 数据库设计

4.1 用户表（users）

```
CREATE TABLE users (
    id VARCHAR(36) PRIMARY KEY,
    tenant_id VARCHAR(36) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255),           -- 可为空，首次登录时设置
    role VARCHAR(20) NOT NULL DEFAULT 'user',
    status VARCHAR(20) DEFAULT 'active',
    is_first_login BOOLEAN DEFAULT true,  -- 是否首次登录
    activation_token VARCHAR(255),        -- 激活令牌
    token_expires_at TIMESTAMP,           -- 令牌过期时间
    last_login_at TIMESTAMP,
    login_count INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    deleted_at TIMESTAMP NULL,
    INDEX idx_tenant_id (tenant_id),
    INDEX idx_email (email),
    INDEX idx_status (status),
    INDEX idx_role (role),
    INDEX idx_activation_token (activation_token)
);
```

4.2 钱包表（user_wallets）

```sql
CREATE TABLE user_wallets (
    id VARCHAR(36) PRIMARY KEY,
    user_id VARCHAR(36) UNIQUE NOT NULL,
    balance DECIMAL(10, 2) DEFAULT 0.00,
    currency VARCHAR(10) DEFAULT 'CNY',
    status VARCHAR(20) DEFAULT 'normal',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_user_id (user_id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

4.3 交易记录表（wallet_transactions）

```sql
CREATE TABLE wallet_transactions (
    id VARCHAR(36) PRIMARY KEY,
    user_id VARCHAR(36) NOT NULL,
    type VARCHAR(20) NOT NULL, -- recharge/consume/refund
    amount DECIMAL(10, 2) NOT NULL,
    balance_after DECIMAL(10, 2) NOT NULL,
    description VARCHAR(500),
    reference_id VARCHAR(36), -- 关联订单ID或消费记录ID
    payment_method VARCHAR(50), -- 支付方式
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_user_id (user_id),
    INDEX idx_type (type),
    INDEX idx_created_at (created_at),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

5. API接口详细说明

5.1 认证接口

5.1.1 用户创建（管理员）

```
POST /api/v1/admin/users
Authorization: Bearer {token}
Content-Type: application/json

Request:
{
  "email": "test@example.com",
  "tenantId": "tenant_123"  // 租户管理员创建时自动使用自己的租户
}

Response:
{
  "code": 0,
  "message": "用户创建成功，激活邮件已发送",
  "data": {
    "userId": "550e8400-e29b-41d4-a716-446655440000",
    "email": "test@example.com",
    "activationUrl": "https://aiproxy.com/activate?token=xxx"
  }
}
```

5.1.2 首次设置密码

```
POST /api/v1/auth/set-password
Content-Type: application/json

Request:
{
  "token": "activation_token_from_email",
  "password": "Test@123456",
  "confirmPassword": "Test@123456"
}

Response:
{
  "code": 0,
  "message": "密码设置成功",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIs...",
    "expiresIn": 7200,
    "user": {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "email": "test@example.com",
      "role": "user",
      "tenantId": "tenant_123"
    }
  }
}
```

5.1.3 用户登录

```
POST /api/v1/auth/login
Content-Type: application/json

Request:
{
  "email": "test@example.com",
  "password": "Test@123456",  // 首次登录时可不填
  "rememberMe": true
}

Response (已设置密码):
{
  "code": 0,
  "message": "登录成功",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIs...",
    "expiresIn": 7200,
    "requireSetPassword": false,
    "user": {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "email": "test@example.com",
      "role": "user",
      "tenantId": "tenant_123"
    }
  }
}

Response (未设置密码):
{
  "code": 0,
  "message": "请先设置密码",
  "data": {
    "requireSetPassword": true,
    "activationToken": "temp_token_xxx",
    "redirectUrl": "/set-password?token=temp_token_xxx"
  }
}
```

5.1.4 刷新Token

```
POST /api/v1/auth/refresh
Content-Type: application/json

Request:
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
}

Response:
{
  "code": 0,
  "message": "Token刷新成功",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIs...",
    "expiresIn": 7200
  }
}
```

5.2 用户管理接口

5.2.1 获取用户信息

```
GET /api/v1/users/profile
Authorization: Bearer {token}

Response:
{
  "code": 0,
  "data": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "email": "test@example.com",
    "role": "user",
    "tenantId": "tenant_123",
    "wallet": {
      "balance": 1000.00,
      "currency": "CNY",
      "status": "normal"
    },
    "quota": {
      "hasQuotaRules": true,
      "currentHourLimit": -1,      // -1表示无限制
      "todayLimit": 60,
      "monthLimit": 350,
      "todayUsage": 25.50,
      "monthUsage": 180.20
    },
    "createdAt": "2025-08-11T10:00:00Z",
    "lastLoginAt": "2025-08-11T15:30:00Z"
  }
}
```

5.2.2 修改密码

```
POST /api/v1/users/change-password
Authorization: Bearer {token}
Content-Type: application/json

Request:
{
  "oldPassword": "OldPass@123",
  "newPassword": "NewPass@456"
}

Response:
{
  "code": 0,
  "message": "密码修改成功，请重新登录"
}
```

5.3 钱包管理接口

5.3.1 查看钱包余额

```
GET /api/v1/users/wallet
Authorization: Bearer {token}

Response:
{
  "code": 0,
  "data": {
    "userId": "550e8400-e29b-41d4-a716-446655440000",
    "balance": 1000.00,
    "currency": "CNY",
    "status": "normal"
  }
}
```

5.3.2 账户充值

```
POST /api/v1/users/wallet/recharge
Authorization: Bearer {token}
Content-Type: application/json

Request:
{
  "amount": 100.00,
  "paymentMethod": "alipay"  // alipay/wechat/bank
}

Response:
{
  "code": 0,
  "message": "充值成功",
  "data": {
    "transactionId": "tx_123456",
    "amount": 100.00,
    "newBalance": 1100.00
  }
}
```

5.3.3 查看交易记录

```
GET /api/v1/users/wallet/transactions?page=1&limit=20
Authorization: Bearer {token}

Response:
{
  "code": 0,
  "data": {
    "transactions": [
      {
        "id": "tx_123456",
        "type": "recharge",
        "amount": 100.00,
        "balanceAfter": 1100.00,
        "description": "账户充值",
        "paymentMethod": "alipay",
        "createdAt": "2025-08-11T10:00:00Z"
      },
      {
        "id": "tx_123457",
        "type": "consume",
        "amount": -50.00,
        "balanceAfter": 1050.00,
        "description": "API调用消费",
        "referenceId": "api_call_123",
        "createdAt": "2025-08-11T11:00:00Z"
      }
    ],
    "total": 50,
    "page": 1,
    "limit": 20
  }
}
```

6. 错误码定义

错误码	说明	HTTP状态码
10001	邮箱已注册	400

错误码	说明	HTTP状态码
10002	密码强度不足	400
10003	邮箱或密码错误	401
10004	账号未激活	401
10005	账号已锁定	401
10006	Token无效	401
10007	Token已过期	401
10008	权限不足	403
10009	用户不存在	404
10010	原密码错误	400
10011	登录失败次数过多	429
10012	余额不足	400
10013	充值金额无效	400
10014	钱包状态异常	400

7. 性能优化

7.1 缓存策略

- 用户信息缓存（Redis，TTL=1小时）
- 权限信息缓存（Redis，TTL=30分钟）
- Session缓存（Redis，TTL=2小时）

7.2 查询优化

- 用户表索引优化
- 分页查询优化
- 批量操作优化

7.3 安全优化

- 密码哈希异步处理
- Token验证缓存
- 限流和熔断机制

8. 监控指标

8.1 业务指标

- 注册用户数
- 活跃用户数（DAU/MAU）
- 登录成功率
- 平均会话时长

8.2 性能指标

- 登录响应时间
- Token验证延迟
- 数据库查询性能

8.3 安全指标

- 登录失败次数
- 异常登录检测
- 账号锁定数量

9. 测试用例

9.1 功能测试

- 用户注册流程测试
- 多种登录方式测试
- 权限控制测试
- 会话管理测试

9.2 性能测试

- 并发登录测试
- Token验证性能测试
- 大量用户查询测试

9.3 安全测试

- SQL注入测试
- XSS攻击测试
- 暴力破解测试
- 权限越权测试

文档版本：v1.0.0
更新日期：2025-08-11