CAID API 接入文档

Status: v1.15

1. 概述

CAID 作为 iOS 系统上广告行业的统一设备标识 ID，目前对外提供 API 的接入方式，这里描述 API 的接口协议。

2. 接入准备

对于需要接入 CAID API 的接入方，需要向 CAID 广告标识服务器申请接入，接入成功后会分配得到两个 ID，分别为 dev_id 和公钥 pub_key。接入时需要传入对应的 dev_id，并使用公钥对上报的信息进行 RSA 加密上报密文，收到 API 返回的响应后进行 RSA 解密，从而获得最终结果。
建议都将参数上传到自己的服务端，从服务端接入 API，客户端接入不可控，容易更新不及时。服务端调用加解密使用分配的 public_for_api.pem 文件中 key 值，客户端调用加解密使用 pub_for_sdk 证书。
由于带宽资源有限，接入方同一台设备（相同的 idfa）平均每天请求量不可超过 5 次，可通过缓存 CAID 方式优化，缓存时间不超过 24 小时，若不符合以上规范，有可能会导致 CAID 返回结果丢包、延迟等情况。若极端过量请求导致服务器宕机等问题，CAID 广告标识服务器将下线请求密钥，不再接受申请 CAID 服务。

3. 协议

3.1 CAID 获取接口

3.1.1 请求

```
curl 'https://caid.china-caa.org/v1.0/get' \
  -H 'Content-Type: application/json' \
  -H 'cache-control: no-cache' \
  -d '{
      "dev_id":"1", // 申请接入后分配的开发 ID，注意为字符串
      "encrypted_device_info":"OTjhCawxt..." // 设备公共信息按照附录组装成字典并序列化，
      使用分配的公钥加密的密文，组装方式及一个示例加密串可见附录
      }'
```

3.1.2 响应

```
{
  "code": 0,
  "message": "ok",
  "data":"fUvClL9b18AaRvxXKR8GFSTm..." // CAID 加密后的密文，使用分配的公钥进行RSA 解密，
  解密代码见附录
}
```

解密 data 之后返回格式如下，返回信息使用及版本迭代详见第 4 章节

```
[
       // CAID 算法升级期间会同时下发新老版本，用版本号来区分
       {
            "version":"20211207",
            "caid":"f1e86165d32b63ad1ad220f95f91b846”
       },
       {
            "version":"20220111",
            "caid":"2aec028e50c20e955c39d73ce422a195”
       }
  ]
```

3.2 测试

为获得最完整的结果响应，调用方可以直接请求线上环境来进行测试验证，示例结果详见附录。

4. 应用与规划

4.1 版本迭代

可以看到解密接口返回的 data 字段，我们可以获得 CAID 明文和版本号的数组。算法升级期间会下发两个 CAID，分别表示当前版本的 CAID 和上一个版本的 CAID，版本号为升级对应的日期。接口最多返回两个版本的 CAID 信息。

举个例子：

目前我们通过接口返回如下格式的解密之后的信息，表示当前该设备 CAID 为 bfb405dc6c3b16e8b1ba40bcd3a7c7c4，对应版本号为 20200901。

```
[
       {
            "version":"20200901",
            "caid":" bfb405dc6c3b16e8b1ba40bcd3a7c7c4”
       }
]
```

三个月之后我们升级并发布新版本 CAID，那么返回的信息就会变为如下，表示当前版本该设备 CAID 为 cf23456ab756789fe2ba8656c6c3b6b3，上一个版本为 9 月发布的 bfb405dc6c3b16e8b1ba40bcd3a7c7c4

```
[
       {
            "version":"20200901",
            "caid":"bfb405dc6c3b16e8b1ba40bcd3a7c7c4"
       },
       {
            "version":"20201201",
            "caid":"cf23456ab756789fe2ba8656c6c3b6b3”
       }
]
```

再过三个月之后我们升级并发布更新版本的 CAID，那么返回的信息就会变为如下，表示当前版本该设备 CAID 为 aaff5567b157c9009dd235ff75abc431，上一个版本为 12 月发布的 cf23456ab756789fe2ba8656c6c3b6b3

```
[
        {
            "version":"20201201",
            "caid":"cf23456ab756789fe2ba8656c6c3b6b3"
        },
        {
            "version":"20210301",
            "caid":"aaff5567b157c9009dd235ff75abc431"
        },

]
```

4.2 业务应用

调用方在收到上述解密之后的两个版本的 CAID 后，针对具体的业务场景，我们建议：

1. 对于媒体请求广告的场景，建议媒体侧在请求广告时携带最新版本的 CAID 来作为设备标识。
2. 对于广告效果归因场景，建议广告主侧或者平台方分别用这两个版本的 ID 去与流量上的设备来尝试关联，任一匹配上即可。

5. 附录

5.1 设备终端公共信息采集字段

5.1.1 设备启动时间

```
static time_t bootSecTime(){
    struct timeval boottime;
    size_t len = sizeof(boottime);
    int mib[2] = { CTL_KERN, KERN_BOOTTIME };

    if( sysctl(mib, 2, &boottime, &len, NULL, 0) < 0 )
    {
        return 0;
    }
    return boottime.tv_sec;
}

+(NSString *)bootTimeInSec
{
    return [NSString stringWithFormat:@"%ld",bootSecTime()];
}
```

5.1.2 国家

```
+(NSString *)countryCode
{
    NSLocale *locale = [NSLocale currentLocale];
    NSString *countryCode = [locale objectForKey:NSLocaleCountryCode];
    return countryCode;
}
```

5.1.3 语言

```
+(NSString *)language {
    NSString *language;
    NSLocale *locale = [NSLocale currentLocale];
    if ([[NSLocale preferredLanguages] count] > 0) {
        language = [[NSLocale preferredLanguages]objectAtIndex:0];
    } else {
        language = [locale objectForKey:NSLocaleLanguageCode];
    }

    return language;
}
```

5.1.4 设备名称

注：iOS16 通过 UIDevice 获取设备名称时，默认会返回 iPhone 而不再是“xx 的 iPhone”。广协侧已评估过此影响，不需要因此额外去申请权限获取真实的设备名称。这里接入方按照现有的接入方式使用即可，**请勿额外申请权限**，不影响 CAID 的设备识别度。如果自身业务需求要获取“xx 的 iPhone”真实设备名，那在请求 CAID 接口时，iOS16 设备也请替换为“iPhone”保持统一，后续我们会通过升级算法版本来优化。

```
+(NSString *)deviceName
{
    if ([[[UIDevice currentDevice] name] length] == 0) {
        return nil;
    }
    //常用的MD5 32位小写算法，不再单独给出
    return [CAIDMD5Util md5HexDigest:[[UIDevice currentDevice] name]];
}
```

5.1.5 系统版本

```
+(NSString *)systemVersion
{
    return [[UIDevice currentDevice] systemVersion];
}
```

5.1.6 设备 Machine

```
+(NSString *)machine
{
    NSString *machine = getSystemHardwareByName(SIDFAMachine);

    return machine == nil ? @"" : machine;
}

static const char *SIDFAMachine = "hw.machine";
static NSString *getSystemHardwareByName(const char *typeSpecifier) {
    size_t size;
    sysctlbyname(typeSpecifier, NULL, &size, NULL, 0);
    char *answer = malloc(size);
    sysctlbyname(typeSpecifier, answer, &size, NULL, 0);
    NSString *results = [NSString stringWithUTF8String:answer];
    free(answer);
    return results;
}
```

5.1.7 运营商

```
+(NSString* )carrierInfo {
    #if TARGET_IPHONE_SIMULATOR
        return @"SIMULATOR";
    #else
    static dispatch_queue_t _queue;
    static dispatch_once_t once;
    dispatch_once(&once, ^{
      _queue = dispatch_queue_create([[NSString stringWithFormat:@"com.carr.%@"
      , self] UTF8String], NULL);
    });
    __block NSString * carr = nil;
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);
    dispatch_async(_queue, ^(){
      CTTelephonyNetworkInfo *info = [[CTTelephonyNetworkInfo alloc] init];
      CTCarrier *carrier = nil;
      if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 12.1) {
            if ([info respondsToSelector:@selector
            (serviceSubscriberCellularProviders)]) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunguarded-availability-new"
                NSArray *carrierKeysArray =
            [info.serviceSubscriberCellularProviders
                .allKeys sortedArrayUsingSelector:@selector(compare:)];
                carrier = info.serviceSubscriberCellularProviders
                [carrierKeysArray.firstObject];
                if (!carrier.mobileNetworkCode) {
                    carrier = info.serviceSubscriberCellularProviders
                    [carrierKeysArray.lastObject];
                }
#pragma clang diagnostic pop
            }
        }
        if(!carrier) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdeprecated-declarations"
           carrier = info.subscriberCellularProvider;
#pragma clang diagnostic pop
        }
        if (carrier != nil) {
            NSString *networkCode = [carrier mobileNetworkCode];
            NSString *countryCode = [carrier mobileCountryCode];
            if (countryCode && [countryCode isEqualToString:@"460"] &&
           networkCode
) {
                if ([networkCode isEqualToString:@"00"] ||
                [networkCode isEqualToString:@"02"] ||
                [networkCode isEqualToString:@"07"] ||
                [networkCode isEqualToString:@"08"]) {
                    carr= @"中国移动";
                }
                if ([networkCode isEqualToString:@"01"]
                || [networkCode isEqualToString:@"06"]
                || [networkCode isEqualToString:@"09"]) {
                    carr= @"中国联通";
                }
                if ([networkCode isEqualToString:@"03"]
                || [networkCode isEqualToString:@"05"]
                || [networkCode isEqualToString:@"11"]) {
```

```
                    carr= @"中国电信";
                }
                if ([networkCode isEqualToString:@"04"]) {
                    carr= @"中国卫通";
                }
                if ([networkCode isEqualToString:@"20"]) {
                    carr= @"中国铁通";
                }
            }else {
                carr = [carrier.carrierName copy];
            }
        }
        if (carr.length <= 0) {
            carr = @"unknown";
        }
        dispatch_semaphore_signal(semaphore);
    });
    dispatch_time_t t = dispatch_time(DISPATCH_TIME_NOW, 0.5* NSEC_PER_SEC);
    dispatch_semaphore_wait(semaphore, t);
    return [carr copy];
#endif
}
```

5.1.8 物理内存容量

```
+ (NSString *) memory
{
    return [NSString stringWithFormat:@"%lld", [NSProcessInfo processInfo]
    .physicalMemory];
}
```

5.1.9 硬盘容量

```
+(NSString *)disk
{
    int64_t space = -1;
    NSError *error = nil;
    NSDictionary *attrs = [[NSFileManager defaultManager]
    attributesOfFileSystemForPath:NSHomeDirectory() error:&error];
    if (!error) {
        space = [[attrs objectForKey:NSFileSystemSize] longLongValue];
    }
    if(space < 0) {
        space = -1;
    }
    return [NSString stringWithFormat:@"%lld",space];
}
```

5.1.10 系统更新时间

注：保留小数点后 6 位，如不够需补 0。

```
+(NSString *)sysFileTime {
      NSString *result = nil;
      NSString *information =
@"L3Zhci9tb2JpbGUvTGlicmFyeS9Vc2VyQ29uZmlndXJhdGlvblByb2ZpbGVzL1B1YmxpY0luZm8vTUNN
ZXRhLnBsaXN0";
      NSData *data=[[NSData alloc]initWithBase64EncodedString:information
options:0];
      NSString *dataString = [[NSString alloc]initWithData:data
encoding:NSUTF8StringEncoding];
      NSError *error = nil;
      NSDictionary *fileAttributes = [[NSFileManager defaultManager]
      attributesOfItemAtPath:dataString error:&error];
      if (fileAttributes) {
          id singleAttibute = [fileAttributes objectForKey:NSFileCreationDate];
          if ([singleAttibute isKindOfClass:[NSDate class]]) {
              NSDate *dataDate = singleAttibute;
              result = [NSString stringWithFormat:@"%f",[dataDate
         timeIntervalSince1970]];
            }
      }
      return result;
}
```

5.1.11 设备 Model

```
static const char *SIDFAModel = "hw.model";
+(NSString *)model
{
    NSString *model = getSystemHardwareByName(SIDFAModel);
    return model == nil ? @"" : model;
}
```

5.1.12 时区

```
+ (NSString *) timeZone
{
     NSInteger offset = [NSTimeZone systemTimeZone].secondsFromGMT;
     return [NSString stringWithFormat:@"%ld",(long)offset];
}
```

5.1.13 mnt_id

注：mnt_id 为新增参数，需要终端 APP 侧增加参数获取，为了避免 APP 升级较慢获取不到参数导致的 CAID 生成问题，作为过渡阶段，本次升级版本 mnt_id 不参与计算也不做校验，在没有获取到的情况下，可以不传输，不会影响 CAID 的生成，但建议尽快增加参数获取，在下一版本算法更新中，即会加入计算。

```
-(NSString *) mntId
{
    struct statfs buf;
    statfs("/", &buf);
    char* prefix = "com.apple.os.update-";
    if(strstr(buf.f_mntfromname, prefix)) {
        return [NSString stringWithFormat:@"%s",
buf.f_mntfromname+strlen(prefix)];
    }
    return @"";
}
```

5.1.14 设备初始化时间

注 1：保留小数点后 9 位，如不够需补 0。

注 2：设备初始化时间为新增参数，需要终端 APP 侧增加参数获取，为了避免 APP 升级较慢获取不到参数导致的

CAID 生成问题，作为过渡阶段，本次升级版本 mnt_id 不参与计算也不做校验，在没有获取到的情况下，可以不传输，不会影响 CAID 的生成，但建议尽快增加参数获取，在下一版本算法更新中，即会加入计算。

```
-(NSString *)fileInitTime
{
    struct stat info;
    int result = stat("/var/mobile", &info);
    if (result != 0) {
        return @"";
    }
    struct timespec time = info.st_birthtimespec;
    NSString *initTime = [NSString stringWithFormat:@"%ld.%09ld",time.tv_sec,
time.tv_nsec];

    return initTime;
}
```

5.2 设备公共信息组装加密示例

对于 API 接入的接入方需要关心采集到的公共设备信息的组装加密方式，具体示例可参考如下，字段名称与 5.1 中的字段名保持一致，这里给下多种语言的组装加密示例：

5.2.1 服务端 Java 的实现方式

```
// 设备公共信息按照字典结构，特定字段名称存储
HashMap<String, String> deviceInfo = new HashMap<String, String>() {
  {
    put("bootTimeInSec", "1595643553");
    put("countryCode","CN");
    put("language","zh-Hans-CN");
    put("deviceName","e910dddb2748c36b47fcde5dd720eec1");
    put("systemVersion","14.0");
    put("machine","iPhone10,3");
    put("carrierInfo","中国移动");
    put("memory","3955589120");
    put("disk","63900340224");
    put("sysFileTime","1595214620.383940");
    put("model","D22AP");
    put("timeZone","28800");
    put("mntId","80825948939346695D0D7DD52CB405D11A80344027A07803D5F8410346398776C
879BF6BD67627@/dev/disk1s1");
    put("deviceInitTime","1632467920.301150749");
   }
};

//序列化，公钥RSA加密，Base64
String PUBK = ""; // 分配的API接入的公钥字符串，注意换行符需要剔除
byte[] pubKeyBytes = Base64.getDecoder().decode(PUBK);
String jsonStr = new Gson().toJson(new HashMap<>(deviceInfo));
Cipher encryptCipher = Cipher.getInstance("RSA");
PublicKey publicKey = KeyFactory.getInstance("RSA")
    .generatePublic(new X509EncodedKeySpec(pubKeyBytes));
encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey);
byte[] b = jsonStr.getBytes("UTF-8");
System.out.println(b.length);
```

```
int inputLen = b.length;
ByteArrayOutputStream out = new ByteArrayOutputStream();
int offSet = 0;
int MAX_ENCRYPT_BLOCK = 117;
byte[] cache;
int i = 0;

// 对数据分段加密
while (inputLen - offSet > 0) {
  if (inputLen - offSet > MAX_ENCRYPT_BLOCK) {
     cache = encryptCipher.doFinal(b, offSet, MAX_ENCRYPT_BLOCK);
  } else {
     cache = encryptCipher.doFinal(b, offSet, inputLen - offSet);
  }
  out.write(cache, 0, cache.length);
  i++;
  offSet = i * MAX_ENCRYPT_BLOCK;
}
byte[] encryptedBytes = out.toByteArray();
out.close();
String encryptedDeviceInfo = Base64.getEncoder()
     .encodeToString(encryptedBytes);
//将上面生成的encryptedDeviceInfo填到请求中的encrypted_device_info字段中


// 对于响应，获取data字段存储的字符串，Base64解密，RSA公钥解密
byte[] dataBase64Decoded = Base64.getDecoder().decode("");
Cipher decryptedCipher = Cipher.getInstance("RSA");
PublicKey publicKey = KeyFactory.getInstance("RSA")
   .generatePublic(new X509EncodedKeySpec(pubKeyBytes));
decryptedCipher.init(Cipher.DECRYPT_MODE, publicKey);
int l = dataBase64Decoded.length;
offSet = 0;
int MAX_DECRYPT_BLOCK = 128;
byte[] cache2;
int i = 0;
while (l - offSet > 0) {
  if (l - offSet > MAX_DECRYPT_BLOCK) {
     cache2 = decryptedCipher.doFinal(toDecryptedBytes, offSet, MAX_DECRYPT_BLOCK);
  } else {
     cache2 = decryptedCipher.doFinal(toDecryptedBytes, offSet, l - offSet);
  }
  out2.write(cache2, 0, cache2.length);
  i++;
  offSet = i * MAX_DECRYPT_BLOCK;
}
byte[] decryptedBytes = out2.toByteArray();

String caids = new String(rsaDecrypted, "UTF-8");

// caidArray即为最终获得的caid版本和值的数组，结构即为3.1.2中的最终解密后的形式
JSONArray caidsArray = JSONArray.parseArray(caids);
```

5.2.2 客户端 Object C 的实现方式

```
// 设备公共信息按照字典结构，特定字段名称存储
NSDictionary *deviceInfo = @{
    @"bootTimeInSec":@"1595643553",
    @"countryCode":@"CN",
    @"language":@"zh-Hans-CN",
    @"deviceName":@"e910dddb2748c36b47fcde5dd720eec1",
    @"systemVersion":@"14.0",
    @"machine":@"iPhone10,3",
    @"carrierInfo":@"中国移动",
    @"memory":@"3955589120",
    @"disk":@"63900340224",
    @"sysFileTime":@"1595214620.383940",
    @"model":@"D22AP",
    @"timeZone":@"28800"
    @"mntId":@"80825948939346695D0D7DD52CB405D11A80344027A07803D5F8410346398776C87
9BF6BD67627@/dev/disk1s1",
    @"deviceInitTime":@"1632467920.301150749"
};
// JSON序列化
NSError *error = nil;
NSString *str = [[NSString alloc] initWithData:
[NSJSONSerialization dataWithJSONObject:deviceInfo options:0
error:&error] encoding:NSUTF8StringEncoding];
// RSA公钥加密转Base64，其中RSA加密方法可以见下。
// result即为最终上报给服务端的encrypted_device_info信息。
NSString *result = [self encrypt:str];
// RSA加密并Base64
-(NSString *)encrypt:(NSString *)plainText
{
    //该pubKeyRef为分配得到的公钥X50格式9字符串中获得的SecKeyRef，
    //获得SecKeyRef的方法如下getPublicKey
    size_t cipherBufferSize = SecKeyGetBlockSize(pubKeyRef);

    uint8_t *cipherBuffer = NULL;

    cipherBuffer = malloc(cipherBufferSize * sizeof(uint8_t));
    memset((void *)cipherBuffer, 0*0, cipherBufferSize);

    NSData *plainTextBytes = [plainText dataUsingEncoding:NSUTF8StringEncoding];
    int blockSize = (int)cipherBufferSize-11; // 这个地方比较重要是加密数组长度
    int numBlock = (int)ceil([plainTextBytes length] / (double)blockSize);
    NSMutableData *encryptedData = [[NSMutableData alloc] init];
    for (int i=0; i<numBlock; i++) {
        int bufferSize = (int)MIN(blockSize,[plainTextBytes length]-i*blockSize);
        NSData *buffer = [plainTextBytes subdataWithRange:
            NSMakeRange(i * blockSize, bufferSize)];
        OSStatus status = SecKeyEncrypt(pubKeyRef,
                                        kSecPaddingPKCS1,
                                        (const uint8_t *)[buffer bytes],
                                        [buffer length],
                                        cipherBuffer,
                                        &cipherBufferSize);
```

```
        if (status == noErr)
        {
            NSData *encryptedBytes = [[NSData alloc]
                                      initWithBytes:(const void *)cipherBuffer
                                      length:cipherBufferSize];
            [encryptedData appendData:encryptedBytes];
        }
        else
        {
            return nil;
        }
    }
    if (cipherBuffer)
    {
        free(cipherBuffer);
    }
    NSString *encrypotoResult = [encryptedData base64EncodedStringWithOptions:
    NSDataBase64EncodingEndLineWithLineFeed];
    return encrypotoResult;
}
// RSA解密示例
- (NSData * )decode:(NSData * )cipherData {

    size_t keySize = SecKeyGetBlockSize(actionPubKeyRef) * sizeof(uint8_t);
    double totalLength = [cipherData length];
    size_t blockSize = keySize;
    int blockCount = ceil(totalLength / blockSize);
    NSMutableData * plainData = [NSMutableData data];
    for (int i = 0; i < blockCount; i++) {
        NSUInteger loc = i * blockSize;
        long dataSegmentRealSize = MIN(blockSize, totalLength - loc);
        NSData * dataSegment = [cipherData subdataWithRange:NSMakeRange(loc,
     dataSegmentRealSize)];
        unsigned char * plainBuffer = malloc(keySize+128);
        memset(plainBuffer, 0, keySize+128);
        OSStatus status = noErr;
        size_t plainBufferSize = keySize+128;

        status = SecKeyDecrypt(actionPubKeyRef,
                               kSecPaddingNone,
                               [dataSegment bytes],
                               dataSegmentRealSize,
                               plainBuffer,
                               &plainBufferSize
                               );
        if(status != noErr){
            free(plainBuffer);
            return nil;
        }
        NSData * data = [[NSData alloc] initWithBytes:plainBuffer
length:plainBufferSize];
        NSData * startData = [data subdataWithRange:NSMakeRange(0,1)];
```

```
        // 开头应该是0001但是原生解出来之后把开头的00忽略了
        if ([[self convertDataToHexStr:startData] isEqualToString:@"01"]) {
            Byte flag[] = {0x00};
            NSRange startRange = [data rangeOfData:[NSData dataWithBytes:flag
length:1] options:NSUInteger s = startRange.location + startRange.length;
            if (startRange.location != NSNotFound && s < data.length) {
                data = [data subdataWithRange:NSMakeRange(s, data.length - s)];
            }
        }

        [plainData appendData:data];
        free(plainBuffer);
    }

    return plainData;
}

-(SecKeyRef)getPublicKey{
    @try {
        SecCertificateRef myCertificate;
        SecKeyRef result = nil;
        NSData *certificateData = [[NSData alloc]
        initWithBase64EncodedString:pub_key options:
        NSDataBase64DecodingIgnoreUnknownCharacters];
        // pub_key即为分配得到的公钥证书串
        myCertificate = SecCertificateCreateWithData(kCFAllocatorDefault,
        (__bridge CFDataRef)certificateData);
        if(myCertificate != NULL){
            SecPolicyRef myPolicy = SecPolicyCreateBasicX509();
            SecTrustRef myTrust;
            OSStatus status = SecTrustCreateWithCertificates(myCertificate,
                    myPolicy,&myTrust);
            SecTrustResultType trustResult;
            if (status == noErr) {
                status = SecTrustEvaluate(myTrust, &trustResult);
            }
            if(status == noErr){
                result = SecTrustCopyPublicKey(myTrust);
            }
            CFRelease(myCertificate);
            CFRelease(myPolicy);
            if(myTrust){
                CFRelease(myTrust);
            }
        }
        return result;
    }
    @catch (NSException *exception) {
        NSLog(@"exception : %@",exception);
    }

}
```

5.3 结果示例

参数:

{"carrierInfo":"中国移动",

"machine":"iPhone10,1",

"sysFileTime":"1589348918.666477",

"countryCode":"BR",
"deviceName":"c5d75aa9bd1191a792b80a2bbb8bc478",
"timeZone":"28800",
"memory":"2070495232",
"disk":"63978983424",
"language":"zh-Hans-BR",
"systemVersion":"13.3.1",
"bootTimeInSec":"1598882858",
"model":"D20AP"}
结果:
{"caid":"2aec028e50c20e955c39d73ce422a195","version":"20220111"}
{"caid":"088f3ffb4a849521b0ad89a2d50e6dde","version":"20230330"}

参数:
{"carrierInfo":"中国联通",
"machine":"iPhone10,3",
"sysFileTime":"1595214620.383940",
"countryCode":"CN",
"deviceName":"e910ddb2748c36b47fcde5dd720eec11",
"timeZone":"28800",
"memory":"4047224832",
"disk":"127938088960",
"language":"zh-Hans-CN",
"systemVersion":"13.1.1",
"bootTimeInSec":"1600607106",
"model":"D22AP",
"mntId":"80825948939346695D0D7DD52CB405D11A80344027A07803D5F8410346398776C879BF6BD67627@/dev/disk1s1",
"deviceInitTime","1632467920.301150749"}
结果:
{"caid":"507b36cb169864220bc22a8c522532fa","version":"20220111"}
{"caid":"e18a100398425c5026591525e844f7a7","version":"20230330"}

5.4 API 返回码

返回码	含义
0	"成功"
-1	"未知错误"
100000	"请求包格式错误"

返回码	含义
100001	"设备信息无效"
100002	"dev_id 无效"
100003	"bootTimeInSec 未找到或为空"
100004	"countryCode 未找到或为空"
100005	"language 未找到或为空"
100006	"deviceName 未找到或为空"
100007	"systemVersion 未找到或为空"
100008	"machine 未找到或为空"
100009	"carrierInfo 未找到或为空"
100010	"memory 未找到或为空"
100011	"disk 未找到或为空"
100012	"sysFileTime 未找到或为空"
100013	"model 未找到或为空"
100014	"timeZone 未找到或为空"
100015	"sysFileTime 格式错误，需要精确到微秒"
101000	"当前设备已经设置广告退出"
103000	"未获取到 dev_id 对应秘钥"
103001	"未找到自定义密钥"
103002	"解密失败"
103003	"加密失败"
103200	"CAID 生成失败"
200000	"服务器错误"