移动解析 HTTPDNS

API 文档

腾讯云

文档目录

API 文档
配置信息说明

最近更新时间：2025-05-06 17:52:52

概述

在接入移动解析 HTTPDNS 过程中，您需在移动解析 HTTPDNS 控制台获取对应配置信息后才可正常接入，本文将对如何获取配置信息以及配置信息含义进行说明。

前提条件

已开通移动解析 HTTPDNS。具体操作请参见 开通移动解析 HTTPDNS 。

操作指南

登录**移动解析 HTTPDNS 控制台** > 开发配置 页面，即可查询到您的配置信息。如下图所示：

- **授权 ID**：使　移动解析 HTTPDNS 服务中，开发配置的唯　标识。调　移动解析 HTTPDNS 的 HTTP 解析接口 `http://119.29.29.98` 时中传　的授权 ID 参数。

- **DES 加密密钥**：调　移动解析 HTTPDNS 的 HTTP 解析接口 `http://119.29.29.98` 并使用 DES 加密方式时，对 DNS 请求数据进　加密时的加密密钥。
- **AES 加密密钥**：调　移动解析 HTTPDNS 的 HTTP 解析接口 `http://119.29.29.98` 并使用 AES 加密方式时，对 DNS 请求数据进　加密时的加密密钥。
- **HTTPS 加密 Token**：调　移动解析 HTTPDNS 的 HTTPS 解析接口 `https://119.29.29.99` ，对 DNS 请求数据进　鉴权的 Token 信息。

HTTP 请求方式查询

最近更新时间：2025-05-22 14:38:42

概述

移动解析 HTTPDNS 通过 HTTP/HTTPS 接口对外提供域名解析服务，服务接入直接使用 IP 地址，移动解析 HTTPDNS 的 HTTP 请求方式查询入口以 `119.29.29.98` 为例。

说明：

- 我们提供2个入口 IP 示例，HTTP 协议的服务 IP：`119.29.29.98`，HTTPS 协议的服务 IP：`119.29.29.99`。
- 请优先使用官方 SDK，如果场景特殊下无法使用 SDK，需要直接访问 HTTP API 接口，请加入 技术支持群 或者 提交工单 联系我们，我们将根据您的具体使用场景，为您提供相关的安全建议。
- 移动解析 HTTPDNS 接入的是 BGP Anycast 网络架构，对外提供的是Anycast IP。Anycast IP 是允许多个服务器共享同一个 IP 地址，具有高可用、负载均衡、减少网络延迟等优势。

前期准备

- 开通移动解析 HTTPDNS 服务，详情请参见 开通移动解析 HTTPDNS 服务。
- 在控制台添加需要解析的域名，详情请参见 添加域名。
- 获取配置信息，详情请参见 配置信息说明。

接口描述

- 接口请求地址：`http://119.29.29.98/d? + {请求参数}`。
- 请求方式：POST 或 GET。
- 入口 IP 的切换逻辑：当接入 IP 访问超时，或者返回的结果非 IP 格式，或者返回为空的时候，请采用其他入口 IP 接入，若所有 IP 均出现异常，请兜底至 LocalDNS 进行域名解析。

请求参数

参数名	参数含义	是否必选	取值	加密	说明
dn	被查询的域名	是	加密前的单个域名长度为253	是	需在移动解析 HTTPDNS 控制台已添加域名并且为传输加密后的字符串。 域名添加请参见 添加域名。

					加密详情请参见 加密与解密算法使用说明 。
id	用户标识（即授权 ID）	是	1 – 100000	否	如果使用 AES、DES 加密方式，必须传入授权 ID，不需要进行加密。
alg	选择使用何种算法	是	[aes/des]	否	默认使用 DES 算法，不同算法具有不同密钥。
ip	DNS 请求的 ECS（EDNS-Client-Subnet）值	否	IPv4/IPv6 地址值	是	默认情况下 HTTPDNS 服务器会查询客户端出口 IP 为 DNS 线路查询 IP，使用“ip=xxx” 参数，可以指定线路 IP 地址。支持 IPv4/IPv6 地址传入，接口会自动识别。加密详情请参见 加密与解密算法使用说明 。
query	结果中返回被查询域名	否	1	否	单域名查询情况下，此参数要求返回结果中携带被查询域名。批量域名查询情况下，此参数固定取值为1。
timeout	超时返回时间	否	1000 – 5000，单位为毫秒	否	可用值[1000, 5000]，单位为ms，查询超时时间，默认值为5秒。
ttl	查询结果是否返回 TTL 值	否	1	否	可用值 [1]，不携带此参数，默认为不传递TTL值。
type	查询类型	否	[aaaa/AAAA/addrs/ADDRS]	否	可用值 [aaaa,AAAA,addrs,ADDRS]。默认查询 A 记录，设置 AAAA/aaaa 查询 AAAA 记录，设置 addrs/ADDRS 同时查询 A 和 AAAA 记录。
clientip	查询结果中返回的客户端 IP 地址	否	1	否	可用值 [1]，不携带此参数，默认为不传递clientip 值。若此参数取值，则返回结果中地址值在 | 符号后，若携带有 ip 参数，返回的是 ip 参数的值，否则返回客户端地址 IP。

说明：
ECS（EDNS-Client-Subnet）协议在 DNS 请求包中附加请求域名解析的用户 IP 地址，DNS 服务器可以根据该地址返回用户更快速访问的服务器 IP 地址。

请求说明

以 ID 为 `xxx` 为例。

> **注意：**
> - 以下示例为 DES 加密方式，其中域名和 IP 参数均需要加密，例如，域名为 `cloud.tencent.com` 需要进行加密，授权 ID 不需要进行加密。
> - 若 HTTPDNS 未查询到解析结果，将返回为空值。
> - HTTP 已接入 BGP Anycast，并实现多地机房容灾，但为了服务质量更高的保障，建议您采用 Failed over 策略 进行接入。

请求 A 记录

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。

返回结果中携带 ttl 信息

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符
串}&id=xxx&ttl=1"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6,120
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。记录值与 ttl 值以 ',' 分隔。

返回结果携带查询线路 IP 地址

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx&clientip=1&ip={DNS 请求的 ECS 值加密后字符串}&ttl=1"
```

- **解密后返回格式：**

```
12.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
```

- **格式说明：**返回结果中携带线路 ip 地址，以'|'分隔。如果没有传入 “ip=xxx” 参数，则返回出口 IP 地址；否则返回 ip 参数中的地址。

同时请求 A 和 AAAA 记录

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx&clientip=1&ip={DNS 请求的 ECS 值加密后字符串}&type=addrs&ttl=1"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6,120-2402:4e00:0123:4567:0::2345;2403:4e00:0123:4567:0::2346,120|1.2.3.4
```

- **格式说明：**A 记录和 AAAA 记录之间以 '-' 分隔，A 记录在前，AAAA 记录在后。

返回结果中携带被查询域名

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx&clientip=1&ip={DNS 请求的 ECS 值加密后字符串}&query=1&ttl=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:2.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
```

- **格式说明：**返回格式为 “域名.:结果” 的格式。

批量域名请求

- **输入示例：**

```
curl "http://119.29.29.98/d?dn=
{cloud.tencent.com,www.qq.com,www.dnspod.cn 加密后字符
串}&id=xxx&clientip=1&ip={DNS 请求的 ECS 值加密后字符串}&ttl=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:2.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
www.qq.com.:3.3.3.4;3.3.3.5;3.3.3.6,180|1.2.3.4
www.dnspod.cn.:4.3.3.4;4.3.3.5;4.3.3.6,60|1.2.3.4
```

- **格式说明：**多个域名返回内容之间以 “换行符” 分隔，ip 地址附加在所有记录值的最后。

POST 方法请求 A 记录

- **输入示例：**

```
curl "http://119.29.29.98/d" -d "dn={cloud.tencent.com 加密后字符
串}&id=xxx" -X POST
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。

请求异常或无记录说明

> ⚠ **注意：**
>
> - 以下示例为 DES 加密方式，其中域名和 IP 参数均需要加密，例如域名为 `cloud.tencent.com` 需要加密，授权 ID 不需要进行加密。
> - 如使用 HTTPS 加密方式，请求地址改为 `119.29.29.99` 并且必须要传入 token。

查询 A 记录

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx"
```

- **解密后返回格式：**空字符串或者0。

- **格式说明：** 没有记录，则返回0。未在控制台添加主域名，返回 HTTP 状态码401和空字符。

返回结果中包含域名

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx&query=1&ip={DNS 请求的 ECS 值加密后字符串}&clientip=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:0|1.2.3.4
```

- **格式说明：** 0表示没有记录。

返回 A 与 AAAA 的记录

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com 加密后字符串}&id=xxx&type=addrs&query=1&ip={DNS 请求的 ECS 值加密后字符串}&clientip=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:0-0|1.2.3.4
```

- **格式说明：** 0表示没有记录。如果某个记录存在，则该记录正常返回在结果中，例如 `cloud.tencent.com.:2.3.4.5;3.3.3.3-0|1.2.3.4` ，表示 AAAA 记录无法查询到。

批量域名请求

- **输入示例：**

```
curl "http://119.29.29.98/d?dn={cloud.tencent.com,www.qq.com,www.dnspod.cn 加密后字符串}&id=xxx&clientip=1&ip={DNS 请求的 ECS 值加密后字符串}&ttl=1"
```

- **解密返回格式：**

```
cloud.tencent.com.:0,0|1.2.3.4
```

```
www.qq.com.:3.3.3.4;3.3.3.5;3.3.3.6,180|1.2.3.4
www.dnspod.cn.:4.3.3.4;4.3.3.5;4.3.3.6,60|1.2.3.4
```

- **格式说明：**未查询到数据的域名则返回0。如果某个记录存在，则该记录正常返回在结果中。

HTTP 状态码

以下为接口业务逻辑相关的 HTTP 状态码。

状态码	描述
200 OK	如果接口调用正确，无论是否查询成功，均返回状态码200。
401 Unauthorized	请求的域名未在控制台添加过主域名。
404 Not Found	接口不存在或 URL 实际上访问了某不存在的资源。
429 Too Many Request	访问过于频繁，超过了服务器限制。
501 Not Implemented	使用了非 “GET” 或 “POST” 请求方式。

HTTPS 请求方式查询

最近更新时间：2025-05-06 17:52:52

概述

移动解析 HTTPDNS 通过 HTTP/HTTPS 接口对外提供域名解析服务，服务接入直接使用 IP 地址，移动解析 HTTPDNS 的 HTTPS 请求方式查询入口以 `119.29.29.99` 为例。

说明：

- 我们提供2个入口 IP 示例，HTTP 协议的服务 IP：`119.29.29.98`，HTTPS 协议的服务 IP：`119.29.29.99`。
- 请优先使用官方 SDK，如果场景特殊下无法使用 SDK，需要直接访问 HTTPS API 接口，请加入 技术支持群 或者 提交工单 联系我们，我们将根据您的具体使用场景，为您提供相关的安全建议。
- 移动解析 HTTPDNS 接入的是 BGP Anycast 网络架构，对外提供的是 Anycast IP。Anycast IP 是允许多个服务器共享同一个 IP 地址，具有高可用、负载均衡、减少网络延迟等优势。

前期准备

- 开通移动解析 HTTPDNS 服务，详情请参见 开通移动解析 HTTPDNS 服务。
- 在控制台添加需要解析的域名，详情请参见 添加域名。
- 获取配置信息，详情请参见 配置信息说明。

接口描述

- 接口请求地址：`https://119.29.29.99/d? + {请求参数}`。
- 请求方式：POST 或 GET。
- 入口 IP 的切换逻辑：当接入 IP 访问超时，或者返回的结果非 IP 格式，或者返回为空的时候，请采用其他入口 IP 接入，若所有 IP 均出现异常，请兜底至 LocalDNS 进行域名解析。

请求参数

参数名	参数含义	是否必选	取值	加密	说明
dn	被查询的域名	是	字符串	否	需在移动解析 HTTPDNS 控制台已添加域名。具体请参见 添加域名。

token	使用 HTTPS 方式的标识	是	整型数据	否	token 获取请参见 配置信息说明 。
ip	DNS 请求的 ECS（EDNS-Client-Subnet）值	否	IPv4/IPv6 地址值	否	默认情况下 HTTPDNS 服务器会查询客户端出口 IP 为 DNS 线路查询 IP，使用“ip=xxx” 参数，可以指定线路 IP 地址。支持 IPv4/IPv6 地址传入，接口会自动识别。
query	结果中返回被查询域名	否	1	否	单域名查询情况下，此参数要求返回结果中携带被查询域名。批量域名查询情况下，此参数固定取值为1。
timeout	超时返回时间	否	1000 – 5000，单位为毫秒	否	可用值[1000, 5000]，单位为ms，查询超时时间，默认值为5秒。
ttl	查询结果是否返回 TTL 值	否	1	否	可用值 [1]，不携带此参数，默认为不传递 TTL值。
type	查询类型	否	[aaaa/AAAA/addrs/ADDRS]	否	可用值 [aaaa,AAAA,addrs,ADDRS]。默认查询 A 记录，设置 AAAA/aaaa 查询 AAAA 记录，设置 addrs/ADDRS 同时查询 A 和 AAAA 记录。
clientip	查询结果中返回的客户端 IP 地址	否	1	否	可用值 [1]，不携带此参数，默认为不传递 clientip 值。若此参数取值，则返回结果中地址值在 | 符号后，若携带有 ip 参数，返回的是 ip 参数的值，否则返回客户端地址 IP。

说明：

- ECS（EDNS-Client-Subnet）协议在 DNS 请求包中附加请求域名解析的用户 IP 地址，DNS 服务器可以根据该地址返回用户更快速访问的服务器 IP 地址。
- 使用 HTTPS 方式，传输的数据会因为 TLS 通道而被加密保护，因此不需要主动对传入的数据额外加密。
- 出于安全和身份认证的考虑，需要传入 HTTPS Token 实现身份鉴权。

请求说明

以请求域名为 `cloud.tencent.com`，token 为 `yyyy` 为例。

> ⚠ **注意：**
> - 若 HTTPDNS 未查询到解析结果，将返回为空值。
> - HTTP 已接入 BGP Anycast，并实现多地机房容灾，但为了服务质量更高的保障，建议您采用 Failed over 策略 进行接入。

请求 A 记录

- **输入示例：**

```
curl "https://119.29.29.99/d?dn=cloud.tencent.com&token=yyyy"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。

返回结果中携带 ttl 信息

- **输入示例：**

```
curl "https://119.29.29.99/d?dn=cloud.tencent.com&token=yyyy&ttl=1"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6,120
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。记录值与 ttl 值以 ',' 分隔。

返回结果携带查询线路 IP 地址

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com&token=yyyy&clientip=1&ip=1.2.3.4&ttl=1"
```

- **解密后返回格式：**

```
12.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
```

- **格式说明：**返回结果中携带线路 IP 地址，以'|'分隔。如果没有传入 “ip=xxx” 参数，则返回出口 IP 地址；否则返回 ip 参数中的地址。

同时请求 A 和 AAAA 记录

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com&token=yyyy&clientip=1&ip=1.2.3.4&type=addrs&ttl=1
"
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6,120-
2402:4e00:0123:4567:0::2345;2403:4e00:0123:4567:0::2346,120|1.2.3.4
```

- **格式说明：**A 记录和 AAAA 记录之间以 '-' 分隔，A 记录在前，AAAA 记录在后。

返回结果中携带被查询域名

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com&token=yyyy&clientip=1&ip=1.2.3.4&query=1&ttl=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:2.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
```

- **格式说明：**返回格式为 “域名.:结果” 的格式。

批量域名请求

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com,www.qq.com,www.dnspod.cn&token=yyyy&clientip=1&ip
```

```
=1.2.3.4&ttl=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:2.3.3.4;2.3.3.5;2.3.3.6,120|1.2.3.4
www.qq.com.:3.3.3.4;3.3.3.5;3.3.3.6,180|1.2.3.4
www.dnspod.cn.:4.3.3.4;4.3.3.5;4.3.3.6,60|1.2.3.4
```

- **格式说明：**多个域名返回内容之间以 “换行符” 分隔，IP 地址附加在所有记录值的最后。

POST 方法请求 A 记录

- **输入示例：**

```
curl "https://119.29.29.99/d" -d "dn=cloud.tencent.com&token=yyyy" -X
POST
```

- **解密后返回格式：**

```
2.3.3.4;2.3.3.5;2.3.3.6
```

- **格式说明：**返回查询结果，多个结果以 ';' 分隔。

请求异常或无记录说明

查询 A 记录

- **输入示例：**

```
curl "https://119.29.29.99/d?dn=cloud.tencent.com&token=yyyy"
```

- **解密后返回格式：** 空字符串或者0。
- **格式说明：** 没有记录，则返回0。未在控制台添加主域名，返回 HTTP 状态码401和空字符。

返回结果中包含域名

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com&token=yyyy&query=1&ip=1.2.3.4&clientip=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:0|1.2.3.4
```

- **格式说明：**0表示没有记录。

返回 A 与 AAAA 的记录

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com&token=yyyy&type=addrs&query=1&ip=1.2.3.4&clientip
=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:0-0|1.2.3.4
```

- **格式说明：**0表示没有记录。如果某个记录存在，则该记录正常返回在结果中，例如 `cloud.tencent.com.:2.3.4.5;3.3.3.3-0|1.2.3.4`，表示 AAAA 记录无法查询到。

批量域名请求

- **输入示例：**

```
curl "https://119.29.29.99/d?
dn=cloud.tencent.com,www.qq.com,www.dnspod.cn&token=yyyy&clientip=1&ip
=1.2.3.4&ttl=1"
```

- **解密后返回格式：**

```
cloud.tencent.com.:0,0|1.2.3.4
www.qq.com.:3.3.3.4;3.3.3.5;3.3.3.6,180|1.2.3.4
www.dnspod.cn.:4.3.3.4;4.3.3.5;4.3.3.6,60|1.2.3.4
```

- **格式说明：**未查询到数据的域名则返回0。如果某个记录存在，则该记录正常返回在结果中。

HTTP 状态码

以下为接口业务逻辑相关的 HTTP 状态码。

状态码	描述
200 OK	如果接口调用正确，无论是否查询成功，均返回状态码200。
401 Unauthorized	请求的域名未在控制台添加过主域名。
404 Not Found	接口不存在或 URL 实际上访问了某不存在的资源。
429 Too Many Request	访问过于频繁，超过了服务器限制。
501 Not Implemented	使用了非 “GET” 或 “POST” 请求方式。

AES、DES 加密解密说明

最近更新时间：2025-05-22 14:38:42

操作场景

使用 DES、AES 加密算法可以对请求参数进行加密，并对其响应数据解密，防止明文请求在传输过程中被恶意篡改。本文将指导您如何使用 DES、AES 加密算法。

> **说明：**
> 使用 HTTPS 请求方式查询，传输的数据会因为 TLS 通道而被加密保护，因此不需要主动对传入的数据额外加密。

前提条件

- 已开通移动解析 HTTPDNS 并已获取授权 ID 和加密密钥及 HTTPS Token 等配置信息。详情可参见 配置信息说明 。
- 已在移动解析 HTTPDNS 控制台添加需要查询的域名。详情可参见 添加域名 。

操作流程

步骤1：确定加密方式。
目前移动解析 HTTPDNS，HTTP 请求查询方式支持 DES、AES、两种加密方式。

> **说明：**
> - 若您使用 HTTPS 请求查询方式，详情可参见 HTTPS 请求方式查询 。
> - 使用对应的密钥和算法将要解析的域名进行加密（如需使用 ip 参数，也需要将该参数值进行加密），并将加密后的结果与 ID（不需要加密）作为请求参数。

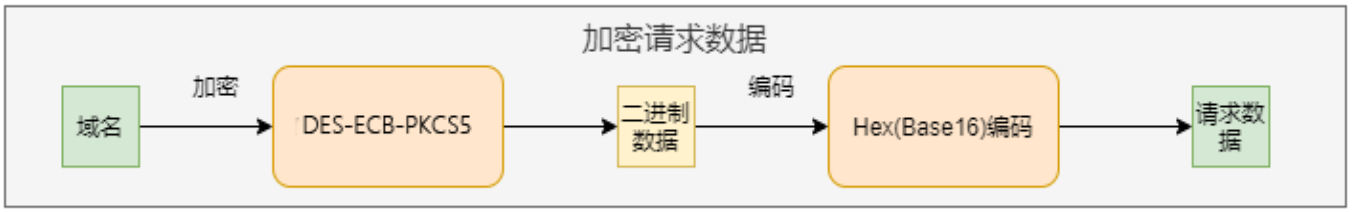
步骤2：发送加密的请求。
步骤3：接受加密的应答。
步骤4：将结果解密，即可获得所查询的域名对应的解析结果。

加密与解密算法使用说明

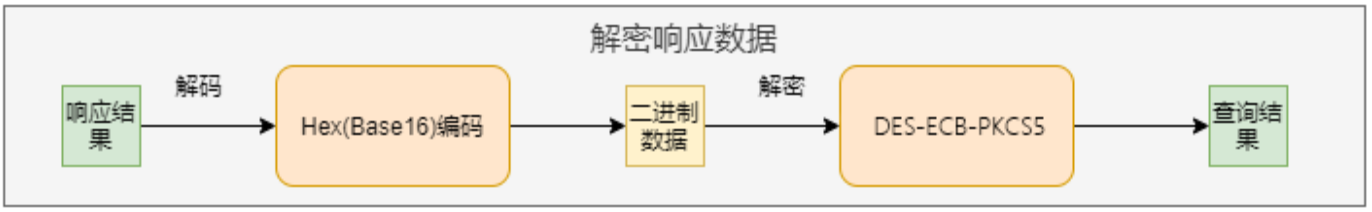
DES 算法

> **说明：**
> 使用 DES 进行加密与解密，密码长度为8个字符，分组加密模式为 `ECB` ，Padding 算法是 `PKCS5Padding` 。

加密后数据使用 `Hex（Base16）` 编码，将二进制数据转换为可见十六进制字符标识，`Hex` 编码后的数据长度将会加倍。具体流程如下图所示：



解密响应数据，先使用 `Hex` 解码为二进制数据，再使用 DES 算法解密为明文数据。具体流程如下图所示：



例如：您的域名为 `www.dnspod.cn` 、加密密钥为： `dnspodpa` 。流程如下：

1. 在 HTTPDNS 控制台 添加域名。
2. 使用 `DES-ECB-PKCS5` 加密算法与 DES 加密密钥 `dnspodpa` 加密域名将得到加密字符串 `87ae992c1321f299da3c0210a9900ae7` 。
3. 使用接口 `curl "http://119.29.29.98/d?dn=87ae992c1321f299da3c0210a9900ae7&id={授权ID}"` 请求 A 记录将得到数据长度加倍的加密字符串，如：
   ```
   fb37fabc98a2f71fca87f9cb259f74a03cc2254f80219f520a6fd36d0ad3eb4380be921f7a491fa4e
   4d84824286fec3df7d615b5e337b2ecf789bbb6db97e9df5533cefd5c41fb7503c0e8fa99cf0c20
   ```
4. 得到加倍的加密字符串后，使用 `DES-ECB-PKCS5` 加密算法与 DES 加密密钥 `dnspodpa` 进行解密后将得到明文的信息：`117.148.128.189;36.150.211.79;111.48.99.206;112.30.175.208;117.163.48.119` 。

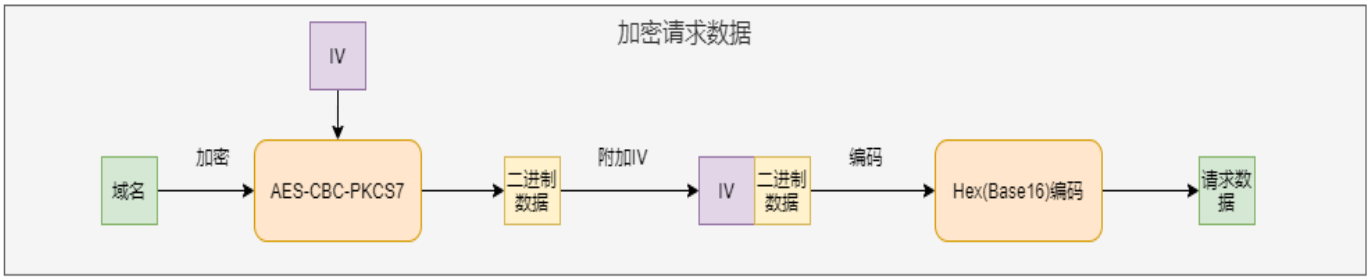
> **说明：**
> 以上字符串仅做示例使用，用于正常请求将无法使用。

AES 算法

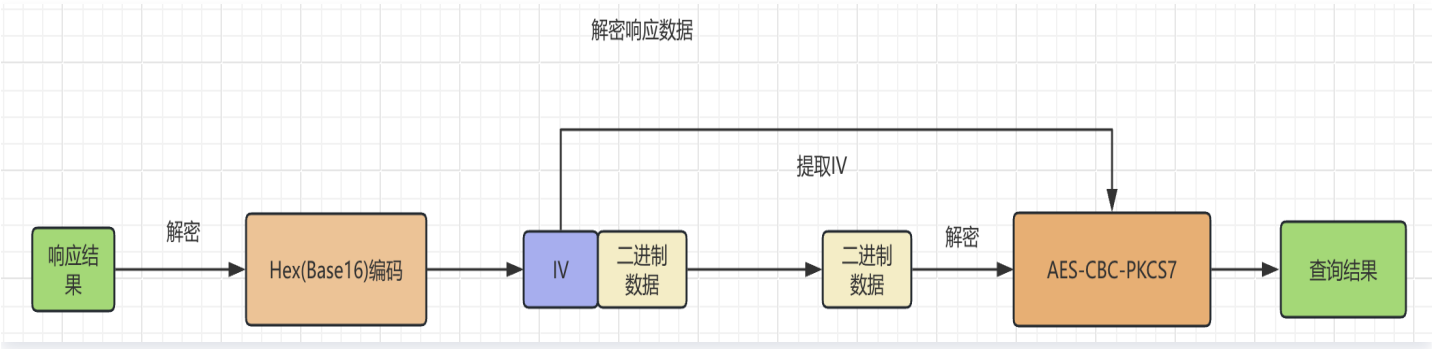
> **说明：**

> 使用 AES 进行加密与解密，密钥长度为16个字符，分组加密模式为 `CBC` ，Padding 算法是 `PKCS7Padding` 。

CBC 模式要求使用随机化 `IV` 作为初始加密与解密输入，因此该 `IV` 也会被带入到请求和响应中。加密后的数据，连同 `IV` 一起使用 `Hex` 编码，转换为可见十六进制标识。具体流程如下图所示：



解密时，使用 `Hex` 解码为二进制数据，前16字节为 `IV` 值， `IV` 后面为待使用 AES 算法解密的数据。使用 AES 算法解密后即为明文数据。具体流程如下图所示：



例如：您的域名为 `www.dnspod.cn` 、加密密钥为： `dnspodaespasswrd` 。流程如下：

1. 在 HTTPDNS 控制台 添加域名。
2. 使用 `AES-CBC-PKCS7` 加密算法与 AES 加密密钥 `dnspodaespasswrd` 、初始化的 `IV` 值： `68058d1967c6c8c9aa734dfc8a18ead9` 加密域名将得到加密字符串 。
3. 使用接口

```
curl "http://119.29.29.98/d?dn=68058d1967c6c8c9aa734dfc8a18ead97a23bf3e8f011f6509a37ed2b25a1192&id={授权ID}&alg=aes"
```

请求 A 记录将得到加密字符串，如：

```
f689db945693bde440729ec30fb3824c739838a72a30777b123dd1b8e0bb89de4c57d546e74f96ceab2664c3bb68485024339ad198675ac59e2cb5b60e726eb234609ac1c7c0925449f3f505f22104e48de7b2ee19aede451b09691c6826bea3
```

4. 提取前16字节的 IV 值得到 `f689db945693bde440729ec30fb3824c` ，剩下的加密字符串为

```
739838a72a30777b123dd1b8e0bb89de4c57d546e74f96ceab2664c3bb68485024339ad198675ac59e2cb5b60e726eb234609ac1c7c0925449f3f505f22104e48de7b2ee19aede451b09691c6826bea3
```

，使用 `AES-CBC-PKCS7` 加密算法与 AES 加密密钥 `dnspodaespasswrd` 、`IV` 值：`f689db945693bde440729ec30fb3824c` 进行解密后将得到明文的信息 `112.30.175.208;117.148.128.189;117.163.48.119;36.150.211.79;111.48.99.206` 。

HTTPDNS 加解密实践

最近更新时间：2025-05-22 14:38:42

操作场景

使用 DES、AES 加密算法可以对请求参数进行加密，并对其响应数据解密，防止明文请求在传输过程中被恶意篡改。本文将指导您如何使用 DES、AES 加密算法。

> **说明：**
> 使用 HTTPS 请求方式查询，传输的数据会因为 TLS 通道而被加密保护，因此不需要主动对传入的数据额外加密。

前提条件

掌握 Java、JavaScript、Go 或者 C++ 语言。

操作步骤

Java 版本

AES 加密

首先，创建 `org/example` 目录，并在 `org/example` 目录下，创建文件 `HTTPDNSAESHelper.java`，执行 `javac ./org/example/HTTPDNSAESHelper.java` 命令，得到 `HTTPDNSAESHelper$1.class`、`HTTPDNSAESHelper.class` 文件。

```
> ls ./org/example
HTTPDNSAESHelper$1.class HTTPDNSAESHelper.class   HTTPDNSAESHelper.java
```

执行 `java org.example.HTTPDNSAESHelper` 命令。
依次输入 HTTPDNS 的 id、查询的域名、16位的 AES 加解密的密钥。

```
> java org.example.HTTPDNSAESHelper
请输入你的id:

请输入你要查询的域名:
www.cloud.tencent.com
请输入你的key:

测试域名加密和解密是否正确:
加密后的域名: 7B746715B5614AFCB03CE3165664BDD0AFFE8B0390766278C45B4E477102790665D33E6DE1D41BDC0645EB8F8896F83E
解密后: www.cloud.tencent.com
响应结果: 891170ced0079572eabe12dd2ffedd4a7fc73a9406b317d75d2e464f98d99393828eb7cf7527c6df5b905a66f10817e49d82dcf3f34de2
6bc62ca00d75a58388
解密后: 183.60.155.197;14.116.174.244;113.105.162.121
```

`HTTPDNSAESHelper.java` **代码**

```java
package org.example;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;

public class HTTPDNSAESHelper {

    private static final int IVSize = 16;
    private static final int KeySize = 16;
    private static final String AlgorithmName = "AES";
    // 下面代码中有对应的pkcs7算法实现
    private static final String AlgorithmCombinationName =
"AES/CBC/NoPadding";
    private static final String Base16Table = "0123456789ABCDEF";
```

```
    private static final Map<Character, Byte> ReversedBase16Table = new
HashMap<Character, Byte>() {
        {
            put('0', (byte) 0);
            put('1', (byte) 1);
            put('2', (byte) 2);
            put('3', (byte) 3);
            put('4', (byte) 4);
            put('5', (byte) 5);
            put('6', (byte) 6);
            put('7', (byte) 7);
            put('8', (byte) 8);
            put('9', (byte) 9);
            put('A', (byte) 10);
            put('B', (byte) 11);
            put('C', (byte) 12);
            put('D', (byte) 13);
            put('E', (byte) 14);
            put('F', (byte) 15);
            put('a', (byte) 10);
            put('b', (byte) 11);
            put('c', (byte) 12);
            put('d', (byte) 13);
            put('e', (byte) 14);
            put('f', (byte) 15);
        }
    };

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("请输入你的id:");
        String id = scanner.nextLine();
        System.out.println("请输入你要查询的域名:");
        String data = scanner.nextLine();
        System.out.println("请输入你的key:");
        // 16个字节的key
        String key = scanner.nextLine();

        if (key.length() != 16) {
            throw new IllegalArgumentException("key的长度必须为16字节");
        }
```

```
        HTTPDNSAESHelper aesHelper = new HTTPDNSAESHelper();

        System.out.println("测试域名加密和解密是否正确: ");
        String encrypt = aesHelper.encrypt(data, key);
        System.out.println("加密后的域名: " + encrypt);

        String decrypt = aesHelper.decrypt(encrypt, key);
        System.out.println("解密后: " + decrypt);

        try {
            String line;
            StringBuilder response = new StringBuilder();
            URL url = new URL("http://119.29.29.98/d?dn=" + encrypt +
"&id=" + id + "&alg=aes");
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setDoOutput(true);
            conn.setRequestMethod("GET");

            int responseCode = conn.getResponseCode();
            if (responseCode != HttpURLConnection.HTTP_OK) {
                throw new RuntimeException("请求HTTP DNS接口失败, 响应码: "
+ responseCode);
            }

            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()))) {
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }

                System.out.println("响应结果: " + response);
                System.out.println("解密后: " +
aesHelper.decrypt(response.toString(), key));
            }
        } catch (MalformedURLException e) {
            throw new RuntimeException("请求HTTP DNS接口失败", e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
```

```
     * base16编码
     *
     * @param data 字节数组
     * @return base16编码后的字符串
     */
    public static String hexEncode(byte[] data) {
        if (data == null || data.length == 0) {
            return "";
        }

        StringBuilder result = new StringBuilder(data.length * 2);
        for (byte datum : data) {
            int high = (datum >> 4) & 0xf;
            int low = datum & 0xf;
            result.append(Base16Table.charAt(high));
            result.append(Base16Table.charAt(low));
        }

        return result.toString();
    }

    /**
     * base16解码
     *
     * @param data base16编码后的字符串
     * @return 字节数组
     */
    public static byte[] hexDecode(String data) {
        if (data == null || data.length() == 0) return null;

        if (data.length() % 2 != 0) {
            throw new RuntimeException("invalid hex encoded string");
        }

        byte[] result = new byte[data.length() / 2];
        for (int i = 0, j = 0; i < result.length; i++, j += 2) {
            byte high = ReversedBase16Table.get(data.charAt(j));
            byte low = ReversedBase16Table.get(data.charAt(j + 1));
            result[i] = (byte) ((high << 4) | (low & 0xf));
        }

        return result;
    }
```

```
    public static byte[] getUTF8Bytes(String str) {
        try {
            return str.getBytes("UTF-8");
        } catch (UnsupportedEncodingException e) {
            throw new RuntimeException("invalid data", e);
        }
    }

    public static boolean isBlank(String str) {
        return str == null || str.length() == 0;
    }

    /**
     * 加密
     *
     * @param data 数据
     * @param key  16个字节长度的key
     * @return 加密后的数据
     */
    public String encrypt(String data, String key) {
        if (key == null || key.length() != KeySize) {
            throw new IllegalArgumentException("key length must be 16");
        } else if (isBlank(data)) {
            throw new IllegalArgumentException("data is blank");
        }

        // 使用pkcs7Padding算法填充, 参考rfc5652
        byte[] rawData = pkcs7Padding(getUTF8Bytes(data), IVSize);
        byte[] ivBytes = generateRandomIVBytes();
        byte[] keyBytes = getUTF8Bytes(key);

        SecretKeySpec aesKey = new SecretKeySpec(keyBytes,
AlgorithmName);

        IvParameterSpec iv = new IvParameterSpec(ivBytes);

        try {
            // AES, CBC模式加密
            Cipher cipher =
Cipher.getInstance(AlgorithmCombinationName);
            cipher.init(Cipher.ENCRYPT_MODE, aesKey, iv);
            byte[] encryptedBytes = cipher.doFinal(rawData);
```

```
            // hex编码
            return hexEncode(concatBytes(ivBytes, encryptedBytes));
        } catch (Exception e) {
            throw new RuntimeException("failed to encrypt data", e);
        }
    }

    /**
     * 解密
     *
     * @param data 数据
     * @param key  16个字节长度的key
     * @return 解密后的数据
     */
    public String decrypt(String data, String key) {
        if (key == null || key.length() != KeySize) {
            throw new IllegalArgumentException("key length must be 16");
        } else if (isBlank(data)) {
            throw new IllegalArgumentException("data is blank");
        }

        // hex解码
        byte[] decodedBytes = hexDecode(data);

        if (decodedBytes == null || decodedBytes.length < IVSize) {
            throw new IllegalArgumentException("invalid encryptedData,
missing iv");
        }

        byte[] keyBytes = getUTF8Bytes(key);
        byte[] encryptedBytes = extractEncryptedData(decodedBytes);

        SecretKeySpec aesKey = new SecretKeySpec(keyBytes,
AlgorithmName);
        IvParameterSpec iv = extractIV(decodedBytes);

        try {
            // AES, CBC模式解密
            Cipher cipher =
Cipher.getInstance(AlgorithmCombinationName);
            cipher.init(Cipher.DECRYPT_MODE, aesKey, iv);
            byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
            // 去掉pkcs7Padding填充的字节
```

```
            return new String(unPkcs7Padding(decryptedBytes, IVSize));
        } catch (Exception e) {
            throw new RuntimeException("failed to decrypt data", e);
        }
    }

    private byte[] concatBytes(byte[] first, byte[] second) {
        if (first == null || first.length == 0) {
            return second;
        }

        if (second == null || second.length == 0) {
            return first;
        }

        byte[] target = new byte[first.length + second.length];
        System.arraycopy(first, 0, target, 0, first.length);
        System.arraycopy(second, 0, target, first.length,
second.length);
        return target;
    }

    /**
     * 从响应数据中提取iv参数
     *
     * @param data 响应数据
     * @return iv参数
     */
    private IvParameterSpec extractIV(byte[] data) {
        if (data.length < IVSize) {
            throw new IllegalArgumentException("missing iv part");
        }

        byte[] ivBytes = new byte[IVSize];
        System.arraycopy(data, 0, ivBytes, 0, ivBytes.length);
        return new IvParameterSpec(ivBytes);
    }

    /**
     * 从响应数据中提取加密数据
     *
     * @param data 响应数据
     * @return 加密数据
```

```
     */
    private byte[] extractEncryptedData(byte[] data) {
        if (data.length - IVSize <= 0) {
            throw new IllegalArgumentException("missing encrypted data
part");
        }

        byte[] encryptedBytes = new byte[data.length - IVSize];
        System.arraycopy(data, IVSize, encryptedBytes, 0,
encryptedBytes.length);
        return encryptedBytes;
    }

    /**
     * 生成随机的16字节的IV
     *
     * @return 随机的16字节的IV
     */
    private byte[] generateRandomIVBytes() {
        byte[] iv = new byte[16];
        Random random = new Random();
        random.nextBytes(iv);
        return iv;
    }

    private byte[] pkcs7Padding(byte[] data, int blockSize) {
        int paddingBytesAmount = blockSize - (data.length % blockSize);
        byte[] paddingResult = new byte[data.length +
paddingBytesAmount];
        System.arraycopy(data, 0, paddingResult, 0, data.length);
        for (int i = 0; i < paddingBytesAmount; i++) {
            paddingResult[data.length + i] = (byte) paddingBytesAmount;
        }
        return paddingResult;
    }

    private byte[] unPkcs7Padding(byte[] data, int blockSize) {
        if (data == null || data.length == 0 || data.length % blockSize
!= 0) {
            throw new RuntimeException("data isn't aligned to
blockSize");
        }
```

```
        int paddingBytesAmount = data[data.length - 1];
        for (int i = 0; i < paddingBytesAmount; i++) {
            if (data[data.length - i - 1] != paddingBytesAmount) {
                throw new RuntimeException("padding had malformed
entries");
            }
        }

        byte[] result = new byte[data.length - paddingBytesAmount];
        System.arraycopy(data, 0, result, 0, result.length);
        return result;
    }
}
```

DES 加密

首先，创建 `org/example` 目录，并在 `org/example` 目录下，创建文件 `HTTPDNSDESHelper.java`，执行 `javac ./org/example/HTTPDNSDESHelper.java` 命令，得到 `HTTPDNSDESHelper$1.class` 和 `HTTPDNSDESHelper.class` 文件。

```
> ls ./org/example
HTTPDNSDESHelper$1.class HTTPDNSDESHelper.class   HTTPDNSDESHelper.java
```

执行 `java org.example.HTTPDNSDESHelper` 命令。
依次输入 HTTPDNS 的 id、查询的域名、8位的 DES 加解密的密钥。

```
> java org.example.HTTPDNSDESHelper
请输入你的id:
[redacted]
请输入你要查询的域名:
www.cloud.tencent.com
请输入你的key:
[redacted]
测试域名加密和解密是否正确:
加密后的域名: 111A9C0C7AF3535CAAD4E9A51C7C3244F6771FED4D3B4837
解密后: www.cloud.tencent.com
响应结果: c17462db3a6f559552c2ae0f4051df03423975eb34c2fed0821b83e24e2bef5e046487e86225cbb3b79c80d1791f6810
解密后: 113.105.162.121;183.60.155.197;14.116.174.244
```

`HTTPDNSDESHelper.java` 代码

```
package org.example;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class HTTPDNSDESHelper {

    private static final int KeySize = 8;
    private static final String AlgorithmName = "DES";
    private static final String AlgorithmCombinationName =
"DES/ECB/PKCS5Padding";
    private static final String Base16Table = "0123456789ABCDEF";
    private static final Map<Character, Byte> ReversedBase16Table = new
HashMap<Character, Byte>() {
        {
            put('0', (byte) 0);
            put('1', (byte) 1);
            put('2', (byte) 2);
            put('3', (byte) 3);
            put('4', (byte) 4);
            put('5', (byte) 5);
            put('6', (byte) 6);
            put('7', (byte) 7);
            put('8', (byte) 8);
            put('9', (byte) 9);
            put('A', (byte) 10);
            put('B', (byte) 11);
            put('C', (byte) 12);
            put('D', (byte) 13);
            put('E', (byte) 14);
            put('F', (byte) 15);
            put('a', (byte) 10);
            put('b', (byte) 11);
```

```
            put('c', (byte) 12);
            put('d', (byte) 13);
            put('e', (byte) 14);
            put('f', (byte) 15);
        }
    };

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("请输入你的id:");
        String id = scanner.nextLine();
        System.out.println("请输入你要查询的域名:");
        String data = scanner.nextLine();
        System.out.println("请输入你的key:");
        // 8个字节的key
        String key = scanner.nextLine();

        if (key.length() != 8) {
            throw new IllegalArgumentException("key的长度必须为8字节");
        }

        HTTPDNSDESHelper desHelper = new HTTPDNSDESHelper();

        System.out.println("测试域名加密和解密是否正确: ");
        String encrypt = desHelper.encrypt(data, key);
        System.out.println("加密后的域名: " + encrypt);

        String decrypt = desHelper.decrypt(encrypt, key);
        System.out.println("解密后: " + decrypt);

        try {
            String line;
            StringBuilder response = new StringBuilder();
            URL url = new URL("http://119.29.29.98/d?dn=" + encrypt +
"&id=" + id + "&alg=des");
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setDoOutput(true);
            conn.setRequestMethod("GET");

            int responseCode = conn.getResponseCode();
            if (responseCode != HttpURLConnection.HTTP_OK) {
```

```
                throw new RuntimeException("请求HTTP DNS接口失败, 响应码: "
+ responseCode);
            }

            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()))) {
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }

                System.out.println("响应结果: " + response);
                System.out.println("解密后: " +
desHelper.decrypt(response.toString(), key));
            }
        } catch (MalformedURLException e) {
            throw new RuntimeException("请求HTTP DNS接口失败", e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * base16编码
     *
     * @param data 字节数组
     * @return base16编码后的字符串
     */
    public static String hexEncode(byte[] data) {
        if (data == null || data.length == 0) {
            return "";
        }

        StringBuilder result = new StringBuilder(data.length * 2);
        for (byte datum : data) {
            int high = (datum >> 4) & 0xf;
            int low = datum & 0xf;
            result.append(Base16Table.charAt(high));
            result.append(Base16Table.charAt(low));
        }

        return result.toString();
    }
```

```
    /**
     * base16解码
     *
     * @param data base16编码后的字符串
     * @return 字节数组
     */
    public static byte[] hexDecode(String data) {
        if (data == null || data.length() == 0) return null;

        if (data.length() % 2 != 0) {
            throw new RuntimeException("invalid hex encoded string");
        }

        byte[] result = new byte[data.length() / 2];
        for (int i = 0, j = 0; i < result.length; i++, j += 2) {
            byte high = ReversedBase16Table.get(data.charAt(j));
            byte low = ReversedBase16Table.get(data.charAt(j + 1));
            result[i] = (byte) ((high << 4) | (low & 0xf));
        }

        return result;
    }

    public static byte[] getUTF8Bytes(String str) {
        return str.getBytes(StandardCharsets.UTF_8);
    }

    public static boolean isBlank(String str) {
        return str == null || str.length() == 0;
    }

    public String encrypt(String data, String key) {
        if (key == null || key.length() != KeySize) {
            throw new IllegalArgumentException("key length must be 8");
        } else if (isBlank(data)) {
            throw new IllegalArgumentException("data is blank");
        }

        byte[] rawBytes = getUTF8Bytes(data);
        byte[] keyBytes = getUTF8Bytes(key);

        SecretKeySpec desKey = new SecretKeySpec(keyBytes,
AlgorithmName);
```

```
        try {
            Cipher cipher =
Cipher.getInstance(AlgorithmCombinationName);
            cipher.init(Cipher.ENCRYPT_MODE, desKey);
            byte[] encryptedData = cipher.doFinal(rawBytes);
            return hexEncode(encryptedData);
        } catch (Exception e) {
            throw new RuntimeException("failed to encrypt", e);
        }
    }

    public String decrypt(String data, String key) {
        if (key == null || key.length() != KeySize) {
            throw new IllegalArgumentException("key length must be 8");
        } else if (isBlank(data)) {
            throw new IllegalArgumentException("data is blank");
        }

        byte[] encryptedBytes = hexDecode(data);
        byte[] keyBytes = getUTF8Bytes(key);

        SecretKeySpec desKey = new SecretKeySpec(keyBytes,
AlgorithmName);

        try {
            Cipher cipher =
Cipher.getInstance(AlgorithmCombinationName);
            cipher.init(Cipher.DECRYPT_MODE, desKey);
            byte[] decryptedData = cipher.doFinal(encryptedBytes);
            return new String(decryptedData);
        } catch (Exception e) {
            throw new RuntimeException("failed to decrypt", e);
        }
    }
}
```

Go 版本

创建一个目录 `test` ，进入 `test` 目录，创建 `main.go` 文件，把以下代码复制到 `main.go` 文件中，然后运行 `go run main.go` 命令。
分别会进行 AES 加解密和 DES 加解密，在此过程中需要分别输入 HTTPDNS 的 id、请求域名、16位的 AES 加解密密钥、HTTPDNS 的 id、请求域名、8位的 DES 加解密密钥。

```
package main

import (
    "bytes"
    "crypto/aes"
    "crypto/cipher"
    "crypto/des"
    "encoding/hex"
    "errors"
    "fmt"
    "io"
    "math/rand"
    "net/http"
)

var (
    ErrInvalidBlockSize    = errors.New("invalid block size")
    ErrInvalidPKCSData     = errors.New("invalid PKCS data")
    ErrInvalidPKCSPadding = errors.New("invalid padding on input")
)

// AesDecrypt aes解密
// 步骤:
// 1. 先把加密文本base16(hex)解码
// 2. 从解码后的数据前16位提取出IV, 其他部分作为待解密数据
// 3. 根据IV和key解密数据
// 4. 把解密后的数据尝试去除填充数据(pkcs7)
func AesDecrypt(encryptedData, key string) (string, error) {
    cipherText, err := hex.DecodeString(encryptedData)
    if err != nil {
      return "", err
    }

    if len(cipherText) < aes.BlockSize {
      return "", errors.New("missing iv")
    }

    // 前16字节为IV
    iv := cipherText[:aes.BlockSize]
    cipherText = cipherText[aes.BlockSize:]

    c, err := aes.NewCipher([]byte(key))
    if err != nil {
```

```
        return "", err
    }

    mode := cipher.NewCBCDecrypter(c, iv)

    mode.CryptBlocks(cipherText, cipherText)

    unPaddingData, err := pkcs7UnPad(cipherText, aes.BlockSize)
    if err != nil {
        return "", err
    }

    return string(unPaddingData), nil
}

// AesEncrypt aes加密
// 步骤:
// 1. 生成16位的随机IV
// 2. 对加密数据尝试进行填充(pkcs7)
// 3. 使用key和IV加密填充后的数据
// 4. 拼接IV和加密后的数据
// 5. 把字节数组通过base16(hex)编码为字符串
func AesEncrypt(data, key string) (string, error) {
    iv := randomIV(aes.BlockSize)
    c, err := aes.NewCipher([]byte(key))
    if err != nil {
        return "", err
    }

    mode := cipher.NewCBCEncrypter(c, iv)

    cipherText, err := pkcs7([]byte(data), aes.BlockSize)
    if err != nil {
        return "", err
    }

    mode.CryptBlocks(cipherText, cipherText)

    // 前16字节为IV
    cipherText = append(iv, cipherText...)

    return hex.EncodeToString(cipherText), nil
}
```

```
func randomIV(blockSize int) []byte {
    iv := make([]byte, blockSize)
    for i := 0; i < blockSize; i++ {
      iv[i] = byte(rand.Int())
    }
    return iv
}

func DesEncrypt(data, key string) (string, error) {
    c, err := des.NewCipher([]byte(key))
    if err != nil {
      return "", err
    }

    cipherText, err := pkcs5([]byte(data), des.BlockSize)
    if err != nil {
      return "", err
    }

    dst := make([]byte, len(cipherText))
    for i := 0; i < len(cipherText); i += des.BlockSize {
      c.Encrypt(dst[i:i+des.BlockSize], cipherText[i:i+des.BlockSize])
    }

    return hex.EncodeToString(dst), nil
}

func DesDecrypt(encryptedData, key string) (string, error) {
    cipherText, err := hex.DecodeString(encryptedData)
    if err != nil {
      return "", err
    }

    if len(cipherText)%des.BlockSize != 0 {
      return "", errors.New("invalid encryptedData")
    }

    c, err := des.NewCipher([]byte(key))
    if err != nil {
      return "", err
    }
```

```
    dst := make([]byte, len(cipherText))
    for i := 0; i < len(cipherText); i += des.BlockSize {
      c.Decrypt(dst[i:i+des.BlockSize], cipherText[i:i+des.BlockSize])
    }

    unPaddingData, err := pkcs5UnPad(dst, des.BlockSize)
    if err != nil {
      return "", err
    }

    return string(unPaddingData), nil
}

func pkcs7(data []byte, blockSize int) ([]byte, error) {
    if blockSize <= 0 {
      return nil, ErrInvalidBlockSize
    }
    if len(data) == 0 {
      return nil, ErrInvalidPKCSData
    }
    n := blockSize - (len(data) % blockSize)
    pd := make([]byte, len(data)+n)
    copy(pd, data)
    copy(pd[len(data):], bytes.Repeat([]byte{byte(n)}, n))
    return pd, nil
}

func pkcs7UnPad(data []byte, blockSize int) ([]byte, error) {
    if blockSize <= 0 {
      return nil, ErrInvalidBlockSize
    }
    if len(data) == 0 {
      return nil, ErrInvalidPKCSData
    }
    if len(data)%blockSize != 0 {
      return nil, ErrInvalidPKCSPadding
    }
    c := data[len(data)-1]
    n := int(c)
    if n == 0 || n > len(data) {
      return nil, ErrInvalidPKCSPadding
    }
    for i := 0; i < n; i++ {
```

```
        if data[len(data)-i-1] != c {
            return nil, ErrInvalidPKCSPadding
        }
    }
    return data[:len(data)-n], nil
}

func pkcs5(data []byte, blockSize int) ([]byte, error) {
    if blockSize <= 0 {
        return nil, ErrInvalidBlockSize
    }
    if len(data) == 0 {
        return nil, ErrInvalidPKCSData
    }
    pd := blockSize - (len(data) % blockSize)
    return append(data, bytes.Repeat([]byte{byte(pd)}, pd)...), nil
}

func pkcs5UnPad(data []byte, blockSize int) ([]byte, error) {
    if blockSize <= 0 {
        return nil, ErrInvalidBlockSize
    }
    if len(data) == 0 {
        return nil, ErrInvalidPKCSData
    }
    if len(data)%blockSize != 0 {
        return nil, ErrInvalidPKCSPadding
    }

    c := data[len(data)-1]
    n := int(c)
    if n == 0 || n > len(data) {
        return nil, ErrInvalidPKCSPadding
    }
    for i := 0; i < n; i++ {
        if data[len(data)-i-1] != c {
            return nil, ErrInvalidPKCSPadding
        }
    }
    return data[:len(data)-n], nil
}

func executeHTTPDNSRequest(id, dn, alg string) (string, error) {
```

```
    resp, err := http.Get(fmt.Sprintf("http://119.29.29.98/d?
dn=%s&id=%s&alg=%s", dn, id, alg))
    if err != nil {
      return "", err
    }

    if resp.StatusCode != http.StatusOK {
      return "", errors.New(fmt.Sprintf("请求HTTP DNS接口失败，状态码：%d",
resp.StatusCode))
    }

    defer resp.Body.Close()
    data, err := io.ReadAll(resp.Body)
    if err != nil {
      return "", err
    }

    return string(data), nil
}

func scan(tip string, dest *string) {
    fmt.Println(tip)
    _, err := fmt.Scanln(dest)
    if err != nil {
      panic(err)
    }
}

func aesEncryptAndDecryptTest() {
    fmt.Println("AES加解密测试:")

    var id, key, domain string
    scan("请输入你的id:", &id)
    scan("请输入你要请求的域名:", &domain)
    scan("请输入你16字节的key:", &key)

    encryptedData, err := AesEncrypt(domain, key)
    if err != nil {
      panic(err)
    }
    fmt.Println("加密后的域名:", encryptedData)
    decryptedData, err := AesDecrypt(encryptedData, key)
    if err != nil {
```

```
        panic(err)
    }
    fmt.Println("解密后的域名:", decryptedData)

    data, err := executeHTTPDNSRequest(id, encryptedData, "aes")
    if err != nil {
        panic(err)
    }

    decryptedIPAddress, err := AesDecrypt(data, key)
    if err != nil {
        panic(err)
    }

    fmt.Println("解密后的IP地址:", decryptedIPAddress)
}

func desEncryptAndDecryptTest() {
    fmt.Println("DES加解密测试:")

    var id, key, domain string
    scan("请输入你的id:", &id)
    scan("请输入你要请求的域名:", &domain)
    scan("请输入你8字节的key:", &key)

    encryptedData, err := DesEncrypt(domain, key)
    if err != nil {
        panic(err)
    }
    fmt.Println("加密后的域名:", encryptedData)
    decryptedData, err := DesDecrypt(encryptedData, key)
    if err != nil {
        panic(err)
    }
    fmt.Println("解密后的域名:", decryptedData)

    data, err := executeHTTPDNSRequest(id, encryptedData, "des")
    if err != nil {
        panic(err)
    }

    decryptedIPAddress, err := DesDecrypt(data, key)
    if err != nil {
```

```
        panic(err)
    }

    fmt.Println("解密后的IP地址:", decryptedIPAddress)
}

func main() {
    aesEncryptAndDecryptTest()
    desEncryptAndDecryptTest()
}
```

C++ 版本

安装 make、g++、以及 libcurl 和 libcrypto 库。
在 Centos 系统上执行以下命令：

```
yum install -y libcurl-devel openssl openssl-dev make gcc gcc-c++
```

创建 `test` 目录，进入 `test` 目录，复制以下文件到 `test` 目录，执行 `make` ，然后执行 `main` 可执行文件，根据提示输入获取响应 HTTPDNS 响应结果。
依次输入授权 ID、需要通过 AES 加解密解析的域名、AES 密钥、授权 ID、需要通过 DES 加解密解析的域名、DES 密钥。

```
$ ./main
AES加解密
请输入你的id:
[redacted]
请输入需要解析的域名：
www.qq.com
请输入16个字节的key:
[redacted]
响应结果：183.194.238.19;183.194.238.117
DES加解密
请输入你的id:
[redacted]
请输入需要解析的域名：
www.qq.com
请输入8个字节的key:
[redacted]
响应结果：183.194.238.117;183.194.238.19
```

Makefile

```
all:
```

```
    g++ -Wall -Werror -g -o main main.cpp des.cpp aes.cpp hex.cpp -
lcrypto -lcurl
clean:
    - rm -rf ./main
```

main.cpp

```
#include "aes.h"
#include "des.h"
#include <iostream>
#include <curl/curl.h>

size_t http_dns_response_callback(char *ptr, size_t size, size_t nmemb,
void *userdata)
{
    size_t real_size = size * nmemb;
    std::string *response = (std::string *)userdata;

    response->append(ptr, real_size);
    return real_size;
}

void send_http_dns_request(const std::string &encrypted_domain, const
std::string &id,
                           const std::string &alg, std::string
&response)
{
    std::string url;
    CURL *easy_handle;
    CURLcode res;

    easy_handle = curl_easy_init();

    if (!easy_handle)
        throw std::runtime_error("curl_easy_init() failed");

    url.append("http://119.29.29.98/d?dn=")
        .append(encrypted_domain)
        .append("&id=")
        .append(id)
        .append("&alg=")
        .append(alg);
```

```
    res = curl_easy_setopt(easy_handle, CURLOPT_URL, url.c_str());
    if (res != CURLE_OK)
    {
        curl_easy_cleanup(easy_handle);
        throw std::runtime_error("failed to set url option");
    }

    res = curl_easy_setopt(easy_handle, CURLOPT_WRITEFUNCTION,
http_dns_response_callback);
    if (res != CURLE_OK)
    {
        curl_easy_cleanup(easy_handle);
        throw std::runtime_error("failed to set writedata option");
    }

    res = curl_easy_setopt(easy_handle, CURLOPT_WRITEDATA, &response);
    if (res != CURLE_OK)
    {
        curl_easy_cleanup(easy_handle);
        throw std::runtime_error("failed to set writedata option");
    }

    res = curl_easy_perform(easy_handle);
    if (res != CURLE_OK)
    {
        curl_easy_cleanup(easy_handle);
        throw std::runtime_error(std::string("curl_easy_perform()
failed: ").append(curl_easy_strerror(res)));
    }

    curl_global_cleanup();
}

void get_user_input(const std::string &tip, std::string &id, std::string
&domain,
                    std::string &key, const size_t key_length)
{
    std::cout << tip << std::endl;
    std::cout << "请输入你的id:" << std::endl;
    std::cin >> id;
    std::cout << "请输入需要解析的域名:" << std::endl;
    std::cin >> domain;
    std::cout << "请输入" << key_length << "个字节的key:" << std::endl;
```

```
    std::cin >> key;

    if (id.size() == 0)
        throw std::runtime_error("请输入有效的id");

    if (domain.size() == 0)
        throw std::runtime_error("请输入有效的域名");

    if (key.size() != key_length)
        throw std::runtime_error(std::string("key长度必须
为").append(std::to_string(key_length)).append("个字节"));
}

void aes_test()
{
    std::string id, domain, key, response;

    get_user_input("AES加解密", id, domain, key, 16);

    std::string &&encrypted_data = aes_encrypt(domain, key);

    send_http_dns_request(encrypted_data, id, "aes", response);

    std::string &&decrypted_data = aes_decrypt(response, key);
    std::cout << "响应结果: " << decrypted_data << std::endl;
}

void des_test()
{
    std::string id, domain, key, response;

    get_user_input("DES加解密", id, domain, key, 8);

    std::string &&encrypted_data = des_encrypt(domain, key);

    send_http_dns_request(encrypted_data, id, "des", response);

    std::string &&decrypted_data = des_decrypt(response, key);
    std::cout << "响应结果: " << decrypted_data << std::endl;
}

int main()
{
```

```
    aes_test();
    des_test();
}
```

aes.cpp

```
#include "aes.h"
#include "hex.h"
#include <openssl/evp.h>
#include <openssl/aes.h>
#include <time.h>
#include <stdlib.h>
#include <stdexcept>

typedef unsigned char uchar;

const size_t IV_SIZE = 16;
const size_t KEY_SIZE = 16;
const size_t BLOCK_SIZE = 16;

static std::string generate_iv()
{
    size_t i;
    std::string iv;

    srand(time(NULL));

    for (i = 0; i < IV_SIZE; i++)
        iv.push_back(rand() % 256);

    return iv;
}

std::string aes_encrypt(const std::string &data, const std::string &key)
{
    if (data.size() == 0)
        throw std::runtime_error("data is empty");
    else if (key.size() != KEY_SIZE)
        throw std::runtime_error("key length must be 16");

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    std::string &&iv = generate_iv();
```

```
    if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), nullptr, (uchar
*)key.c_str(), (uchar *)iv.c_str()))
    {
        EVP_CIPHER_CTX_free(ctx);
        throw std::runtime_error("EVP_EncryptInit_ex() failed");
    }

    EVP_CIPHER_CTX_set_key_length(ctx, KEY_SIZE);

    uchar *output = new uchar[data.size() + 2 * BLOCK_SIZE];
    int output_len = 0;

    if (1 != EVP_EncryptUpdate(ctx, output, &output_len, (uchar
*)data.c_str(), data.size()))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_EncryptUpdate() failed");
    }

    uchar *pad_buf = output + output_len;
    int pad_len;

    if (1 != EVP_EncryptFinal_ex(ctx, pad_buf, &pad_len))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_EncryptFinal_ex() failed");
    }

    std::string result;

    result.append(iv);
    result.append((char *)output, output_len + pad_len);

    EVP_CIPHER_CTX_free(ctx);
    delete[] output;

    return encode(result);
}

std::string aes_decrypt(const std::string &data, const std::string &key)
{
```

```
    if (data.size() == 0)
        throw std::runtime_error("data is empty");
    else if (key.size() != KEY_SIZE)
        throw std::runtime_error("key length must be 16");

    std::string &&decoded_data = decode(data);

    if (decoded_data.size() <= IV_SIZE)
        throw std::runtime_error("invalid data");

    std::string &&iv = decoded_data.substr(0, IV_SIZE);
    std::string &&encrypted_data = decoded_data.substr(IV_SIZE);

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();

    if (1 != EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, (uchar
*)key.c_str(), (uchar *)iv.c_str()))
    {
        EVP_CIPHER_CTX_free(ctx);
        throw std::runtime_error("EVP_DecryptInit_ex() failed");
    }

    EVP_CIPHER_CTX_set_key_length(ctx, KEY_SIZE);

    uchar *output = new uchar[data.size() + 2 * BLOCK_SIZE];
    int output_len = 0;

    if (1 != EVP_DecryptUpdate(ctx, output, &output_len, (uchar
*)encrypted_data.c_str(), encrypted_data.size()))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_DecryptUpdate() failed");
    }

    int pad_len = 0;
    if (1 != EVP_DecryptFinal_ex(ctx, output + output_len, &pad_len))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_DecryptFinal_ex() failed");
    }
```

```
    output_len += pad_len;
    output[output_len] = 0;

    std::string result = (char *)output;

    EVP_CIPHER_CTX_free(ctx);
    delete[] output;

    return result;
}
```

aes.h

```
#ifndef AES_H
#define AES_H

#include <string>

std::string aes_encrypt(const std::string &data, const std::string
&key);
std::string aes_decrypt(const std::string &data, const std::string
&key);

#endif
```

des.cpp

```
#include "des.h"
#include "hex.h"
#include <openssl/evp.h>
#include <openssl/des.h>
#include <stdexcept>

typedef unsigned char uchar;

const size_t BLOCK_SIZE = 16;
const size_t KEY_SIZE = 8;

std::string des_encrypt(const std::string &data, const std::string &key)
{
    if (data.size() == 0)
        throw std::runtime_error("data is empty");
```

```
    else if (key.size() != KEY_SIZE)
        throw std::runtime_error("key length must be 8");

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();

    if (1 != EVP_EncryptInit_ex(ctx, EVP_des_ecb(), NULL, (uchar
*)key.c_str(), NULL))
    {
        EVP_CIPHER_CTX_free(ctx);
        throw std::runtime_error("EVP_EncryptInit_ex() failed");
    }

    EVP_CIPHER_CTX_set_key_length(ctx, KEY_SIZE);

    uchar *output = new uchar[data.size() + BLOCK_SIZE * 2];
    int output_len = 0;

    if (1 != EVP_EncryptUpdate(ctx, output, &output_len, (uchar
*)data.c_str(), data.size()))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_EncryptUpdate() failed");
    }

    int pad_len = 0;
    if (1 != EVP_EncryptFinal_ex(ctx, output + output_len, &pad_len))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_EncryptFinal_ex() failed");
    }

    output[output_len + pad_len] = 0;
    std::string result = (char *)output;

    EVP_CIPHER_CTX_free(ctx);
    delete[] output;

    return encode(result);
}

std::string des_decrypt(const std::string &data, const std::string &key)
```

```
{
    if (data.size() == 0)
        throw std::runtime_error("data is empty");
    else if (key.size() != KEY_SIZE)
        throw std::runtime_error("key length must be 8");

    std::string &&decoded_data = decode(data);

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();

    if (1 != EVP_DecryptInit_ex(ctx, EVP_des_ecb(), NULL, (uchar
*)key.c_str(), NULL))
    {
        EVP_CIPHER_CTX_free(ctx);
        throw std::runtime_error("EVP_DecryptInit_ex() failed");
    }

    EVP_CIPHER_CTX_set_key_length(ctx, KEY_SIZE);

    uchar *output = new uchar[data.size() + 2 * BLOCK_SIZE];
    int output_len = 0;

    if (1 != EVP_DecryptUpdate(ctx, output, &output_len, (uchar
*)decoded_data.c_str(), decoded_data.size()))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_DecryptUpdate() failed");
    }

    int pad_len = 0;
    if (1 != EVP_DecryptFinal_ex(ctx, output + output_len, &pad_len))
    {
        EVP_CIPHER_CTX_free(ctx);
        delete[] output;
        throw std::runtime_error("EVP_DecryptFinal_ex() failed");
    }

    output_len += pad_len;
    output[output_len] = 0;

    std::string result = (char *)output;
```

```
    EVP_CIPHER_CTX_free(ctx);
    delete[] output;

    return result;
}
```

des.h

```
#ifndef DES_H
#define DES_H

#include <string>

std::string des_encrypt(const std::string &data, const std::string
&key);
std::string des_decrypt(const std::string &data, const std::string
&key);

#endif
```

hex.h

```
#ifndef HEX_H
#define HEX_H

#include <string>

std::string encode(const std::string &data);
std::string decode(const std::string &data);

#endif
```

hex.cpp

```
#include "hex.h"
#include <stdexcept>
#include <iostream>
#include <map>

static const char HEX_STRING[] = "0123456789ABCDEF";
static std::map<char, int> HEX_TABLE = {
    {'0', 0},
```

```
    {'1', 1},
    {'2', 2},
    {'3', 3},
    {'4', 4},
    {'5', 5},
    {'6', 6},
    {'7', 7},
    {'8', 8},
    {'9', 9},
    {'a', 10},
    {'b', 11},
    {'c', 12},
    {'d', 13},
    {'e', 14},
    {'f', 15},
    {'A', 10},
    {'B', 11},
    {'C', 12},
    {'D', 13},
    {'E', 14},
    {'F', 15},
};

std::string encode(const std::string &data)
{
    size_t i;
    int high, low;
    std::string res;

    for (i = 0; i < data.size(); i++)
    {
        high = (data[i] >> 4) & 0xf;
        low = data[i] & 0xf;
        res.push_back(HEX_STRING[high]);
        res.push_back(HEX_STRING[low]);
    }

    return res;
}

std::string decode(const std::string &data)
{
    size_t i;
```

```
    std::string res;

    if (data.size() == 0)
        return res;
    else if (data.size() % 2 != 0)
        throw std::runtime_error("invalid data length");

    for (i = 0; i < data.size(); i += 2)
    {
        if (HEX_TABLE.find(data[i]) == HEX_TABLE.end() ||
HEX_TABLE.find(data[i + 1]) == HEX_TABLE.end())
            throw std::runtime_error("invalid character");
        res.push_back((HEX_TABLE[data[i]] << 4) | HEX_TABLE[data[i +
1]]);
    }

    return res;
}
```

JavaScript版本

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DES和AES加解密实践</title>
</head>
<body>
    <h1>DES和AES加解密实践</h1>
</body>
<!-- 注意：需要引入crypto-js库 -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-
js/4.2.0/crypto-js.min.js"></script>
<script>
    class AESHelper {
        constructor(key) {
            if (typeof (key) !== "string" || key.length < 16) {
                throw "[AES] failed to construct AESHelper, invalid
key";
            }
            this.key = CryptoJS.enc.Utf8.parse(key);
        }
```

```
        Encrypt(data) {
            if (typeof (data) !== "string" || data.length === 0) {
                throw "[AES] failed to encrypt, invalid data";
            }
            const iv = CryptoJS.lib.WordArray.random(16);
            const encryptedContent = CryptoJS.AES.encrypt(data,
this.key, {
                iv: iv,
                mode: CryptoJS.mode.CBC,
                padding: CryptoJS.pad.Pkcs7
            });
            return iv.toString(CryptoJS.enc.Hex) +
encryptedContent.ciphertext.toString(CryptoJS.enc.Hex);
        }

        Decrypt(data) {
            if (typeof (data) !== "string" || data.length <= 32) {
                throw "[AES] failed to decrypt, invalid data";
            }

            // 前32个字符是16个字节的iv值通过hex编码的
            const iv = CryptoJS.enc.Hex.parse(data.substr(0, 32));
            data = data.substr(32, data.length - 32);

            const decryptionParam = CryptoJS.lib.CipherParams.create({
                ciphertext: CryptoJS.enc.Hex.parse(data)
            });

            const decryptedContent =
CryptoJS.AES.decrypt(decryptionParam, this.key, {
                iv: iv,
                mode: CryptoJS.mode.CBC,
                padding: CryptoJS.pad.Pkcs7
            });
            return decryptedContent.toString(CryptoJS.enc.Utf8);
        }
    }

    class DESHelper {
        constructor(key) {
            if (typeof (key) !== "string" || key.length < 8) {
                throw "[DES] failed to construct DESHelper, invalid
key";
```

```
            }
            this.key = CryptoJS.enc.Utf8.parse(key);
        }

        Encrypt(data) {
            if (typeof (data) !== "string" || data.length === 0) {
                throw "[DES] failed to encrypt, invalid data";
            }
            const encryptedContent = CryptoJS.DES.encrypt(data,
this.key, {
                mode: CryptoJS.mode.ECB,
                padding: CryptoJS.pad.Pkcs7
            });
            return
encryptedContent.ciphertext.toString(CryptoJS.enc.Hex);
        }

        Decrypt(data) {
            if (typeof (data) !== "string" || data.length == 0) {
                throw "[DES] failed to decrypt, invalid data";
            }

            const decryptionParam = CryptoJS.lib.CipherParams.create({
                ciphertext: CryptoJS.enc.Hex.parse(data)
            });

            const decryptedContent =
CryptoJS.DES.decrypt(decryptionParam, this.key, {
                mode: CryptoJS.mode.ECB,
                padding: CryptoJS.pad.Pkcs7
            });
            return decryptedContent.toString(CryptoJS.enc.Utf8);
        }
    }

    {
        // 输入对应的aes key和域名
        const aesHelper = new AESHelper("your aes key");
        const encryptedData = aesHelper.Encrypt("your domain");
        console.log('AES加密后的数据', encryptedData);
        const decryptedData = aesHelper.Decrypt(encryptedData);
        console.log('AES解密后的数据', decryptedData);
    }
```

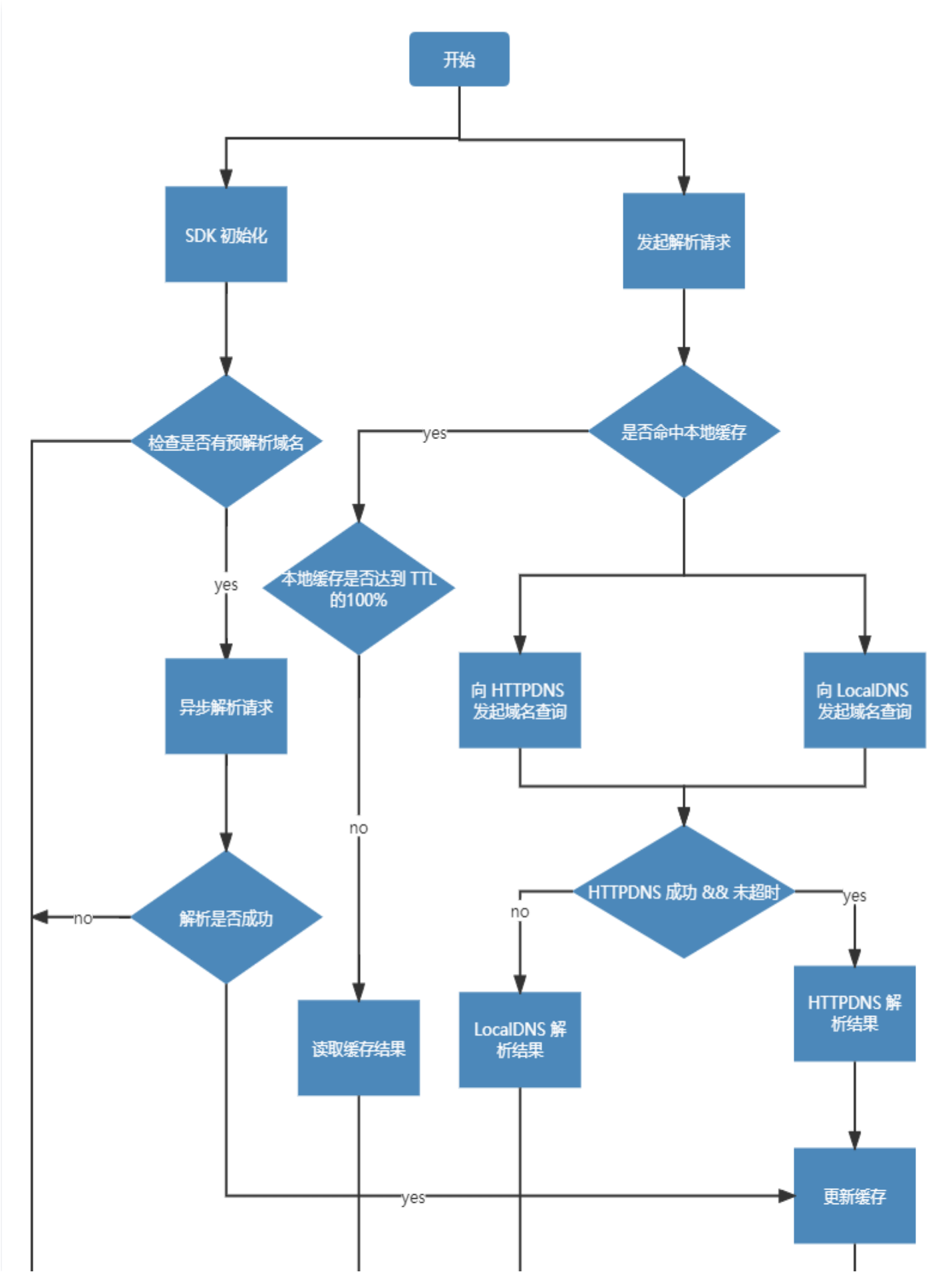
```
    {
        // 输入对应的des key和域名
        const desHelper = new DESHelper("your des key");
        const encryptedData = desHelper.Encrypt("your domain");
        console.log('DES加密后的数据', encryptedData);
        const decryptedData = desHelper.Decrypt(encryptedData);
        console.log('DES解密后的数据', decryptedData);
    }
</script>
</html>
```

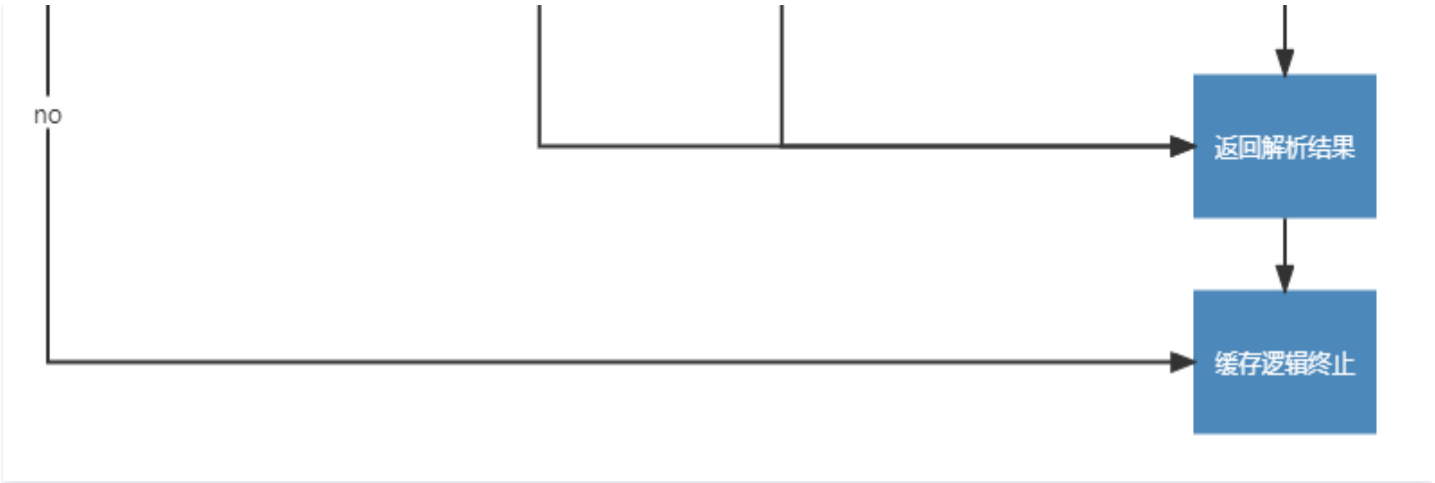
API 接入实践教程

最近更新时间：2023-07-25 17:05:12

客户端接入流程

接入 HTTPDNS 过程中，需要改造移动客户端的域名解析机制，流程参考如下：


开始
SDK 初始化
发起解析请求
检查是否有预解析域名
是否命中本地缓存
yes
yes
本地缓存是否达到 TTL 的100%
异步解析请求
向 HTTPDNS 发起域名查询
向 LocalDNS 发起域名查询
no
HTTPDNS 成功 && 未超时
no
yes
解析是否成功
no
读取缓存结果
LocalDNS 解析结果
HTTPDNS 解析结果
yes
更新缓存



设计策略

改造过程中需要遵循以下两个设计策略：

Failed over 策略

虽然 HTTPDNS 已经接入 BGP Anycast，并实现了多地跨机房容灾，但为了保证在最坏的情况下客户端域名解析依然不受影响，建议您采用以下的 Failed over 策略：

1. 先向 HTTPDNS 发起域名查询请求。
2. 如果 HTTPDNS 查询返回的结果不是一个 IP 地址（结果为空、结果非 IP、连接超时等），则通过本地 LocalDNS 进行域名解析。超时时间建议为5s。

缓存策略

移动互联网用户的网络环境比较复杂，为了尽可能地减少由于域名解析导致的延迟，建议在本地进行缓存。缓存规则如下：

- **缓存时间**：缓存时间建议设置为120s至600s，不可低于60s。
- **缓存更新**：
  缓存更新应在以下两种情形下进行：
  - **用户网络状态发生变化时：**
    移动互联网用户的网络状态由3G切换 Wi-Fi，Wi-Fi 切换3G的情况下，其接入点的网络归属可能发生变化，用户的网络状态发生变化时，需要重新向 HTTPDNS 发起域名解析请求，以获得用户当前网络归属下的最优指向。
  - **缓存过期时：**
    当域名解析的结果缓存时间到期时，客户端应该向 HTTPDNS 重新发起域名解析请求以获取最新的域名对应的 IP。为了减少用户在缓存过期后重新进行域名解析时的等待时间，建议在 75%TTL 时就开始进行域名解析。例如，本地缓存的 TTL 为600s，那么在第600 * 0.75 = 450s 时，客户端就应该进行域名解析。

除了以上几点建议外，减少域名解析的次数也能有效的减少网络交互，提升用户访问体验。建议在业务允许的情况下，尽量减少域名的数量。如需区分不同的资源，建议通过 url 来进行区分。

其他注意事项

为了让您更好的改造移动客户端，请改造前阅读以下注意事项：

- 请尽量将不同功能用同样域名，资源区分通过 url 来实现，减少域名解析次数（用户体验好，容灾切换方便。多一个域名，即使域名已命中缓存，至少多100ms的访问延迟）。
- 设置的缓存 TTL 值不可太低（不可低于60s），防止频繁进行 HTTPDNS 请求。
- 接入移动解析 HTTPDNS 的业务需要保留用户本地 LocalDNS 作为容灾通道，当 HTTPDNS 无法正常服务时（移动网络不稳定或 HTTPDNS 服务出现问题），可以使用 LocalDNS 进行解析。
- Android 程序中可能出现404错误，但浏览器中正常，可能为权限问题或者其他问题。详情请参考 Android 请求返回 404 。
- bytetohex&hextobyte，需自己实现接口，进行16进制字符串与字节的转换。
- HTTPS 问题，需在客户端 hook 客户端检查证书的 domain 域和扩展域看是否包含本次请求的 host 的过程，将 IP 直接替换成原来的域名，再执行证书验证。或者忽略证书认证，类似于 curl -k 参数。
- HTTPDNS 请求建议超时时间2 - 5s左右。
- 在网络类型变化时，例如，4G切换到 Wi-Fi，不同 Wi-Fi 间切换等，需要重新执行 HTTPDNS 请求刷新本地缓存。

HTTPDNS 服务 IP 容灾方案说明

最近更新时间：2025-03-24 17:24:42

操作场景

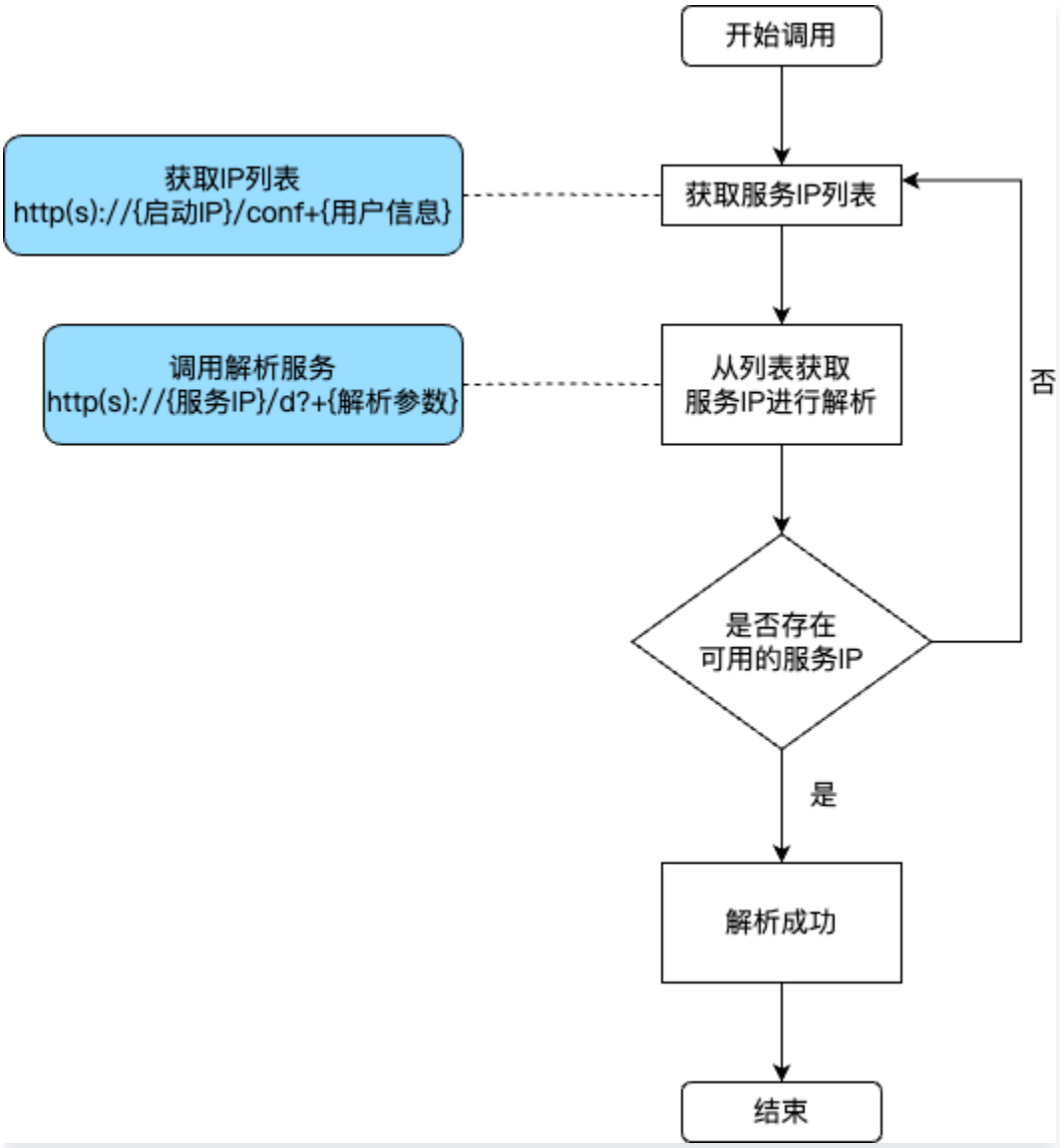
该实践方案适用于使用 API 接入 HTTPDNS 的用户。使用 SDK 的用户，不需要参照该方案进行修改，升级 SDK 到最新版本即可（ iOS 端 SDK 升级到v1.11.0，Android 端 SDK 升级到 v4.10.0 ），SDK 新版本已具备自动更换服务 IP 的能力。

术语介绍

- 服务 IP：用于提供 HTTPDNS 解析服务的 IP，例如119.29.29.98。
- 启动 IP：用于调用 API，动态获取服务 IP 列表。

> **说明：**
> 启动 IP 请联系售后或 在线支持 获取。

接入流程



1. 通过启动 IP，调用接口获取服务 IP 列表 。
2. 从 IP 列表中轮询/随机选择服务 IP，发起解析请求。
3. 如果 IP 列表中，全部服务 IP 均调用失败，重新获取服务 IP 列表。
4. 定时调用启动 IP 更新服务 IP 列表（建议每12小时），保持服务 IP 列表的有效性，或者在特殊场景下进行调用，例如：
   - 客户端冷启动时进行更新
   - 切换网络环境时
5. 调用“获取 IP 列表接口”的次数和时间，需要做好控制，避免调用失败时，阻塞 HTTPDNS 解析流程（建议重试1次，超时时间设置为2秒）；也可以在实际开发过程中，考虑采用异步的方式，刷新服务 IP 列表。

获取 IP 列表接口说明

接口描述

接口请求地址：http(s)://{启动IP}/conf? + {请求参数}。

请求方式：POST 或 GET。

HTTP 请求参数

参数名	参数含义	是否必选	取值	加密	说明
id	用户标识（即授权 ID）	是	1 – 100000	否	如果使用 AES、DES 加密方式，必须传入授权 ID，不需要进行加密。
alg	选择使用何种算法	否	[aes/des]	否	默认使用 DES 算法，不同算法具有不同密钥。

HTTP 请求/返回示例

输入示例：

```
curl "http://{启动IP}/conf?id=xxxx"
```

解密后返回格式：

```
log:0|domain:0|ip:119.29.29.98;119.28.28.98|ttl:60
```

参数“ip”后面的结果为服务 IP，不同服务 IP 采用分号分隔，其余参数可以忽略。

HTTPS 请求参数：

参数名	参数含义	是否必选	取值	加密	说明
token	使用 HTTPS 方式的标识	是	整型数据	否	token 获取详情请参见 配置信息说明

HTTPS 请求/返回示例：

输入示例：

```
curl -S -s -k "https://{启动IP}/conf?token=xxxx"
```

返回格式：

```
log:0|domain:0|ip:119.29.29.99;119.28.28.99|ttl:60
```

参数“ip”后面的结果为服务 IP，不同服务 IP 采用分号分隔，其余参数可以忽略。

> **说明：**
> 如果使用 HTTP 请求获取的服务 IP，仅限于 HTTP 解析，如果使用 HTTPS 请求获取的服务 IP，仅限于 HTTPS 解析。