EdgeOne技术教程

注册并添加站点

登录EdgeOne控制台，添加您的网站域名

使用案例

实际业务场景中的EdgeOne应用案例

电商网站加速

为全球电商平台提供加速和安全防护，提升用户购物体验。

商品图片CDN加速
API接口智能加速
DDoS攻击防护
Bot流量管理

效果展示

页面加载速度 提升65%
图片加载时间 减少40%
API响应时间 优化50%

视频流媒体平台

为视频平台提供全球加速和智能调度，保证观看体验。

HLS/DASH自适应
视频转码加速
防盗链保护
带宽成本优化

核心指标

视频首屏时间 <2秒
卡顿率 <0.5%
带宽节省 30%

移动应用后端

为移动应用提供API加速和安全防护，提升用户体验。

```
// 移动端接入示例
const apiUrl = 'https://api.example.com.eo.dnse5.com';
const response = await fetch(`${apiUrl}/user/profile`, {
    method: 'GET',
    headers: {
        'Host': 'api.example.com',
        'Authorization': 'Bearer ' + token
    }
});
```

游戏加速服务

为在线游戏提供低延迟加速和DDoS防护。

UDP加速优化
智能路由选择
游戏DDoS防护
反外挂检测

集成代码示例

Go语言集成

```
package main

import (
    "fmt"
    "net/http"
    "io/ioutil"
    "time"
)

func main() {
    start := time.Now()
    body, _ := DoRequest("https://api.example.com.eo.dnse5.com/ping")
    elapsed := time.Since(start)
    fmt.Println("Request with EdgeOne, use", elapsed)
    fmt.Println("Response body:", body)
}

func DoRequest(url string) (string, error) {
    client := &http.Client{}
    request, err := http.NewRequest(http.MethodGet, url, nil)
    if err != nil {
        return "", err
    }

    // 设置Host头
    request.Host = "api.example.com"
```

```
    rsp, err := client.Do(request)
    if err != nil {
        return "", err
    }
    defer rsp.Body.Close()

    body, err := ioutil.ReadAll(rsp.Body)
    return string(body), err
}
```

JavaScript集成

```
// 使用EdgeOne加速的API请求
class EdgeOneClient {
    constructor(baseUrl, host) {
        this.baseUrl = baseUrl;
        this.host = host;
    }

    async request(endpoint, options = {}) {
        const url = `${this.baseUrl}${endpoint}`;
        const headers = {
            'Host': this.host,
            'Content-Type': 'application/json',
            ...options.headers
        };

        const response = await fetch(url, {
            ...options,
            headers
        });

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        return response.json();
    }
}

// 使用示例
const client = new EdgeOneClient(
    'https://api.example.com.eo.dnse5.com',
    'api.example.com'
);

// 发起请求
const data = await client.request('/user/profile', {
    method: 'GET'
});
```

Pages 模板

快速部署静态网站和前端应用的模板集合

EdgeOne Pages

EdgeOne Pages是一个类似于Vercel和Cloudflare Pages的静态网站托管平台，支持自动部署、边缘函数和全球加速。

Next.js 模板

基于Next.js的现代Web应用模板，支持SSR和SSG。

TypeScript支持
Tailwind CSS
自动部署

```
# 快速开始
npx create-next-app@latest my-app
cd my-app
npm run dev

# 部署到EdgeOne Pages
git push origin main
```

Vue.js 模板

基于Vue 3的单页应用模板，包含路由和状态管理。

Vue 3 + Vite
Vue Router
Pinia状态管理

```
# 创建Vue项目
npm create vue@latest my-vue-app
cd my-vue-app
npm install
npm run dev
```

```
# 构建部署
npm run build
```

Angular 模板

企业级Angular应用模板，包含完整的开发工具链。

Angular CLI
RxJS
Material UI

```
# 创建Angular项目
ng new my-angular-app
cd my-angular-app
ng serve

# 构建生产版本
ng build --prod
```

Hexo 博客

静态博客生成器，支持Markdown和主题定制。

Markdown支持
主题丰富
插件生态

```
# 安装Hexo
npm install -g hexo-cli
hexo init my-blog
cd my-blog
npm install

# 本地预览
hexo server

# 生成静态文件
hexo generate
```

文档站点

基于VitePress的技术文档站点模板。

VitePress
全文搜索
多语言支持

```
# 创建文档站点
npx create-vitepress my-docs
cd my-docs
npm install

# 开发模式
npm run dev

# 构建文档
npm run build
```

电商模板

现代电商网站模板，包含完整的购物功能。

商品展示
购物车
支付集成

```
# 克隆电商模板
git clone https://github.com/example/ecommerce-template
cd ecommerce-template
npm install

# 配置环境变量
cp .env.example .env
npm run dev
```

部署配置

GitHub Actions配置

```
# .github/workflows/deploy.yml
name: Deploy to EdgeOne Pages
```

```
on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build

      - name: Deploy to EdgeOne Pages
        uses: TencentEdgeOne/pages-action@v1
        with:
          api-token: ${{ secrets.EDGEONE_API_TOKEN }}
          project-name: my-project
          directory: ./dist
```

边缘函数开发

在边缘节点运行自定义代码，实现高性能计算

边缘函数特性

EdgeOne边缘函数提供Serverless代码执行环境，支持JavaScript/TypeScript，在全球边缘节点运行，延迟极低。

基础示例

```
// 简单的边缘函数示例
export default {
  async fetch(request, env, ctx) {
    // 获取请求信息
    const url = new URL(request.url);
    const userAgent = request.headers.get('User-Agent');

    // 根据用户设备返回不同内容
    if (userAgent.includes('Mobile')) {
      return new Response('Mobile Version', {
        headers: { 'Content-Type': 'text/plain' }
      });
    }

    return new Response('Desktop Version', {
      headers: { 'Content-Type': 'text/plain' }
    });
  }
};
```

请求处理

```
// 处理不同HTTP方法
export default {
  async fetch(request, env, ctx) {
    const url = new URL(request.url);
    const method = request.method;

    switch (method) {
      case 'GET':
        return handleGet(request, url);
      case 'POST':
        return handlePost(request, url);
      case 'PUT':
        return handlePut(request, url);
      default:
        return new Response('Method Not Allowed', {
          status: 405
        });
    }
  }
};

async function handleGet(request, url) {
  // 处理GET请求
  return new Response(JSON.stringify({
    message: 'Hello from EdgeOne',
    timestamp: new Date().toISOString(),
    path: url.pathname
  }), {
    headers: {
      'Content-Type': 'application/json',
      'Cache-Control': 'max-age=300'
    }
  });
```

```
}
```

高级功能示例

地理位置路由

```
// 基于地理位置的内容路由
export default {
  async fetch(request, env, ctx) {
    const country = request.cf?.country || 'US';
    const url = new URL(request.url);

    // 根据国家重定向到不同的源站
    const regionMapping = {
      'CN': 'https://china.example.com',
      'US': 'https://us.example.com',
      'EU': 'https://eu.example.com'
    };

    let targetOrigin = regionMapping[country] || regionMapping['US'];

    // 如果是欧盟国家，使用EU服务器
    const euCountries = ['DE', 'FR', 'GB', 'IT', 'ES'];
    if (euCountries.includes(country)) {
      targetOrigin = regionMapping['EU'];
    }

    // 构建新的请求URL
    const newUrl = new URL(url.pathname + url.search, targetOrigin);

    // 发起请求到目标源站
    const response = await fetch(newUrl, {
      method: request.method,
      headers: request.headers,
      body: request.body
    });

    // 添加地理位置信息到响应头
    const newResponse = new Response(response.body, response);
    newResponse.headers.set('X-Country', country);
    newResponse.headers.set('X-Origin', targetOrigin);

    return newResponse;
  }
};
```

身份验证中间件

```
// JWT身份验证边缘函数
import { verify } from 'jsonwebtoken';

export default {
  async fetch(request, env, ctx) {
    const url = new URL(request.url);

    // 检查是否需要身份验证
    const protectedPaths = ['/api/user', '/api/admin'];
    const needsAuth = protectedPaths.some(path =>
      url.pathname.startsWith(path)
    );

    if (needsAuth) {
      const authResult = await authenticate(request, env);
      if (!authResult.success) {
        return new Response(JSON.stringify({
          error: 'Unauthorized',
          message: authResult.message
        }), {
          status: 401,
          headers: { 'Content-Type': 'application/json' }
        });
      }

      // 将用户信息添加到请求头
      request.headers.set('X-User-ID', authResult.userId);
      request.headers.set('X-User-Role', authResult.role);
    }

    // 继续处理请求
    return fetch(request);
  }
};

async function authenticate(request, env) {
  const authHeader = request.headers.get('Authorization');

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return { success: false, message: 'Missing or invalid authorization header' };
  }

  const token = authHeader.substring(7);

  try {
    const payload = verify(token, env.JWT_SECRET);
    return {
      success: true,
      userId: payload.sub,
      role: payload.role
```

```
    };
  } catch (error) {
    return { success: false, message: 'Invalid token' };
  }
}
```

API限流

```
// 基于IP的API限流
export default {
  async fetch(request, env, ctx) {
    const clientIP = request.headers.get('CF-Connecting-IP') ||
                     request.headers.get('X-Forwarded-For') ||
                     'unknown';

    // 检查限流
    const rateLimitResult = await checkRateLimit(clientIP, env);

    if (!rateLimitResult.allowed) {
      return new Response(JSON.stringify({
        error: 'Rate limit exceeded',
        retryAfter: rateLimitResult.retryAfter
      }), {
        status: 429,
        headers: {
          'Content-Type': 'application/json',
          'Retry-After': rateLimitResult.retryAfter.toString()
        }
      });
    }

    // 继续处理请求
    const response = await fetch(request);

    // 添加限流信息到响应头
    response.headers.set('X-RateLimit-Limit', '100');
    response.headers.set('X-RateLimit-Remaining', rateLimitResult.remaining.toString());
    response.headers.set('X-RateLimit-Reset', rateLimitResult.resetTime.toString());

    return response;
  }
};

async function checkRateLimit(clientIP, env) {
  const key = `rate_limit:${clientIP}`;
  const windowSize = 60; // 1分钟窗口
  const maxRequests = 100; // 最大请求数

  // 这里可以使用KV存储或其他持久化方案
  // 简化示例使用内存存储
  const now = Math.floor(Date.now() / 1000);
  const windowStart = now - windowSize;

  // 获取当前窗口内的请求数
  const currentCount = await getCurrentCount(key, windowStart);

  if (currentCount >= maxRequests) {
    return {
      allowed: false,
      retryAfter: windowSize - (now % windowSize)
    };
  }

  // 记录当前请求
  await recordRequest(key, now);

  return {
    allowed: true,
    remaining: maxRequests - currentCount - 1,
    resetTime: windowStart + windowSize
  };
}
```

开发工具链

本地开发

```
# 安装EdgeOne CLI
npm install -g @edgeone/cli

# 创建新项目
edgeone init my-function
cd my-function

# 本地开发服务器
edgeone dev

# 部署到EdgeOne
edgeone deploy
```

测试配置

```
// jest.config.js
module.exports = {
  testEnvironment: 'node',
  setupFilesAfterEnv: ['/test/setup.js'],
```

```
    testMatch: ['**/__tests__/**/*.test.js'],
    collectCoverageFrom: [
      'src/**/*.js',
      '!src/**/*.test.js'
    ]
  };

  // 单元测试示例
  import { handleRequest } from '../src/index.js';

  test('should return mobile version for mobile user agent', async () => {
    const request = new Request('https://example.com', {
      headers: { 'User-Agent': 'Mobile Safari' }
    });

    const response = await handleRequest(request);
    const text = await response.text();

    expect(text).toBe('Mobile Version');
  });
```