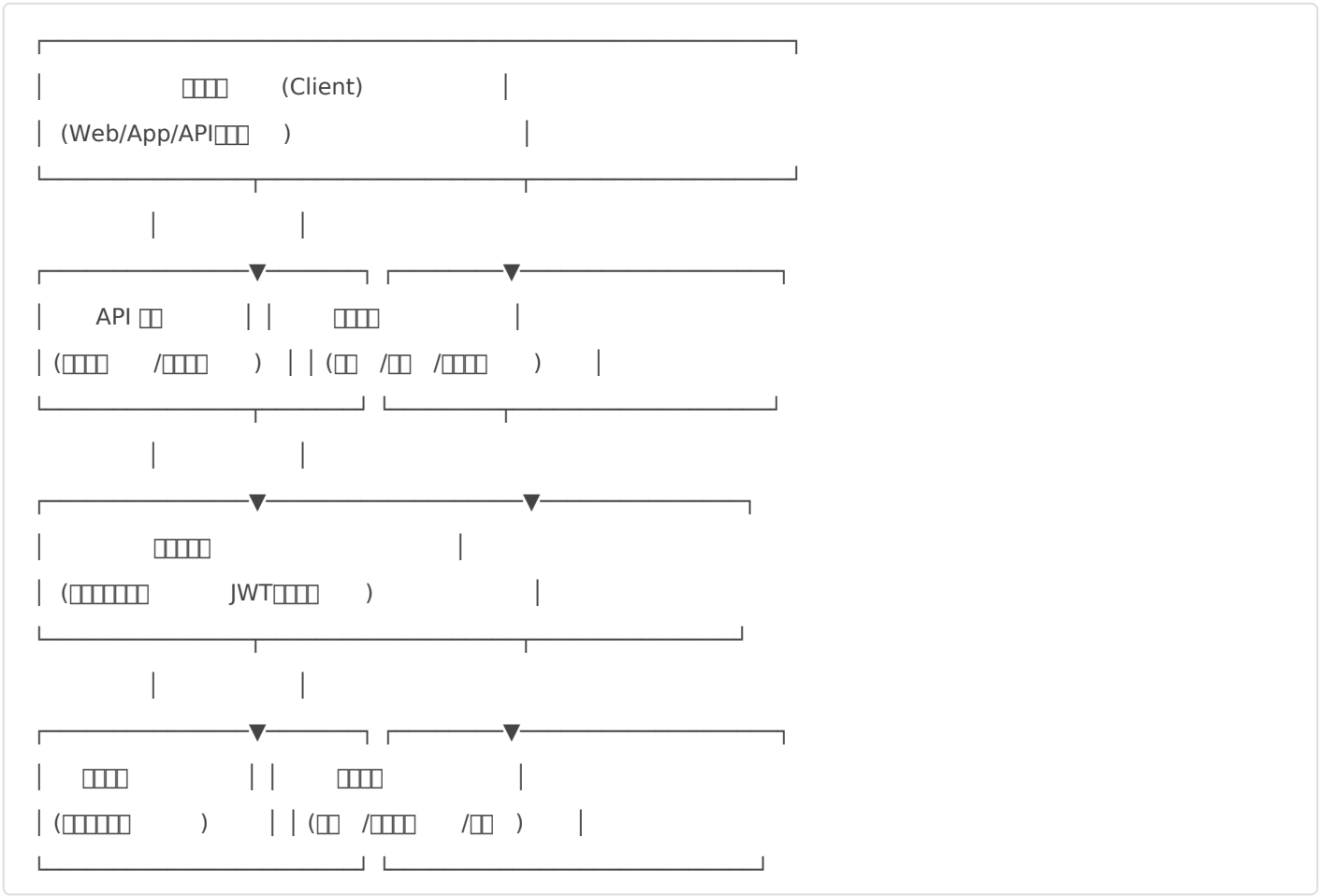

PHP 微服务架构下的用户认证与权限系统设计

1. 整体架构设计

架构图

```
┌─────────────────────────────────────────────────────────┐
│                  客户端层  (Client)                    │
│  (Web/App/API调用方)                                    │
└────────────────┬───────────────────┬────────────────────┘
                 │                   │
┌────────────────▼─────┐ ┌───────────▼───────────────────┐
│    API 网关          │ │        认证服务               │
│ (身份验证/路由转发)  │ │ (登录/注册/令牌签发)          │
└────────────────┬─────┘ └──────────┬────────────────────┘
                 │                   │
┌────────────────▼────────────────────▼─────────────────┐
│            微服务集群                                 │
│  (各业务服务通过JWT验证权限)                          │
└────────────────┬───────────────────┬──────────────────┘
                 │                   │
┌────────────────▼─────┐ ┌───────────▼───────────────────┐
│     用户服务         │ │       权限服务                │
│ (用户数据管理)       │ │ (角色/权限分配/验证)          │
└──────────────────────┘ └───────────────────────────────┘
```

2. □□□□□

2.1 JWT □□□

□□□□ :

```
composer require firebase/php-jwt lcobucci/jwt
```

JWT □□□ :

```
<?php
namespace App\Services\Auth;

use Firebase\JWT\JWT;
use Firebase\JWT\Key;

class JwtService
{
    private $secretKey;
    private $algorithm = 'HS256';

    public function __construct(string $secretKey) {
        $this->secretKey = $secretKey;
    }

    public function generateToken(array $payload, int $expire = 3600): string
    {
        $issuedAt = time();
        $payload += [
            'iat' => $issuedAt,
            'exp' => $issuedAt + $expire,
            'iss' => 'your-auth-service',
            'aud' => 'your-microservices'
        ];

        return JWT::encode($payload, $this->secretKey, $this->algorithm);
    }
```

```
    public function validateToken(string $token): ?array
    {
        try {
            $decoded = JWT::decode($token, new Key($this->secretKey, $this->algorithm));
            return (array)$decoded;
        } catch (\Exception $e) {
            return null;
        }
    }
}
```

2.2 权限服务设计

RBAC (基于角色的访问控制 **)** 数据库结构 :

```
CREATE TABLE `users` (
  `id` bigint PRIMARY KEY AUTO_INCREMENT,
  `username` varchar(255) UNIQUE NOT NULL,
  `password_hash` varchar(255) NOT NULL
);

CREATE TABLE `roles` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `name` varchar(255) UNIQUE NOT NULL,
  `description` varchar(255)
);

CREATE TABLE `user_roles` (
  `user_id` bigint,
  `role_id` int,
  PRIMARY KEY (`user_id`, `role_id`)
);

CREATE TABLE `permissions` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `name` varchar(255) UNIQUE NOT NULL,
  `description` varchar(255)
);
```

```
CREATE TABLE `role_permissions` (
  `role_id` int,
  `permission_id` int,
  PRIMARY KEY (`role_id`, `permission_id`)
);
```

□□□□□□ :

```
<?php
namespace App\Services\Auth;

class PermissionService
{
    private $pdo;

    public function __construct(\PDO $pdo) {
        $this->pdo = $pdo;
    }

    public function userHasPermission(int $userId, string $permission): bool
    {
        $stmt = $this->pdo->prepare("
            SELECT COUNT(*)
            FROM user_roles ur
            JOIN role_permissions rp ON ur.role_id = rp.role_id
            JOIN permissions p ON rp.permission_id = p.id
            WHERE ur.user_id = :userId AND p.name = :permission
        ");

        $stmt->execute([
            ':userId' => $userId,
            ':permission' => $permission
        ]);

        return $stmt->fetchColumn() > 0;
    }

    public function getUserPermissions(int $userId): array
    {
```

```
        $stmt = $this->pdo->prepare("
            SELECT p.name
            FROM user_roles ur
            JOIN role_permissions rp ON ur.role_id = rp.role_id
            JOIN permissions p ON rp.permission_id = p.id
            WHERE ur.user_id = :userId
        ");

        $stmt->execute([':userId' => $userId]);
        return $stmt->fetchAll(\PDO::FETCH_COLUMN, 0);
    }
}
```

3. API □□□

3.1 □□□□□□

```
<?php
namespace App\Middleware;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\MiddlewareInterface;
use Psr\Http\Server\RequestHandlerInterface;

class AuthMiddleware implements MiddlewareInterface
{
    private $jwtService;
    private $permissionService;

    public function __construct(JwtService $jwtService, PermissionService $permissionService) {
        $this->jwtService = $jwtService;
        $this->permissionService = $permissionService;
    }

    public function process(ServerRequestInterface $request, RequestHandlerInterface $handler):
```

```
ResponseInterface
    {
        // 1. 获取  Token
        $token = $this->getTokenFromRequest($request);

        if (!$token) {
            return new JsonResponse(['error' => 'Token missing'], 401);
        }

        // 2. 验证  Token
        $payload = $this->jwtService->validateToken($token);

        if (!$payload) {
            return new JsonResponse(['error' => 'Invalid token'], 401);
        }

        // 3. 检查权限      (从路由获取所需权限            )
        $requiredPermission = $request->getAttribute('route')->getArgument('permission');

        if ($requiredPermission &&
            !$this->permissionService->userHasPermission($payload['user_id'], $requiredPermission)) {
            return new JsonResponse(['error' => 'Forbidden'], 403);
        }

        // 4. 将用户信息添加到请求属性
        $request = $request->withAttribute('auth_user', $payload);

        return $handler->handle($request);
    }

    private function getTokenFromRequest(ServerRequestInterface $request): ?string
    {
        $header = $request->getHeaderLine('Authorization');

        if (preg_match('/Bearer\s(\S+)/', $header, $matches)) {
            return $matches[1];
        }

        return $request->getQueryParams()['token'] ?? null;
```

```
    }
}
```

4. □□□□□

4.1 □□□□

```
<?php
namespace App\Controller\Auth;

use App\Services\Auth\JwtService;
use App\Repository\UserRepository;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

class LoginController
{
    private $userRepository;
    private $jwtService;

    public function __construct(UserRepository $userRepository, JwtService $jwtService) {
        $this->userRepository = $userRepository;
        $this->jwtService = $jwtService;
    }

    public function login(ServerRequestInterface $request): ResponseInterface
    {
        $data = json_decode($request->getBody()->getContents(), true);

        // 1. □□□□
        if (empty($data['username']) || empty($data['password'])) {
            return new JsonResponse(['error' => 'Invalid credentials'], 400);
        }

        // 2. □□□□
        $user = $this->userRepository->findByUsername($data['username']);
```

```
        if (!$user || !password_verify($data['password'], $user['password_hash'])) {
            return new JsonResponse(['error' => 'Invalid credentials'], 401);
        }

        // 3. 生成 JWT
        $token = $this->jwtService->generateToken([
            'user_id' => $user['id'],
            'username' => $user['username']
        ]);

        // 4. 返回响应
        return new JsonResponse([
            'token' => $token,
            'expires_in' => 3600,
            'token_type' => 'Bearer'
        ]);
    }
}
```

5. 微服务集成方案

5.1 服务间认证

使用服务账户 :

```
// 在服务启动时获取服务令牌
$serviceToken = $jwtService->generateToken([
    'service_name' => 'order-service',
    'scope' => ['internal']
], 86400 * 365); // 长期有效的服务令牌

// 在服务间调用时携带
$response = $httpClient->request('GET', 'http://user-service/api/users', [
    'headers' => [
        'Authorization' => 'Bearer ' . $serviceToken,
        'X-Service-Auth' => 'order-service'
```

```
    ]
]);
```

5.2 权限缓存优化

Redis缓存权限 :

```
class CachedPermissionService extends PermissionService
{
    private $redis;
    private $ttl = 3600;

    public function __construct(\PDO $pdo, \Redis $redis) {
        parent::__construct($pdo);
        $this->redis = $redis;
    }

    public function getUserPermissions(int $userId): array
    {
        $cacheKey = "user_permissions:$userId";

        if ($this->redis->exists($cacheKey)) {
            return json_decode($this->redis->get($cacheKey), true);
        }

        $permissions = parent::getUserPermissions($userId);
        $this->redis->setex($cacheKey, $this->ttl, json_encode($permissions));

        return $permissions;
    }
}
```

6. 安全增强措施

6.1 令牌黑名单

```
class JwtServiceWithBlacklist extends JwtService
{
    private $redis;

    public function __construct(string $secretKey, \Redis $redis) {
        parent::__construct($secretKey);
        $this->redis = $redis;
    }

    public function invalidateToken(string $token, int $expire): void
    {
        $payload = $this->validateToken($token);
        if ($payload) {
            $jti = $payload['jti'] ?? md5($token);
            $this->redis->setex("jwt_blacklist:$jti", $expire, '1');
        }
    }

    public function validateToken(string $token): ?array
    {
        $payload = parent::validateToken($token);

        if ($payload) {
            $jti = $payload['jti'] ?? md5($token);
            if ($this->redis->exists("jwt_blacklist:$jti")) {
                return null;
            }
        }

        return $payload;
    }
}
```

6.2 □□□

```
class RateLimitMiddleware implements MiddlewareInterface
{
    private $redis;
```

```
    public function __construct(\Redis $redis) {
        $this->redis = $redis;
    }

    public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface
    {
        $ip = $request->getServerParams()['REMOTE_ADDR'];
        $key = "rate_limit:$ip";

        $current = $this->redis->incr($key);
        if ($current === 1) {
            $this->redis->expire($key, 60);
        }

        if ($current > 100) {
            return new JsonResponse(['error' => 'Too many requests'], 429);
        }

        return $handler->handle($request);
    }
}
```

7. 部署建议

1. 认证服务独立部署：将认证服务部署为独立微服务
2. 多区域部署：在多个区域部署认证服务以减少延迟
3. 数据库分片：用户数据按地区或ID范围分片
4. 监控：监控认证服务的成功率、延迟和错误率
5. 灾备：建立多活数据中心，确保认证服务高可用

8. 性能优化

1. 使用**OPcache**:PHP配置中启用OPcache
2. 连接池：数据库和Redis使用连接池
3. 异步日志：使用异步方式记录审计日志
4. 热点缓存：缓存频繁访问的用户权限数据

5. **JWT预验证**：在API网关层先验证JWT有效性

这种架构设计提供了灵活的认证和授权机制，能够支持高并发的微服务系统，同时保持了良好的安全性和可扩展性。

Revision #1
Created 26 May 2025 05:36:34 by Admin
Updated 26 May 2025 05:36:50 by Admin