游戏数据库 TcaplusDB

使用 TcaplusDB RESTful API

产品文档

腾讯云

文档目录

使用 TcaplusDB RESTful API
RESTful API 接口说明

最近更新时间：2023-12-18 14:50:46

本文档为 Tcaplus RESTful API v1.0 用户手册

概述

Tcaplus RESTful API 为开发者提供了一种通过 HTTP 请求与 Tcaplus 数据库远程交互的方式。当您通过 RESTful API 用 Json 携带数据发送 HTTP 请求后，您会收到对应的 Json 格式的响应包。开发者可以通过任何语言或工具发送 RESTful API 请求对数据进行增、删、改、查操作。

准备工作

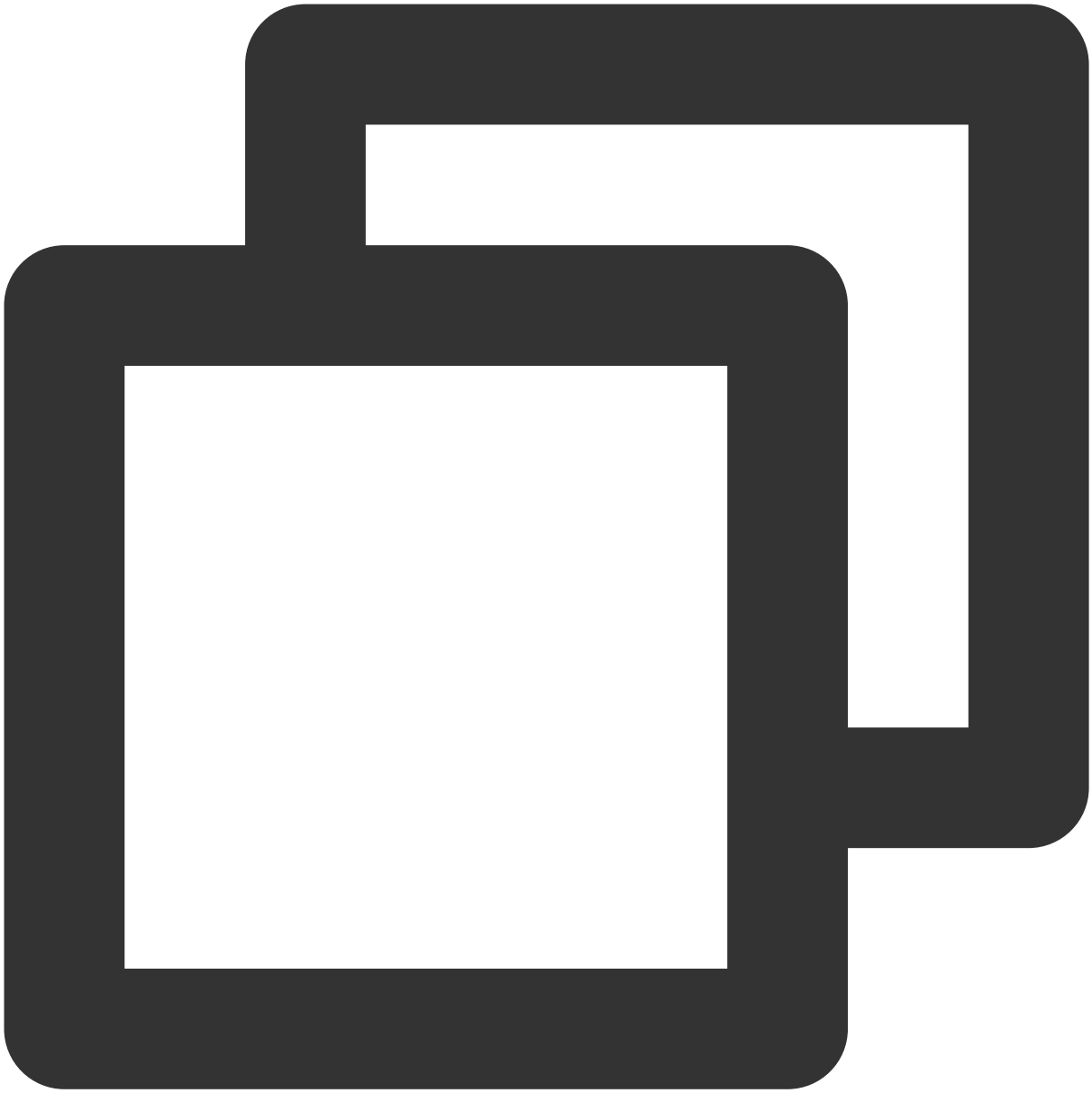
确保您已经在 腾讯云TcaplusDB 创建了集群，并且已经获取对应的集群信息（包括 AppId(集群接入 ID)，ZoneId(表格组 ID)，AppKey（集群访问密码））。当前 Tcaplus RESTful API 只支持通过 protobuf 定义的表。

当前版本

当前，所有 Tcaplus RESTful API 的请求默认都使用 ver1.0 版本。

请求

所有的 API 访问请求都是通过 HTTP，所有的数据都通过 JSON 格式传递。以下是典型的请求访问 URI 格式：



```
http://{Tcaplus_REST_URL}/{Version}/apps/{AppId}/zones/{ZoneId}/tables/{TableName}/
```

Tcaplus_REST_URL：Tcaplus RESTful URL 接入点

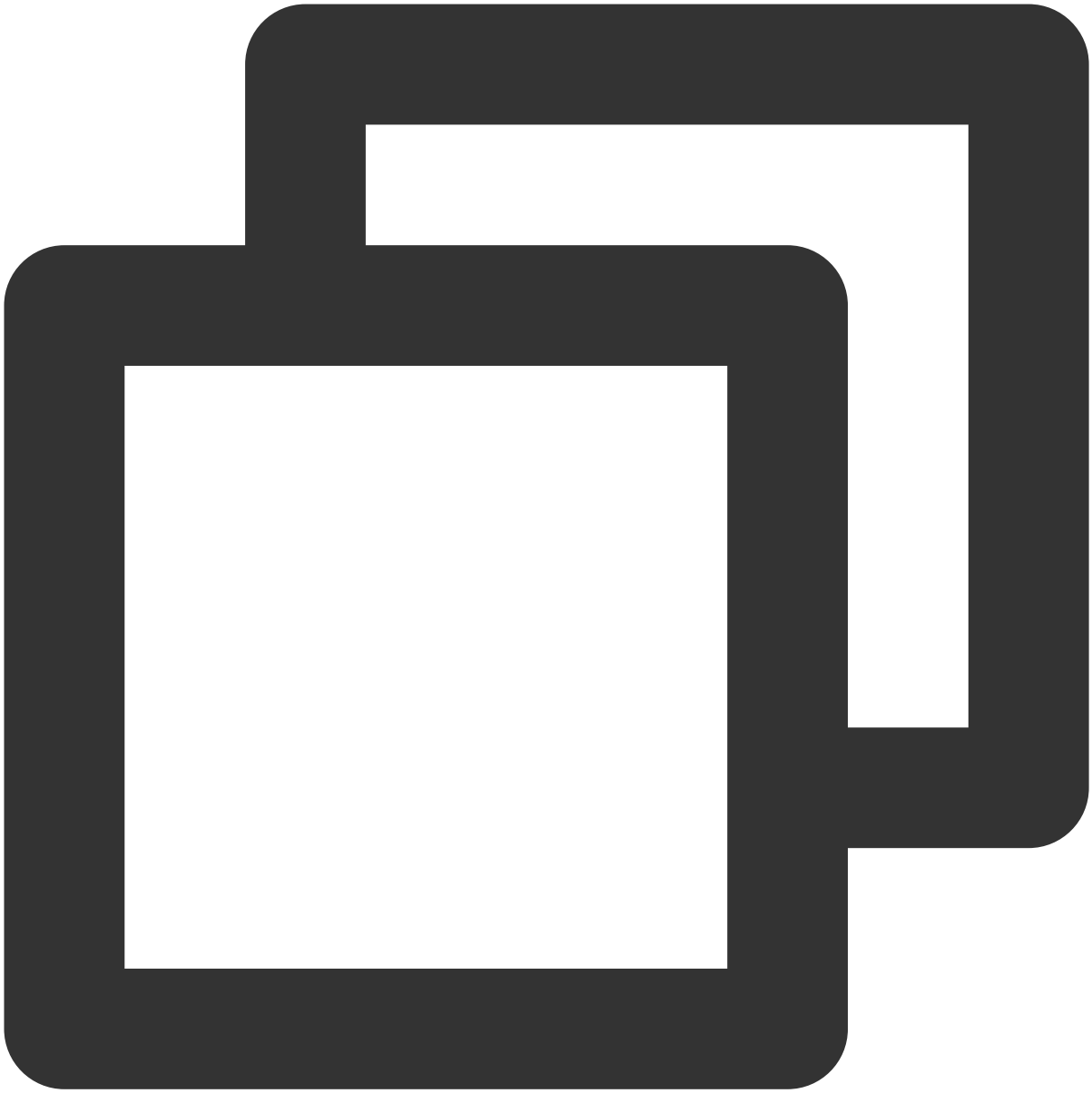
Version：Tcaplus RESTful API 版本号，默认"ver1.0"

AppId：集群的**接入 ID**

ZoneId：表格组 **ID**

TableName：表名

示例：



```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_rest_test/records
```

HTTP 头

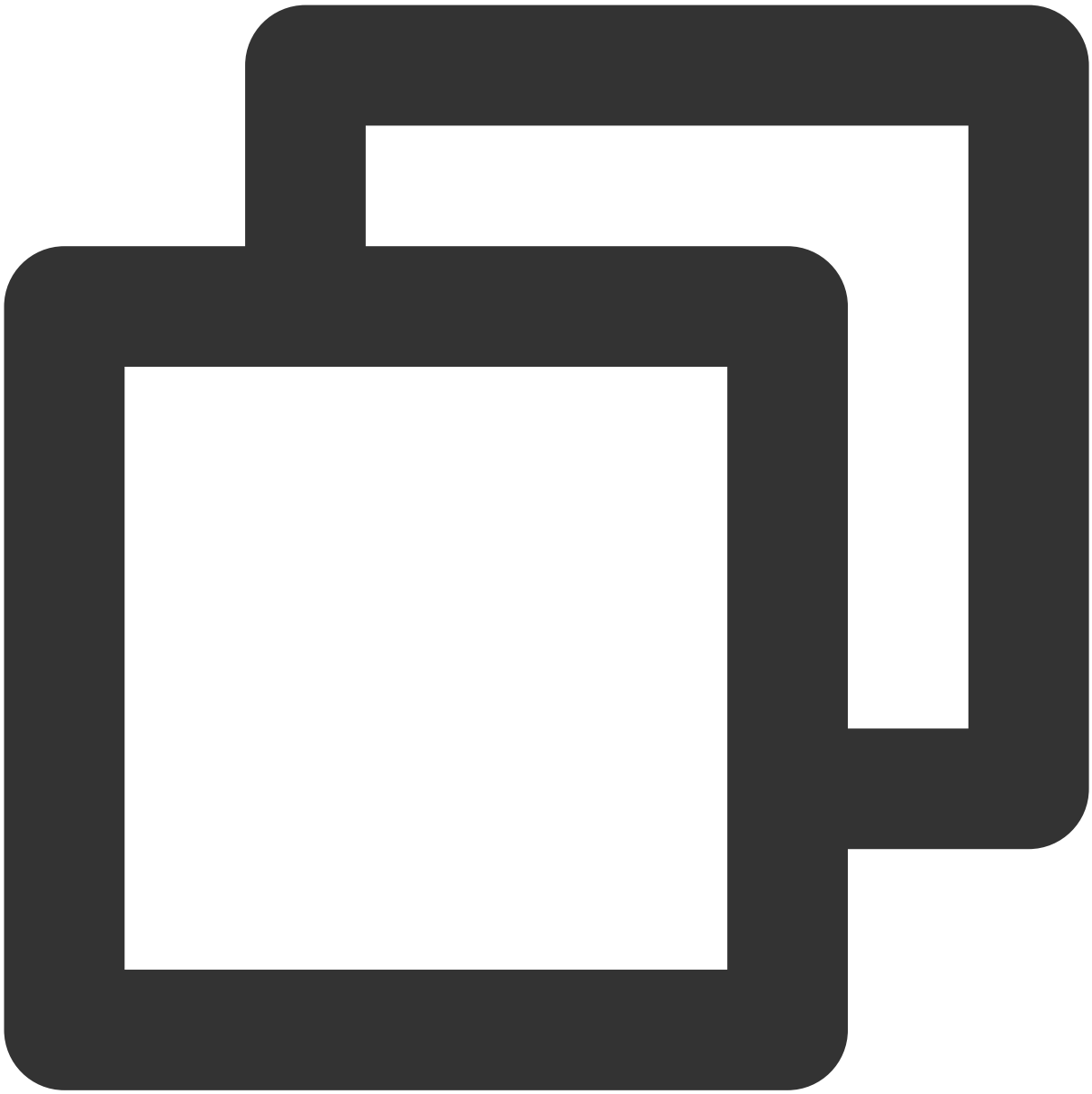
通过设置 HTTP 头可以允许用户 HTTP 客户端通过请求和回包传递一些额外的信息。

鉴权

`x-tcaplus-pwd-md5`

此字段填写用户 app key 的 MD5 计算结果用于客户端权限验证。

请通过以下 bash 命令计算字符串的 MD5 值：



```
# echo -n "c3eda5f013f92c81dda7afcdc273cf82" | md5sum
879423b88d153cace7b31773a7f46039  -
```

操作校验

`x-tcaplus-target` 指定 Tcaplus RESTful API 请求操作，具体包含以下操作类型：

Tcaplus.GetRecord 获取记录

Tcaplus.AddRecord 插入记录

Tcaplus.SetRecord 设置记录

Tcaplus.DeleteRecord 删除记录

Tcaplus.FieldGetRecord 部分字段读

Tcaplus.FieldSetRecord 部分字段设置

Tcaplus.FieldIncRecord 部分字段自增

Tcaplus.PartkeyGetRecord 索引批量读

`x-tcaplus-idl-type` 指定 Tcaplus 表类型，目前只支持 protobuf（pb）类型。

设置标记

`x-tcaplus-data-version-check` Specifies Tcaplus data version check policy use with `x-tcaplus-data-version` . It can be set as:

指定 Tcaplus 数据版本号校验策略，实现乐观锁功能，与 `x-tcaplus-data-version` 标记配合使用，可选值如下：

`1` ：当设置为1，客户端的数据版本号必须与存储层数据版本号一致才能写操作，操作会令数据版本号+1。

`2` ：当设置为2，就不检测客户端版本号与服务端版本号之间的关系，强制将客户端传入的版本号设置到存储层。

`3` ：当设置为3，就不检测客户端版本号与服务端版本号之间的关系，写操作会将存储层数据版本号+1。

此标记仅对 Tcaplus.AddRecord 和 Tcaplus.SetRecord 有意义。

`x-tcaplus-data-version` 与 `x-tcaplus-data-version-check` 标记相配合，设置客户端的数据版本号。可选值如下：

version <= 0 忽略版本检查策略。

version > 0 指定客户端数据记录版本号。

`x-tcaplus-result-flag` 设置应答中是否包含完整数据的策略，可能的取值有：

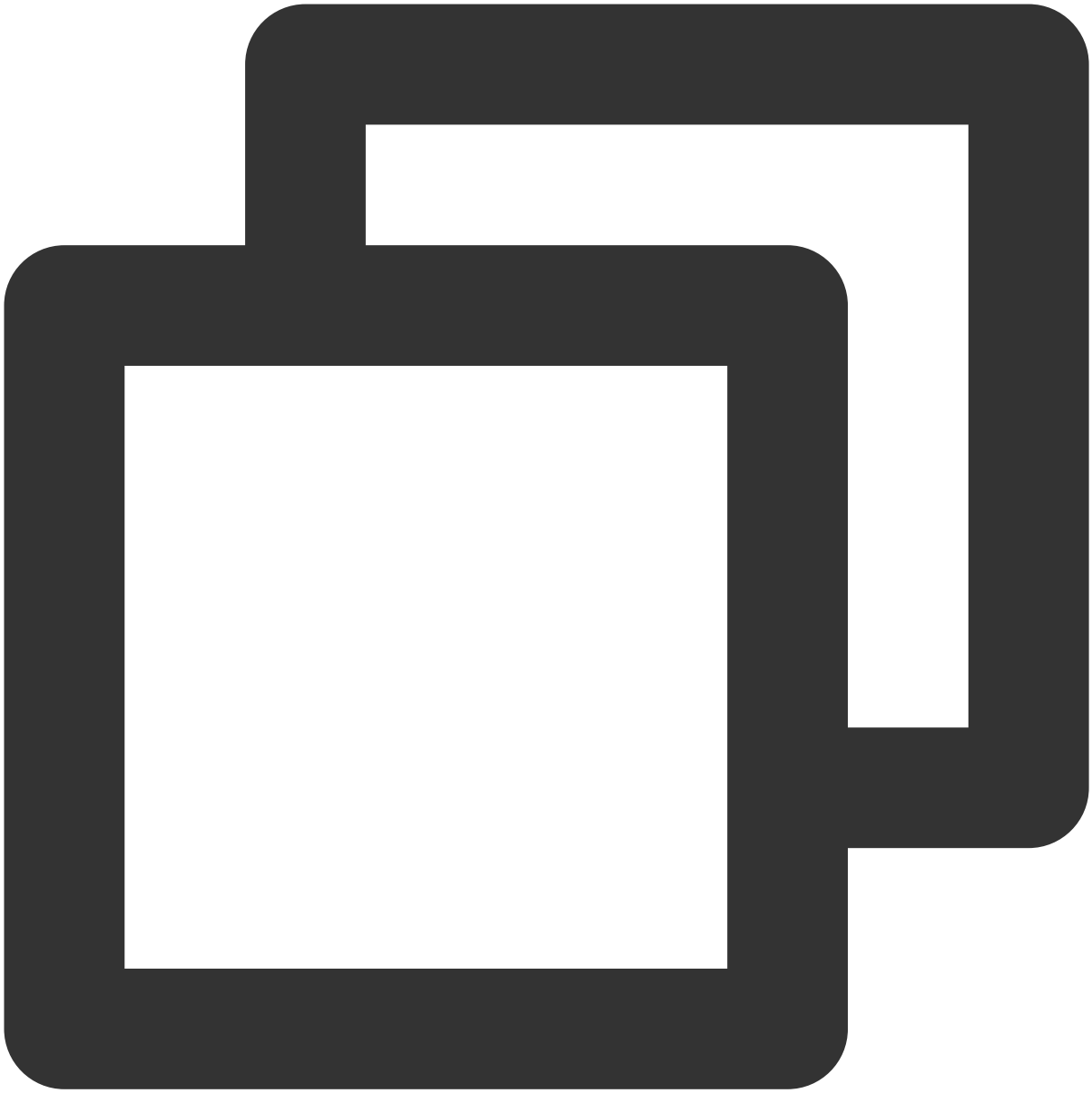
`0` 设置为0，应答中仅包含请求成功或失败。

`1` 设置为1，应答中包含与请求一致的值。

`2` 设置为2，应答中包含被修改的数据的所有字段最新值。

`3` 设置为3，应答中包含记录被修改前的值。

请求 JSON 数据

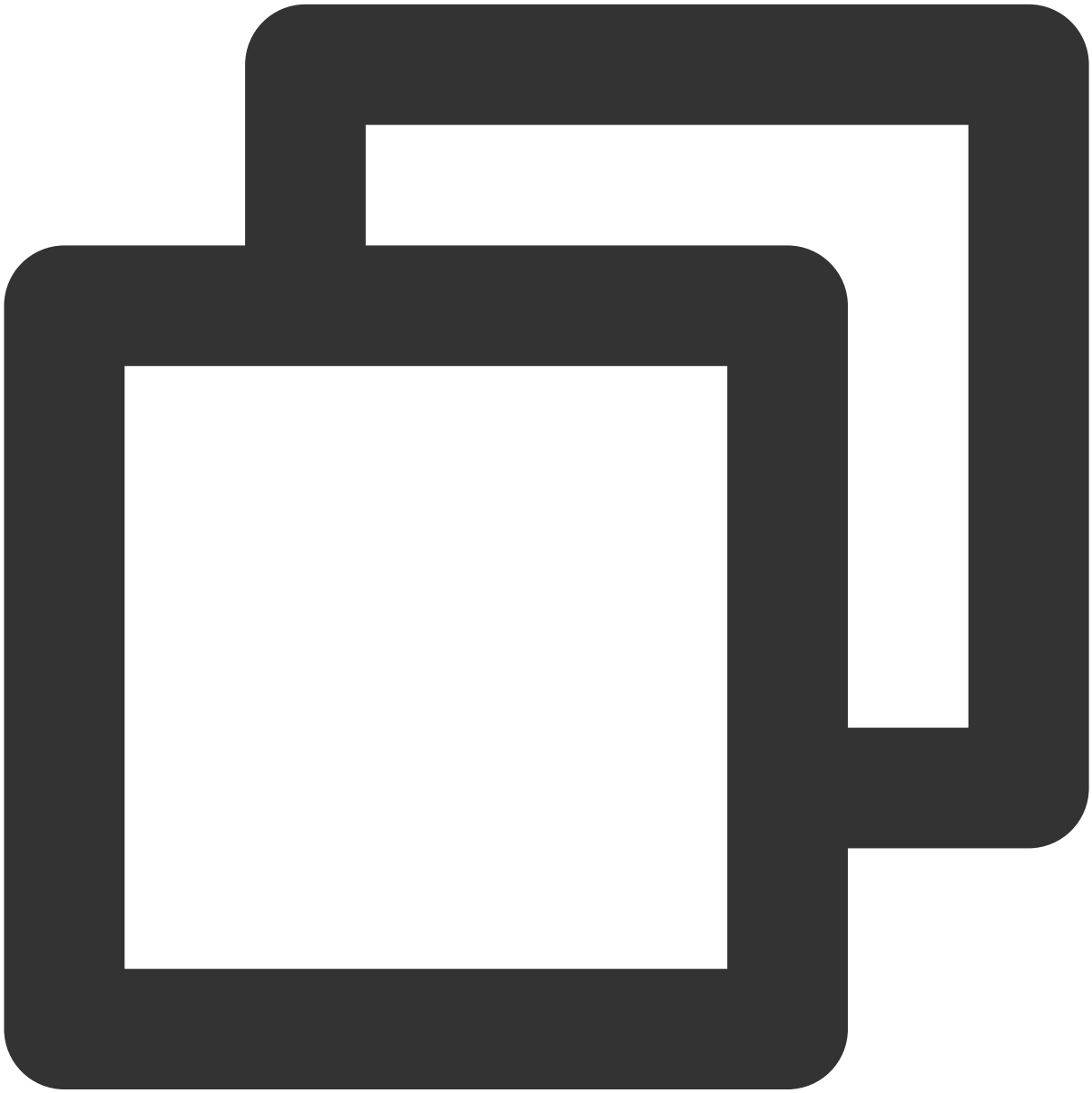


```
Request Data:
{
 "ReturnValues": "...", // 用户设置的保留数据，随请求到达tcaplus并由应答原样带回
 "Record": {
     ... // 数据记录，详细格式请参见“API简明示例”一节
 }
}
```

应答

应答中的 JSON 数据

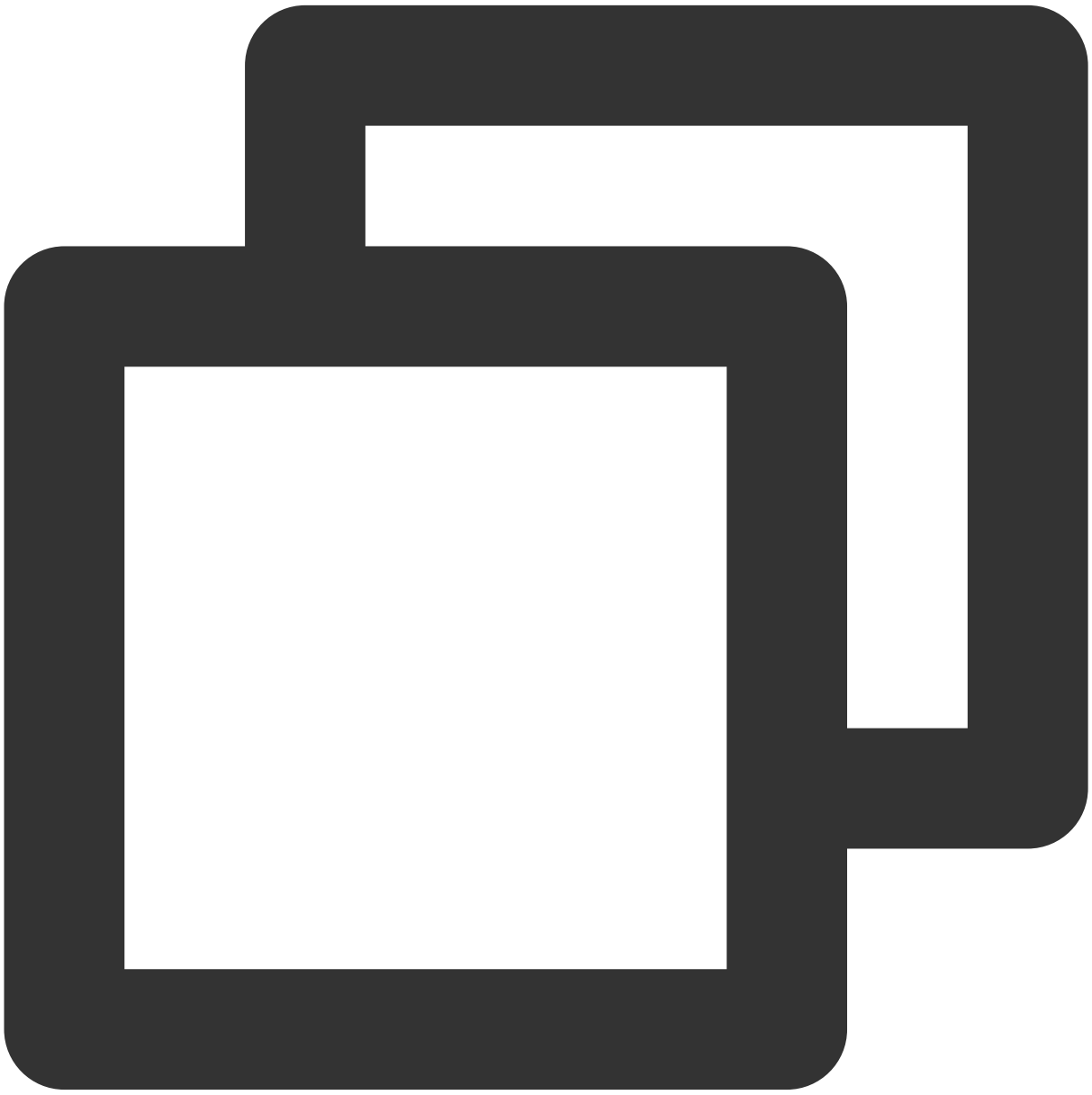
GetRecord，SetRecord，AddRecord，DeleteRecord，FieldGetRecord，FieldSetRecord 和 FieldIncRecord 操作的结果将会返回单条数据，应答 JSON 数据格式如下：



```
Response Data
{
 "ErrorCode": 0, // 返回码
 "ErrorMsg": "Succeed", // 返回信息
```

```
  "RecordVersion": 1, // 数据版本号
  "ReturnValues": "...", // 用户设置的保留数据，随请求到达tcaplus并由应答原样带回
  "Record": { // 数据记录，详细格式请参见“API简明示例”一节
      ...
  }
}
```

PartkeyGetRecord 操作的结果有可能带回多条数据，应答的 JSON 数据格式如下：



```
Response Data
{
  "ErrorCode": 0, // 返回码
```

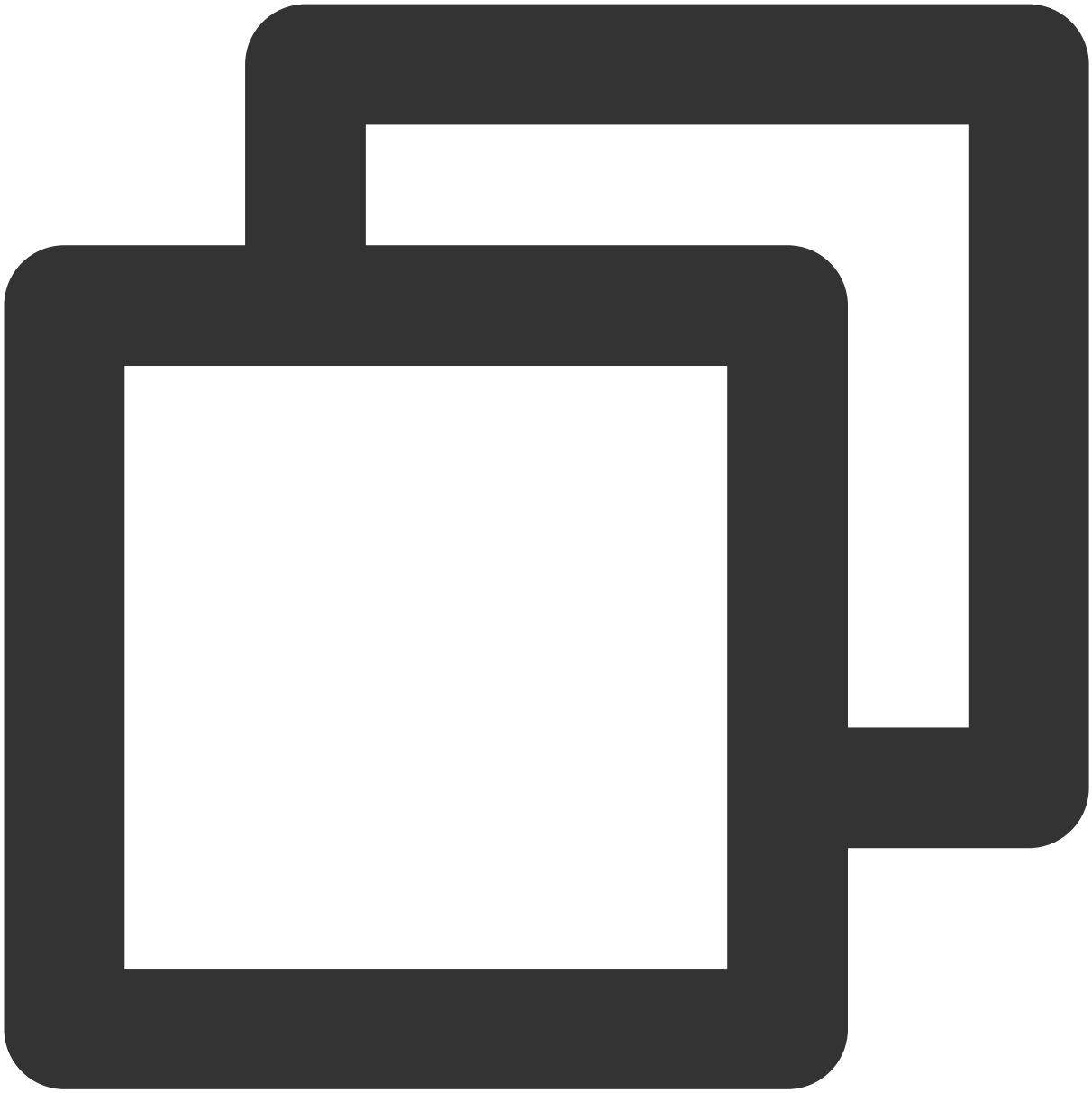
```
  "ErrorMsg": "Succeed", // 返回信息
  "MultiRecords": [ //多条数据构成的数组
   {"RecordVersion": 1,  //每条数据的版本号
    "Record": {...}  // 数据记录，详细格式请参见“API简明示例”一节
   },
   ...
   ],
  "RemainNum": 0, //还未访问到的数据条数
  "TotalNum": 5   //满足条件的总数据条数
}
```

返回码

HTTP 状态码	返回码	返回信息
200	0	成功
400	-2579	数据反序列化错误
401	-279	鉴权错误
400	-11539	非法请求 详细原因请参考返回信息字段
400	-2067	操作类型不匹配
400	-1811	非法参数
400	-5395	没有设置主键字段
400	-2323	数据序列化错误
400	-7949	非法数据版本号 一般发生在写操作，应用数据版本检查策略时
400	-1293	记录已经存在
404	261	记录不存在
404	-34565	索引不存在
500	-	系统内部错误 请参考返回信息字段

API 简明示例

Tcaplus.GetRecord



```
GET /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records?keys={JSONKey
```

从一个 Tcaplus pb 表中通过指定一条记录的 key 信息查询此记录。这个操作将会把整条记录取出，但您可以设置 select 变量，select 变量可指定您需要在应答中返回的字段，如果 select 变量不指定，将会显示所有字段信息。如果数据记录不存在，将会返回错误。

必须在 URI 中指定 `keys` 变量，而 select 变量则是可选项。keys 指所有主键的值，select 指需要显示的 value 字段的名称。并且您可以通过点分路径的方式指定嵌套结构中的字段，例如：“pay.total_money”。

注意：

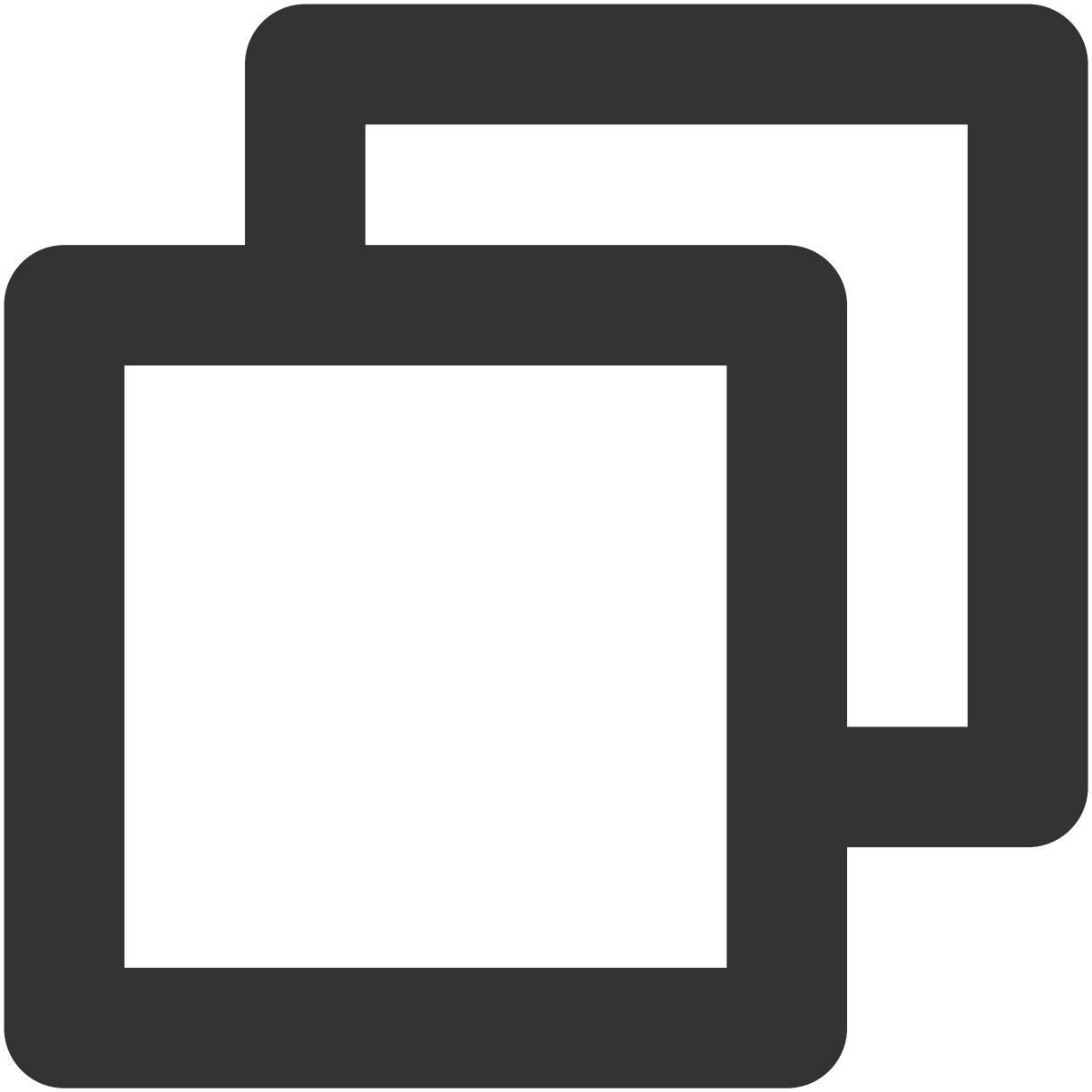
请求的变量必须通过 UrlEncode 编码，请将url中的空格编码为"%20"而不是"+"。

名称	类型	取值
x-tcaplus-target	String	Tcaplus.GetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-idl-type	String	protobuf

示例：

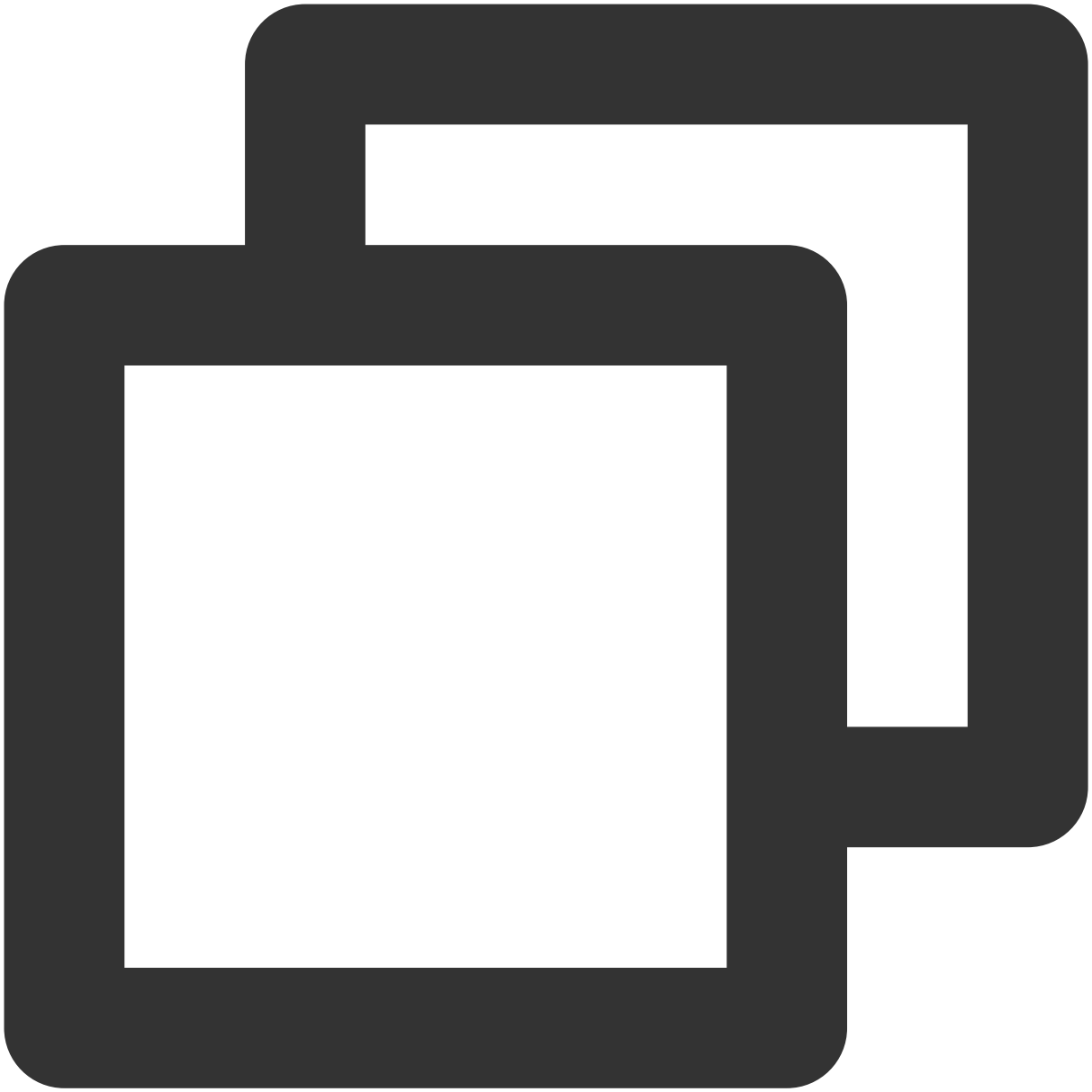
URL:

URL 未 UrlEncode 结果：



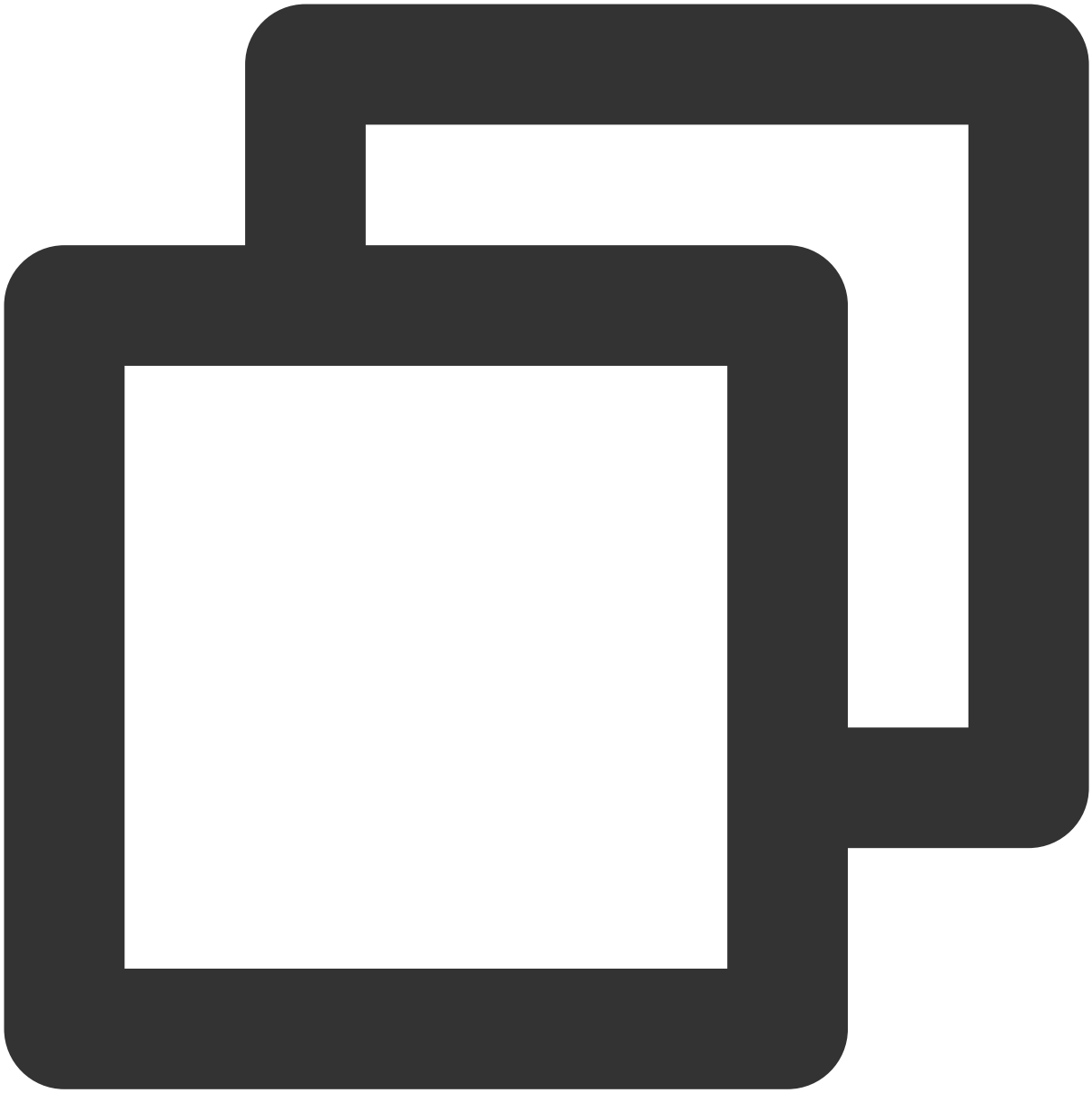
```
http://10.123.9.70:31002/ver1.0/apps/2/zones/1/tables/tb_example/records?keys={'reg
```

URL UrlEncode 结果：



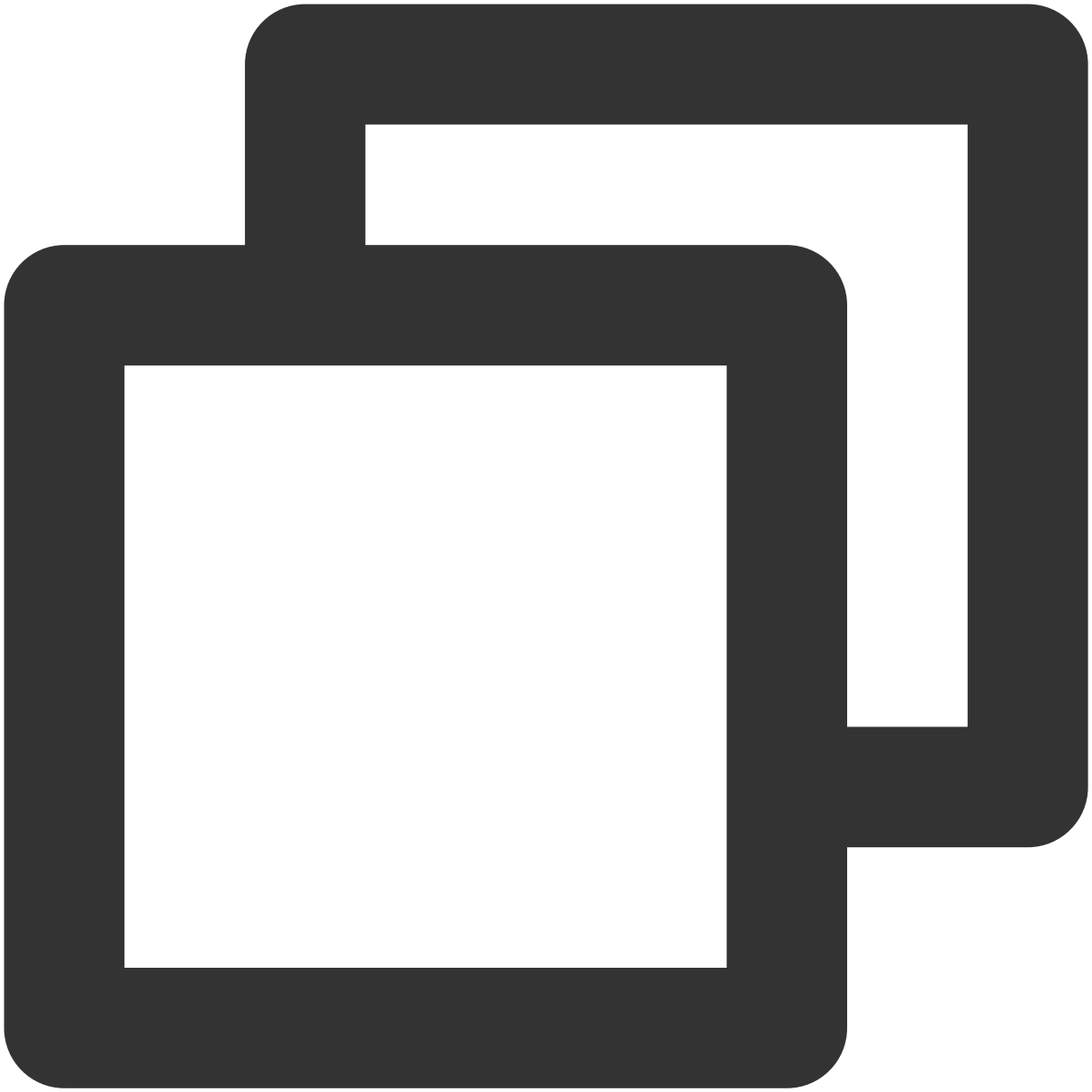
```
http://10.123.9.70:31002/ver1.0/apps/2/zones/1/tables/tb_example/records?keys=%7B%2
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.GetRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-idl-type:protobuf"
]
```

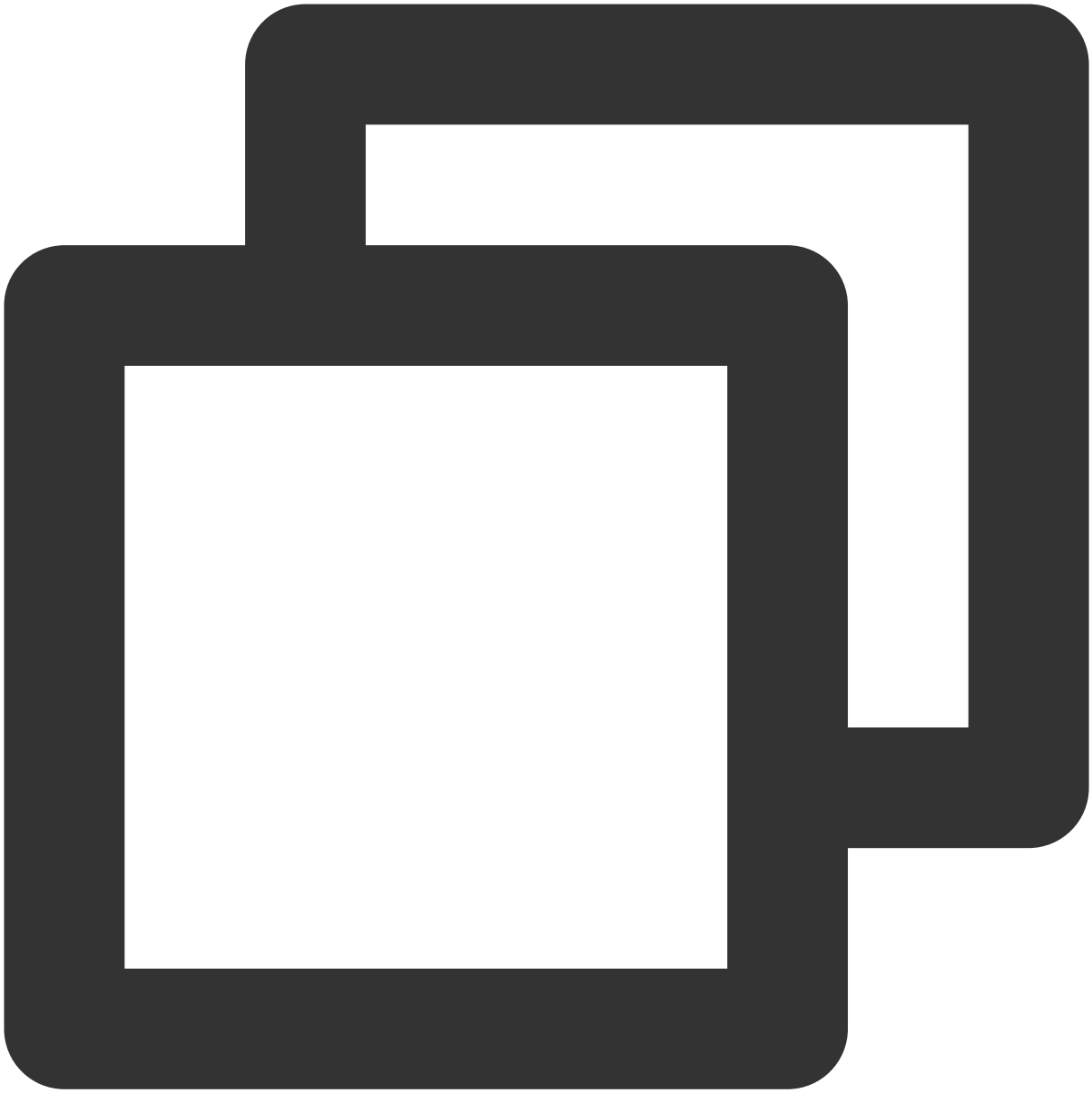
应答数据：




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "RecordVersion": 1,
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70
  ],
```

```
   "pay": {
    "pay_times": 2,
    "total_money": 10000,
    "pay_id": 5,
    "auth": {
     "pay_keys": "adqwacsasafasda",
     "update_time": 1528018372
    }
   },
   "region": 101,
   "uin": 100,
   "is_available": true,
   "gamesvrid": 4099,
   "logintime": 404
  }
 }
```

Tcaplus.SetRecord



```
PUT /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records
```

通过指定一条记录的 key 信息设置此记录。如果记录存在执行覆盖操作，否则，执行插入操作。

SetRecord 操作支持 resultflag 设置以下取值：

`0` 设置为0，应答中仅包含请求成功或失败

`1` 设置为1，应答中包含与请求一致的值

`2` 设置为2，应答中包含被修改的数据的所有字段最新值

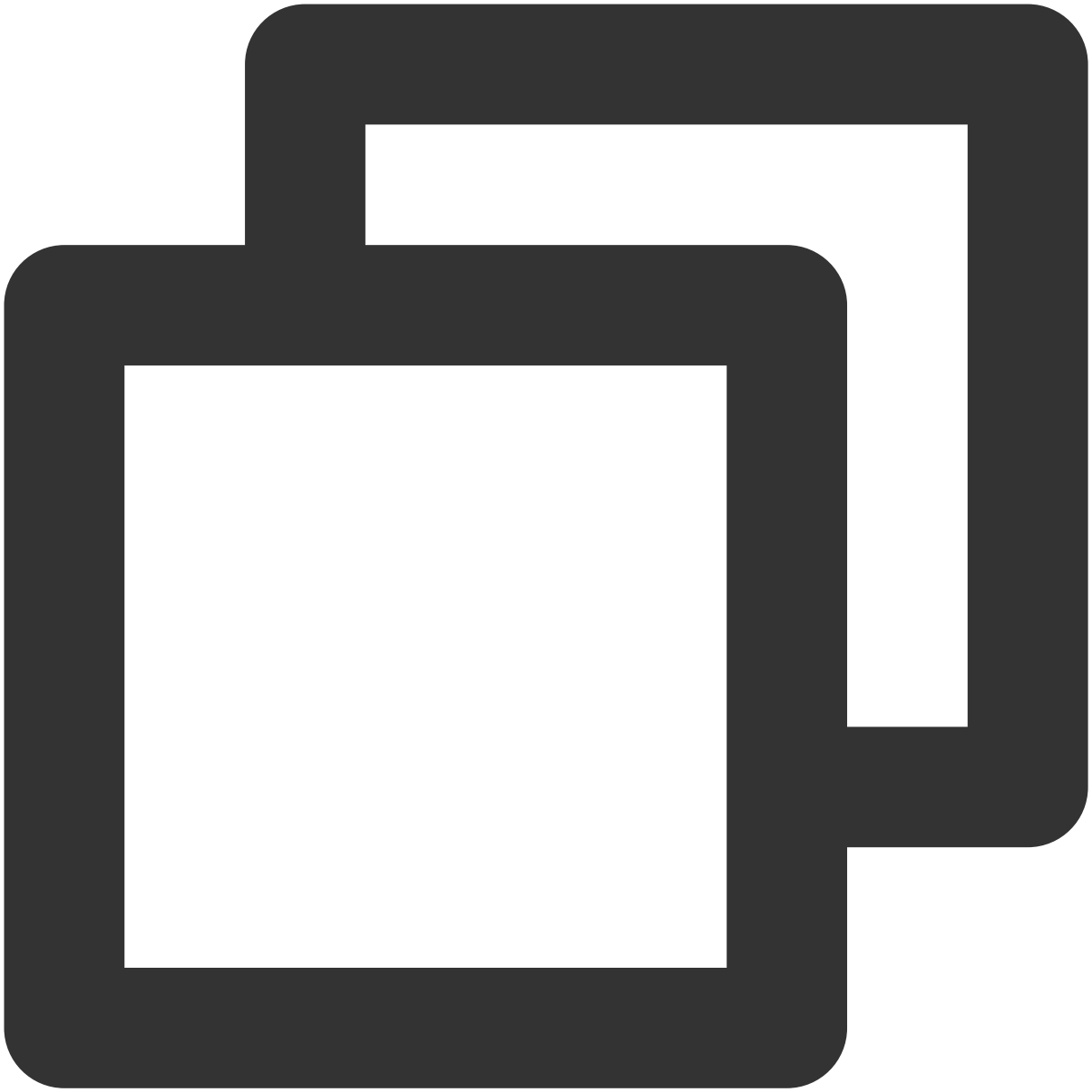
`3` 设置为3，应答中包含记录被修改前的值

名称	类型	取值

x-tcaplus-target	String	Tcaplus.SetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-result-flag	Int	2
x-tcaplus-data-version-check	Int	2
x-tcaplus-data-version	Int	-1
x-tcaplus-idl-type	String	protobuf

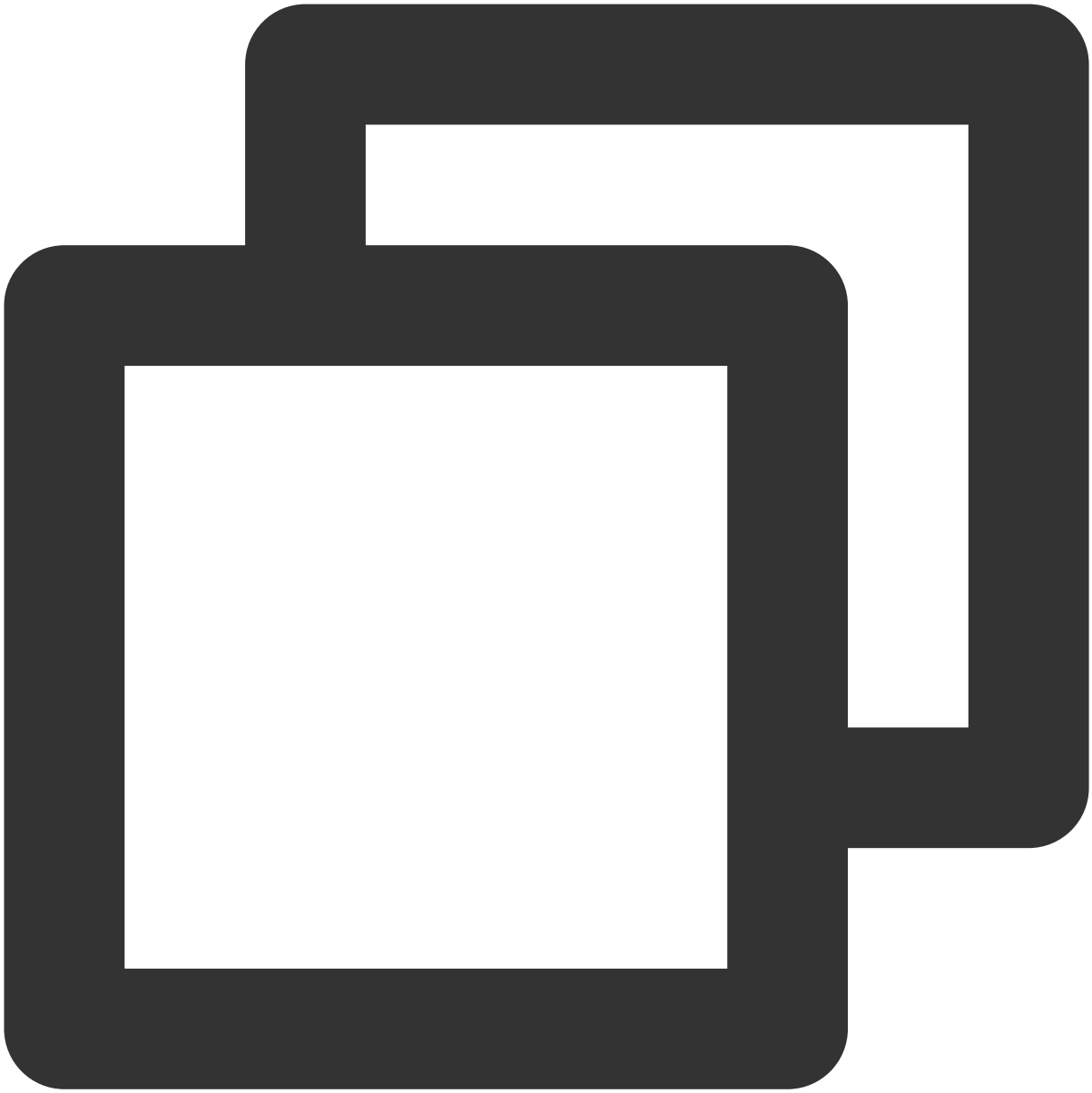
示例：

URL:



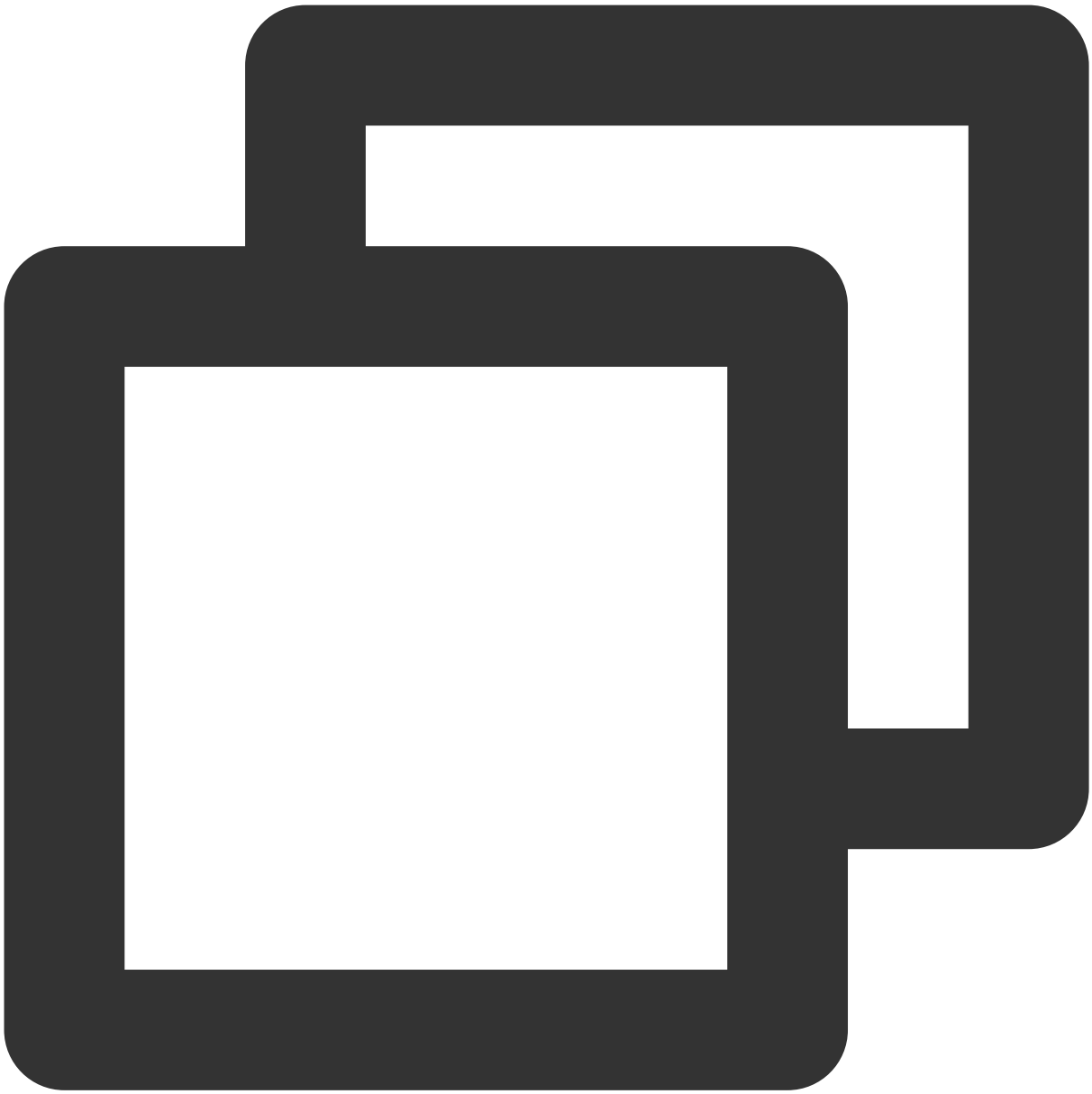
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.SetRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-result-flag:2",
 "x-tcaplus-data-version-check:2",
 "x-tcaplus-data-version:-1",
 "x-tcaplus-idl-type:Protobuf"
]
```

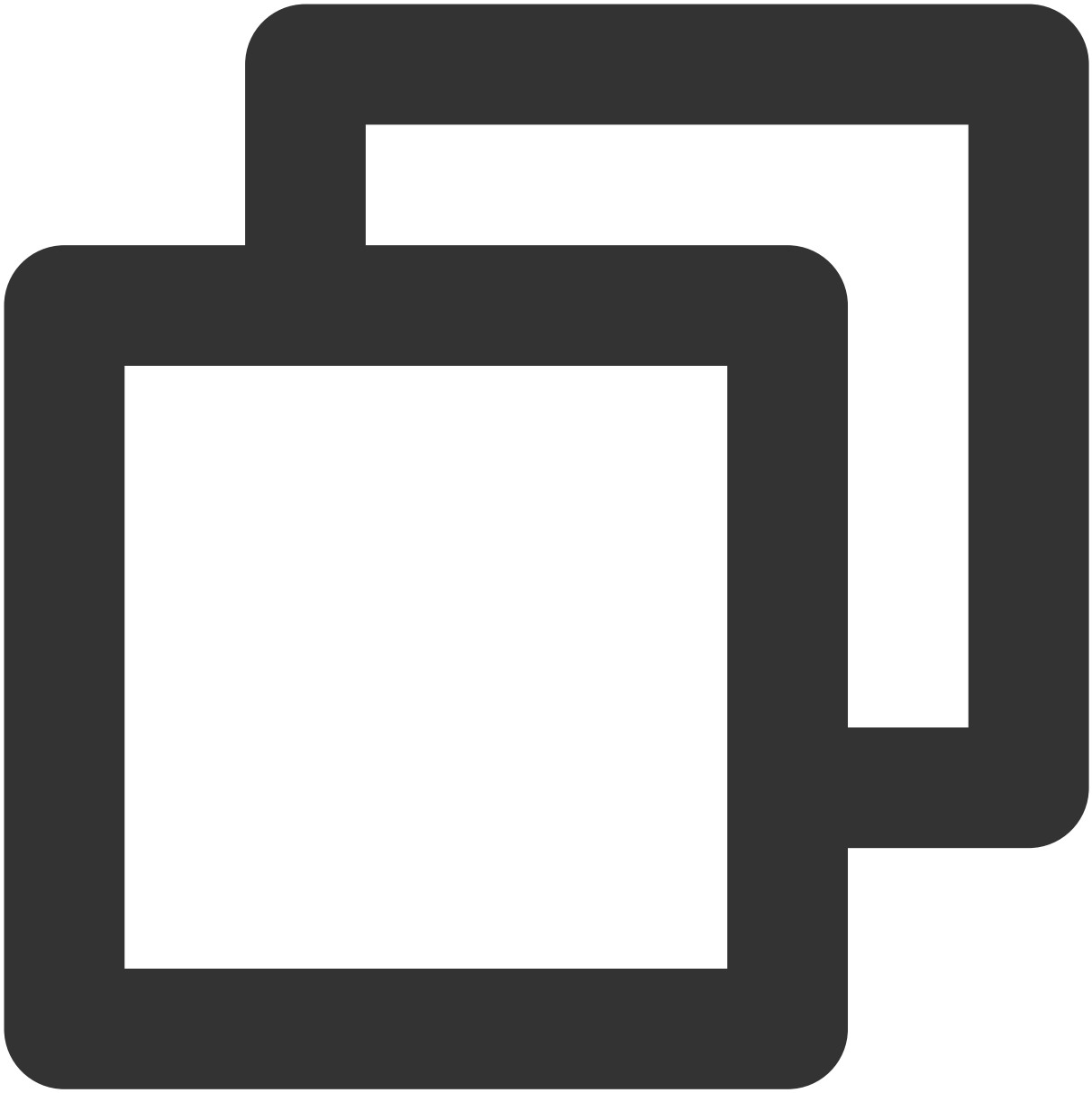
请求 JSON 数据：



```
{
 "ReturnValues": "Send to tcaplus by calvinshao",
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70,
   80,
```

```
    90,
    100
   ],
   "pay": {
    "pay_times": 3,
    "total_money": 12000,
    "pay_id": 5,
    "auth": {
     "pay_keys": "adqwacsasafasda",
     "update_time": 1528018372
    }
   },
   "region": 101,
   "uin": 100,
   "is_available": false,
   "gamesvrid": 4099,
   "logintime": 404
  }
 }
```

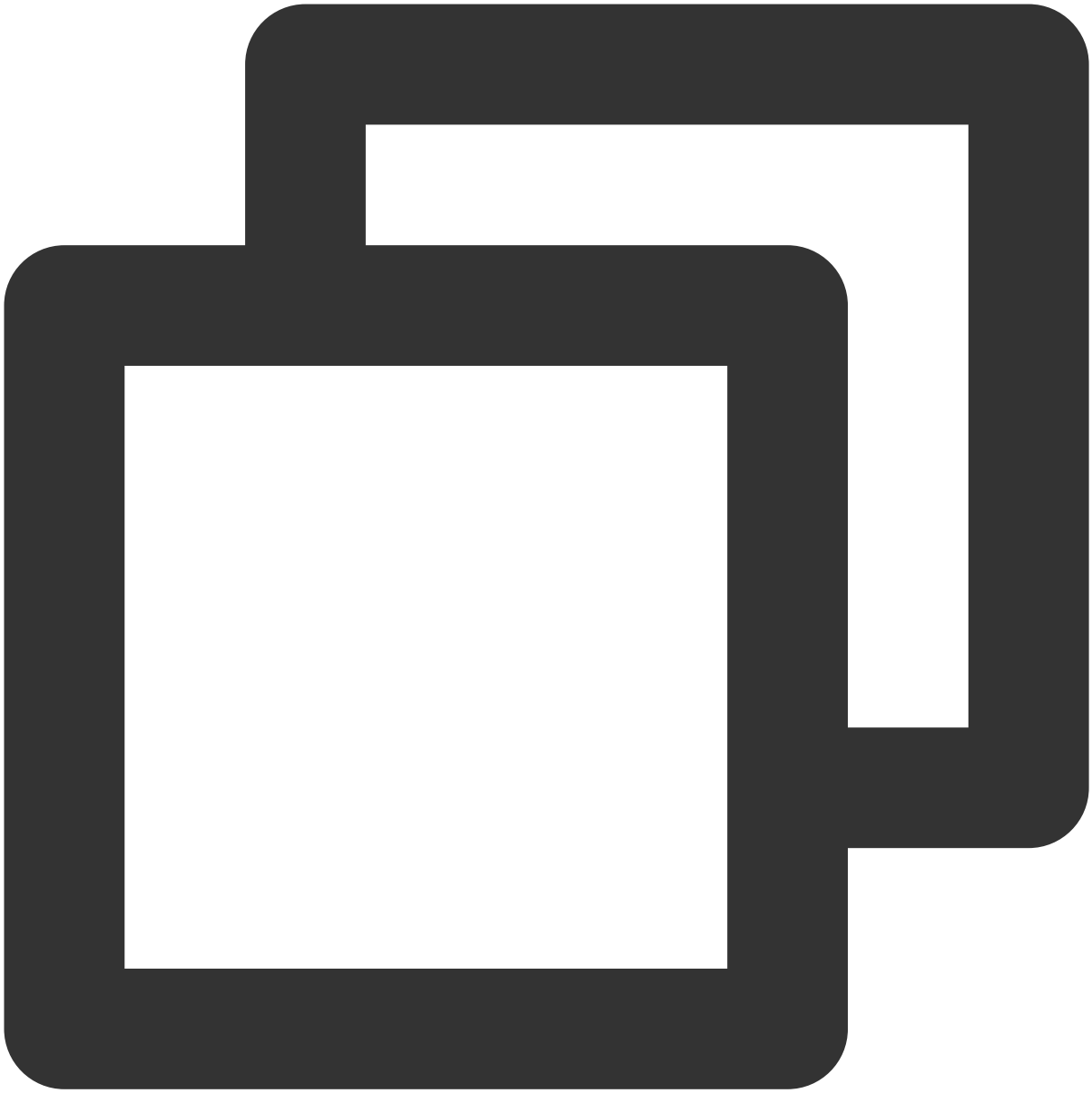
应答 JSON 数据




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "RecordVersion": 1,
 "ReturnValues": "Send to tcaplus by calvinshao",
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70,
```

```
    80,
    90,
    100
   ],
   "pay": {
    "pay_times": 3,
    "total_money": 12000,
    "pay_id": 5,
    "auth": {
     "pay_keys": "adqwacsasafasda",
     "update_time": 1528018372
    }
   },
   "region": 101,
   "uin": 100,
   "is_available": false,
   "gamesvrid": 4099,
   "logintime": 404
  }
}
```

Tcaplus.AddRecord



```
POST /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records
```

通过指定一条记录的 key 信息插入一条记录。如果记录存在返回错误。

AddRecord 操作支持 resultflag 设置以下取值：

`0` 设置为0，应答中仅包含请求成功或失败

`1` 设置为1，应答中包含与请求一致的值

`2` 设置为2，应答中包含被修改的数据的所有字段最新值

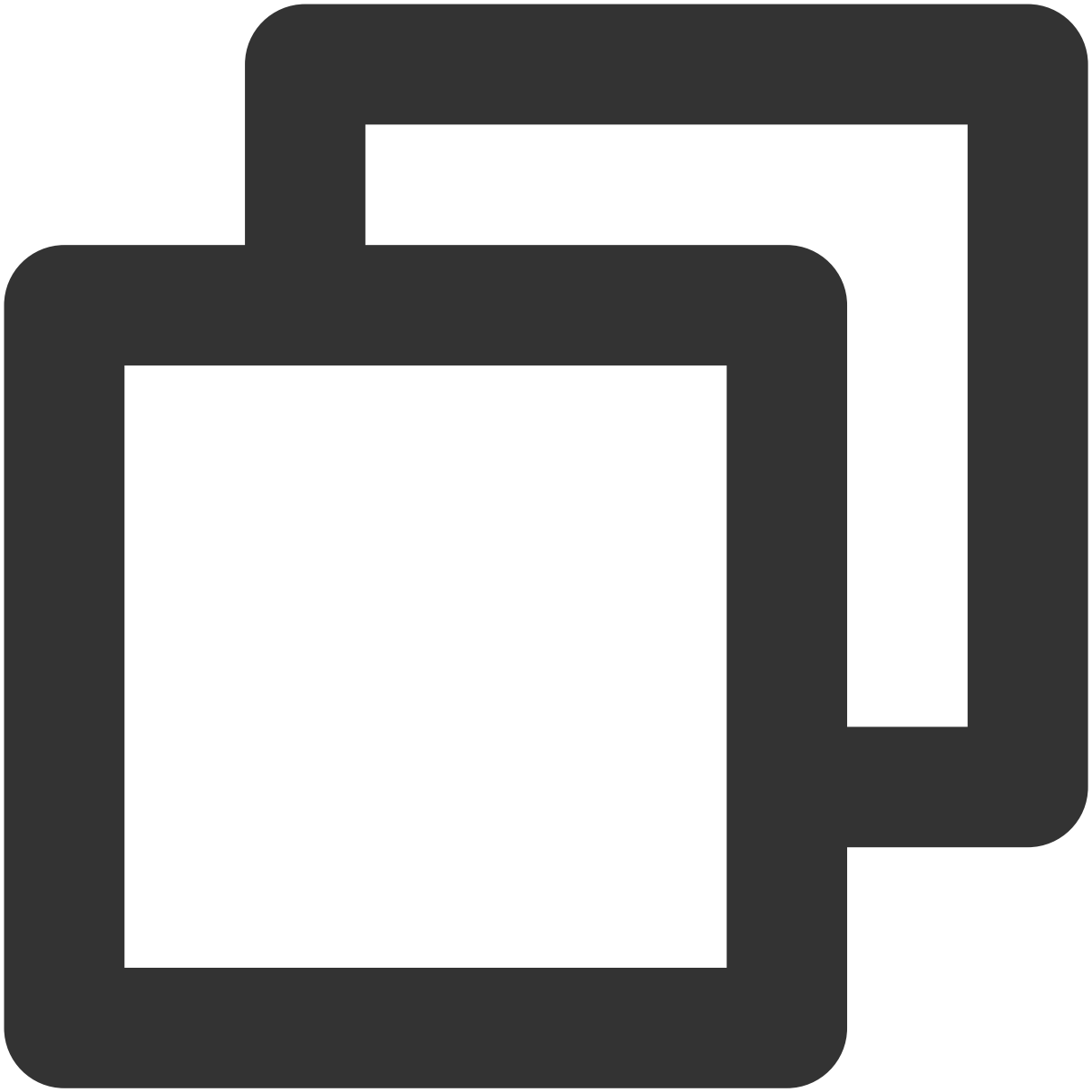
`3` 设置为3，应答中包含记录被修改前的值

名称	类型	取值

x-tcaplus-target	String	Tcaplus.SetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-result-flag	Int	1
x-tcaplus-idl-type	String	protobuf

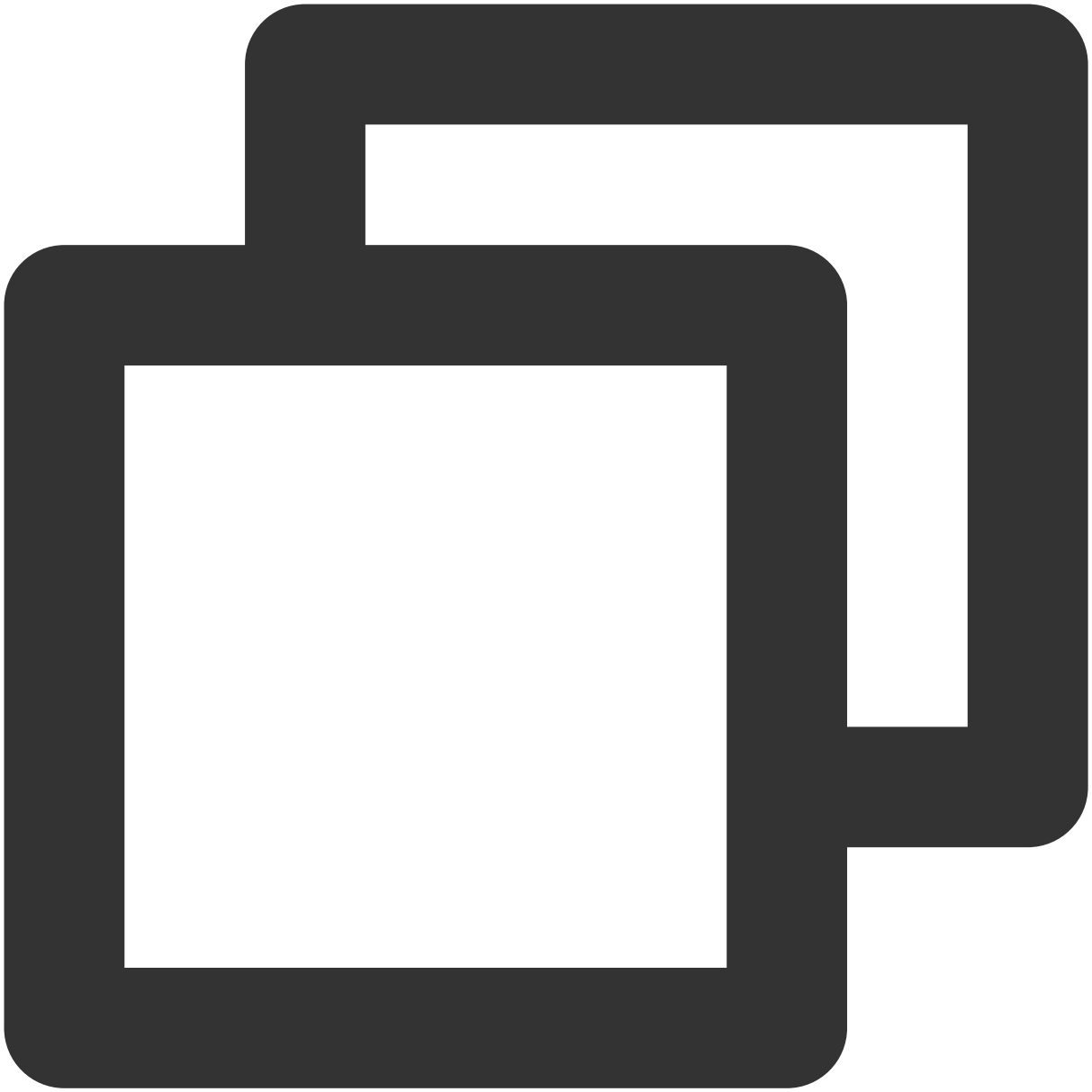
示例：

URL：



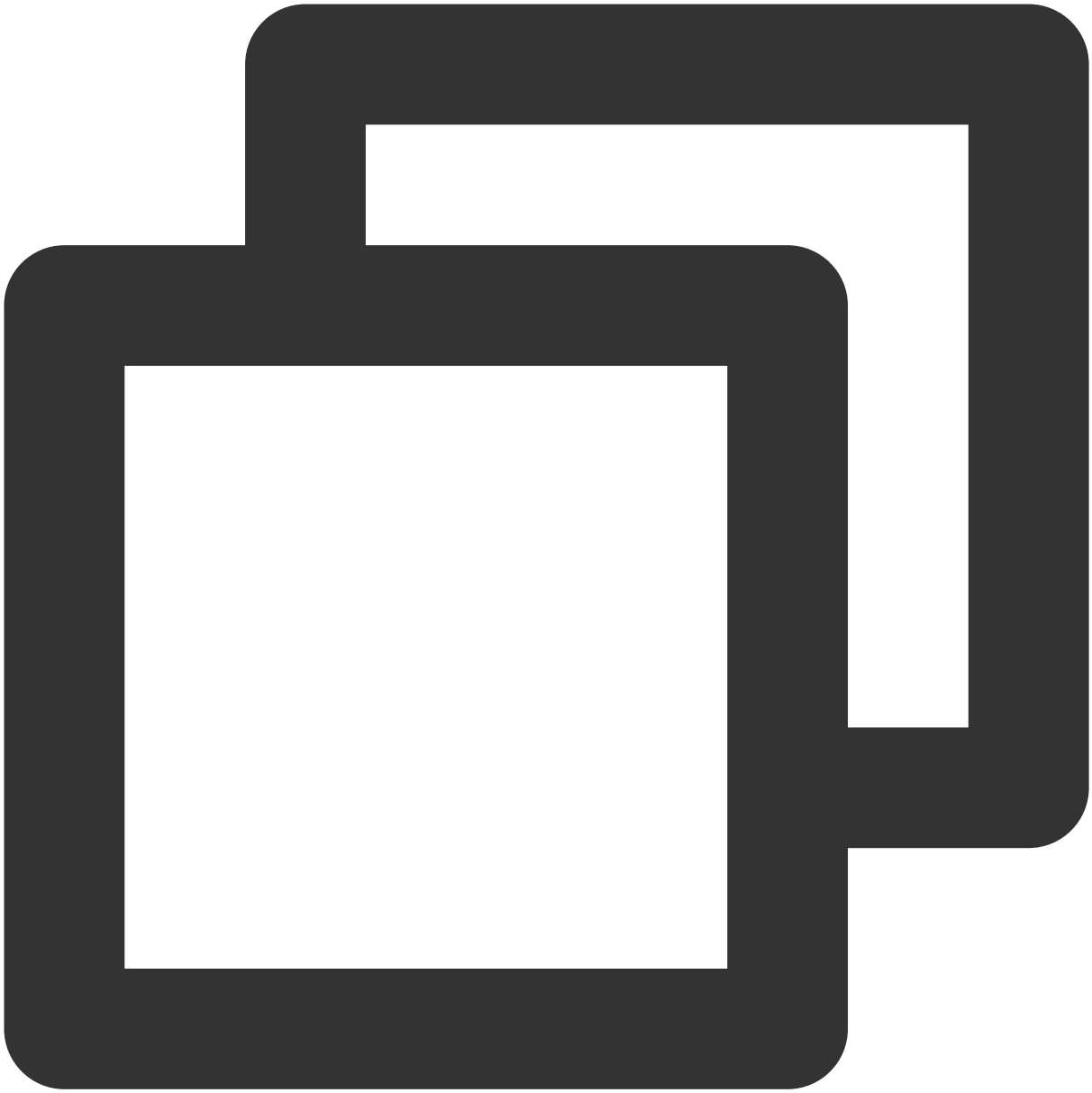
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.AddRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-result-flag:1",
 "x-tcaplus-idl-type:protobuf"
]
```

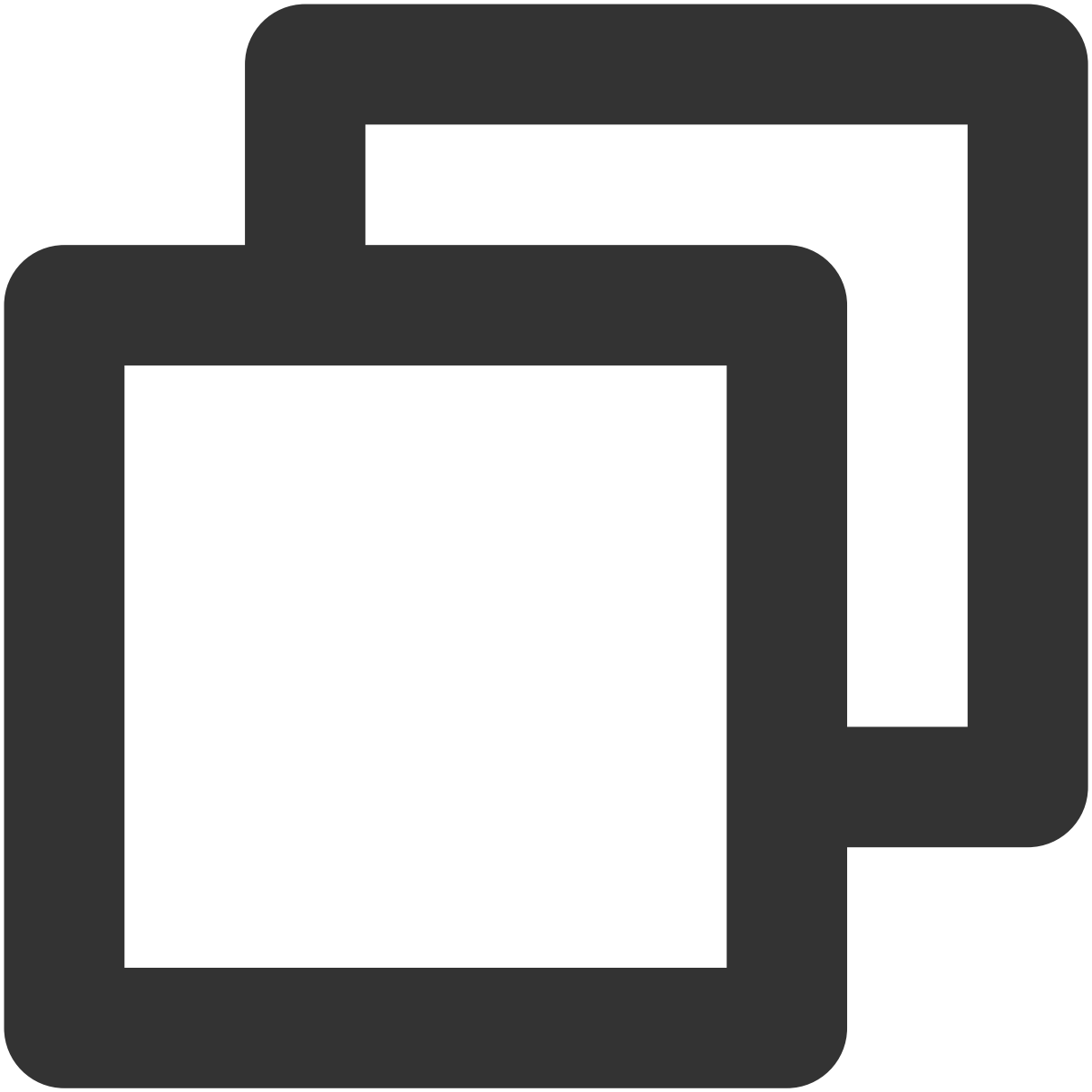
请求 JSON 数据：



```
{
 "ReturnValues": "Send to tcaplus by calvinshao",
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70
  ],
  "pay": {
   "pay_times": 2,
```

```
    "total_money": 10000,
    "pay_id": 5,
    "auth": {
     "pay_keys": "adqwacsasafasda",
     "update_time": 1528018372
    }
   },
   "region": 101,
   "uin": 100,
   "is_available": true,
   "gamesvrid": 4099,
   "logintime": 404
  }
 }
```

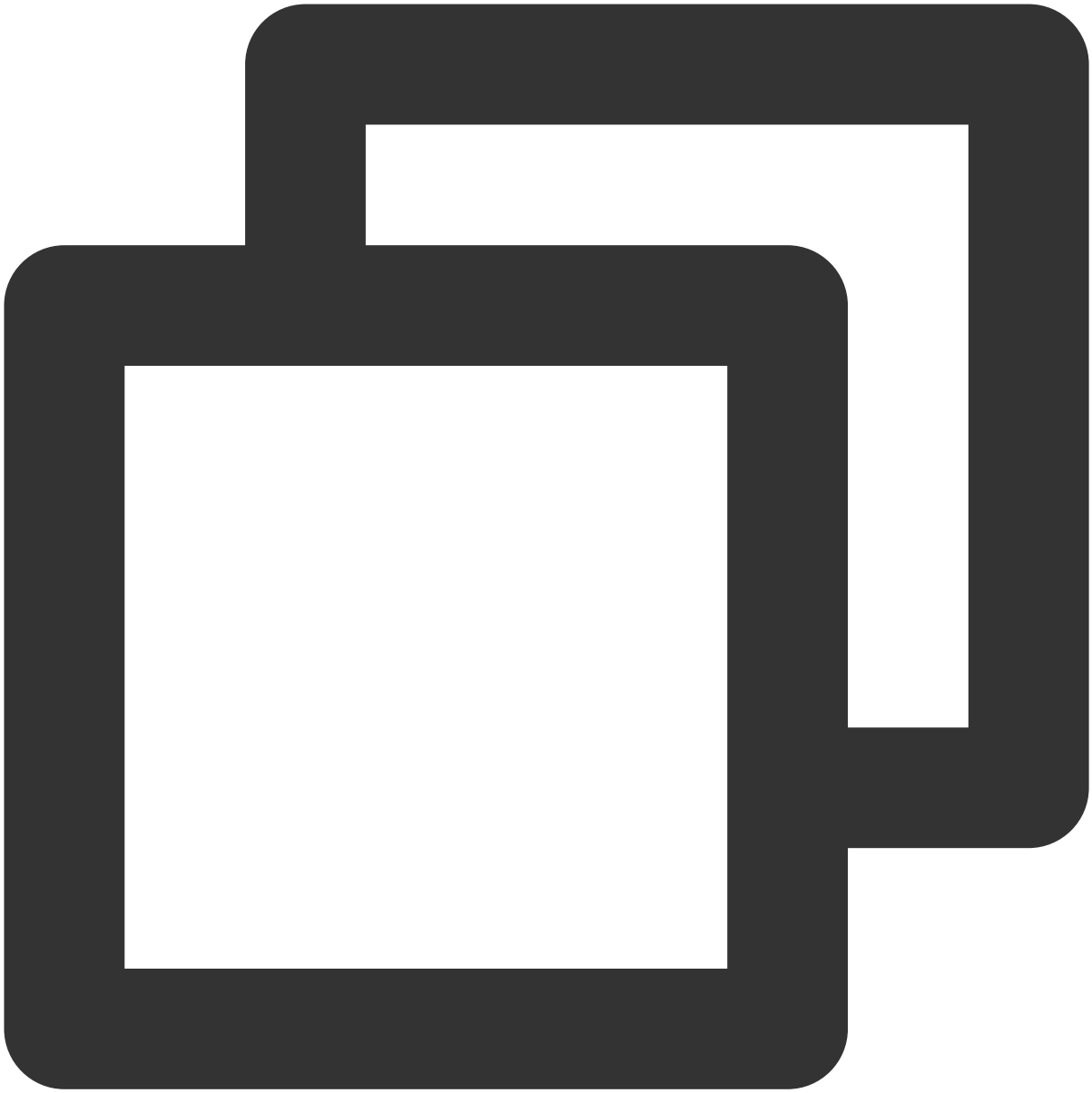
应答 JSON 数据：




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "RecordVersion": 1,
 "ReturnValues": "Send to tcaplus by calvinshao",
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70
```

```
    ],
    "pay": {
     "pay_times": 2,
     "total_money": 10000,
     "pay_id": 5,
     "auth": {
      "pay_keys": "adqwacsasafasda",
      "update_time": 1528018372
     }
    },
    "region": 101,
    "uin": 100,
    "is_available": true,
    "gamesvrid": 4099,
    "logintime": 404
   }
}
```

Tcaplus.DeleteRecord



```
DELETE /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records
```

通过指定一条记录的 key 信息删除此记录，如果数据不存在则返回错误。

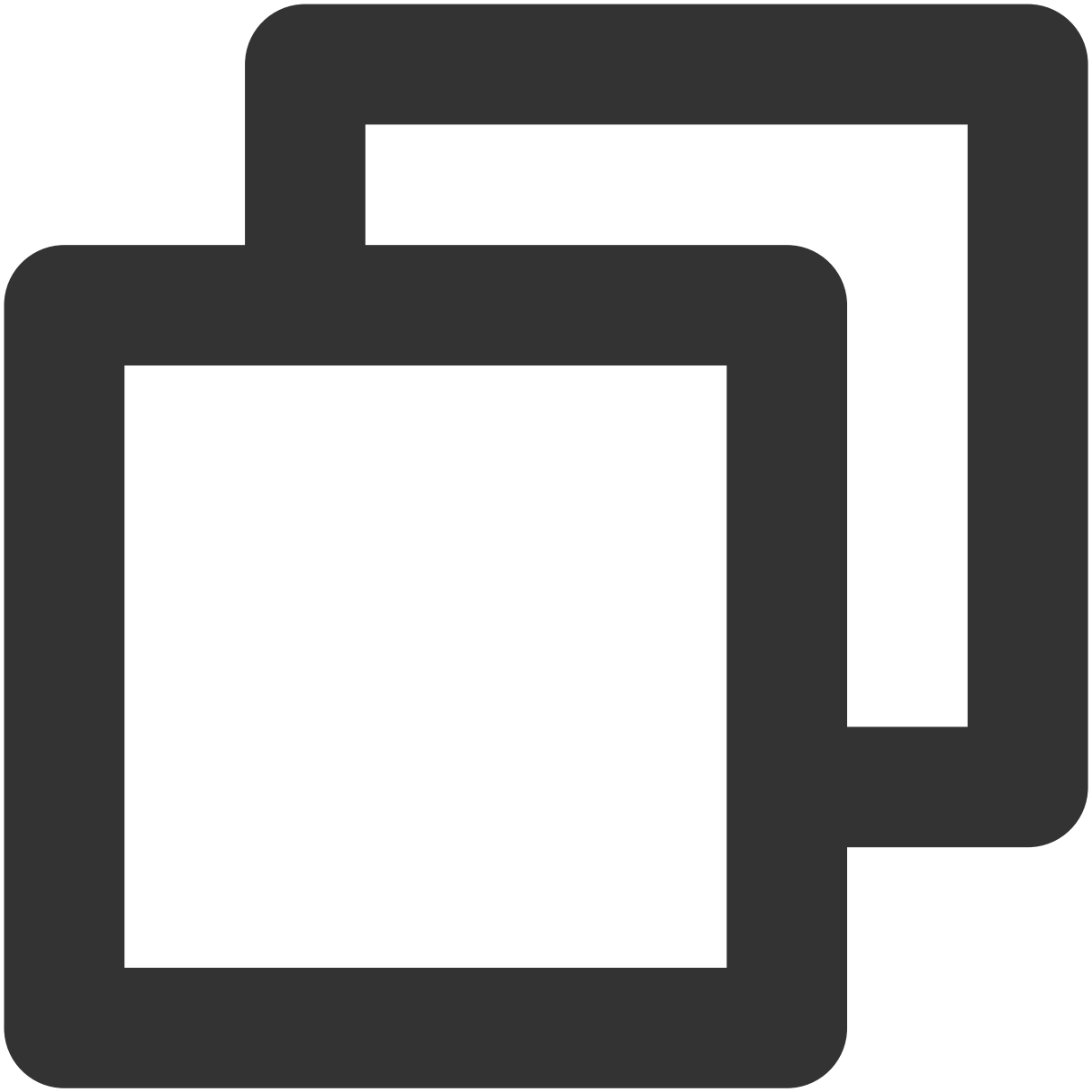
DeleteRecord 操作支持 resultflag 设置以下取值：

- `0` 设置为0，应答中仅包含请求成功或失败
- `1` 设置为1，应答中包含与请求一致的值
- `2` 设置为2，应答中包含被修改的数据的所有字段最新值
- `3` 设置为3，应答中包含记录被修改前的值

名称	类型	取值
x-tcaplus-target	String	Tcaplus.SetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-result-flag	Int	1
x-tcaplus-idl-type	String	protobuf

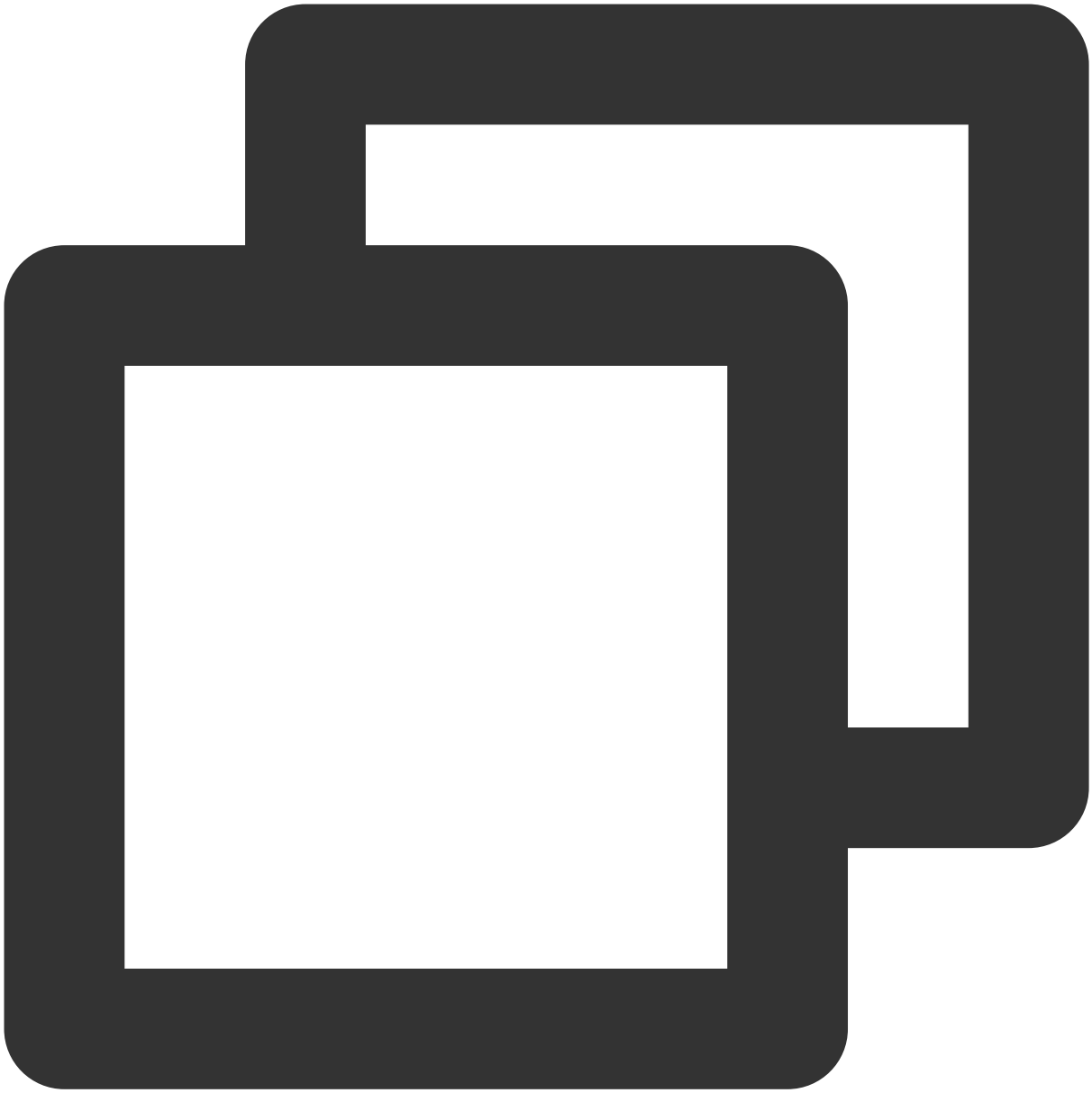
示例：

URI：



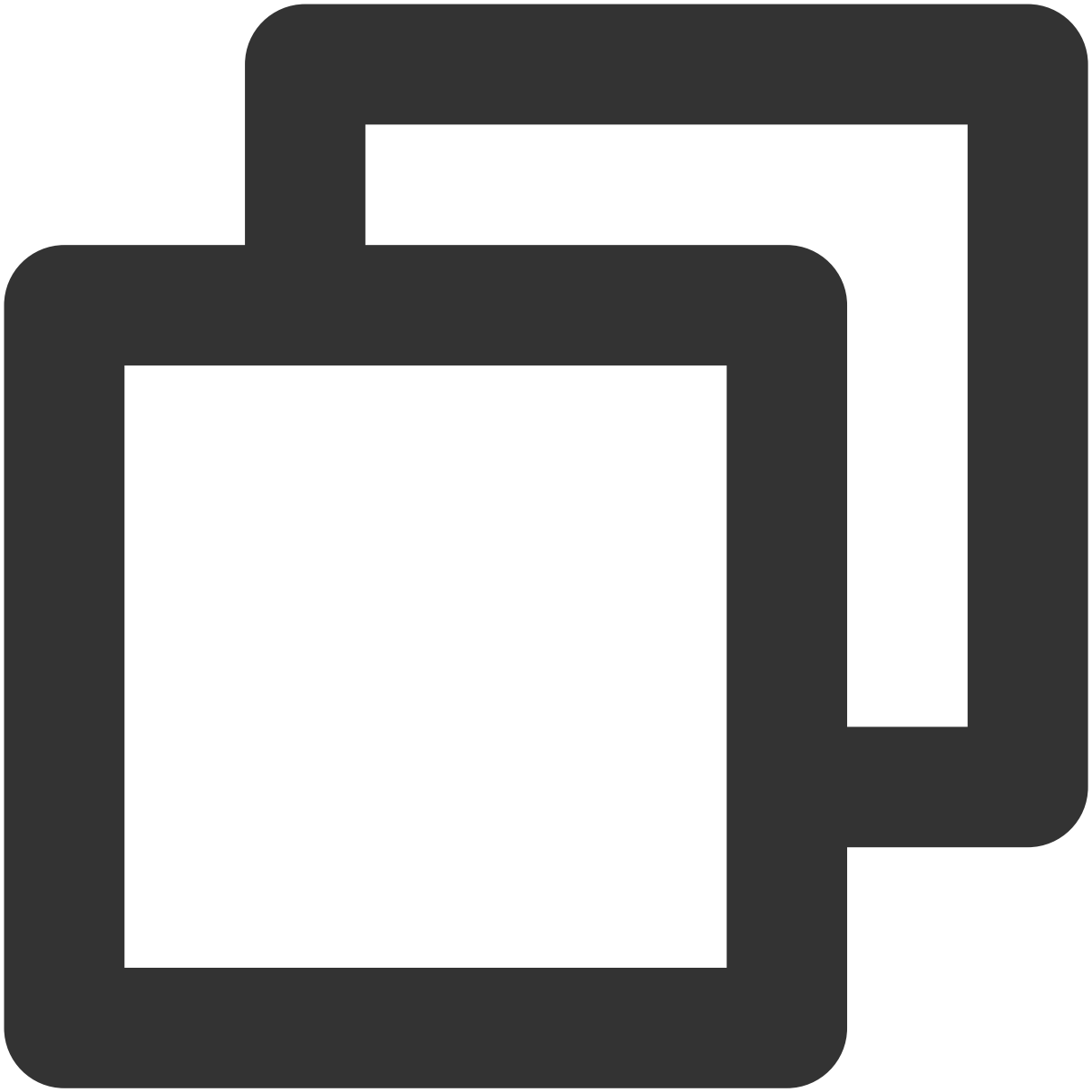
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records
```

请求 HTTP 头：



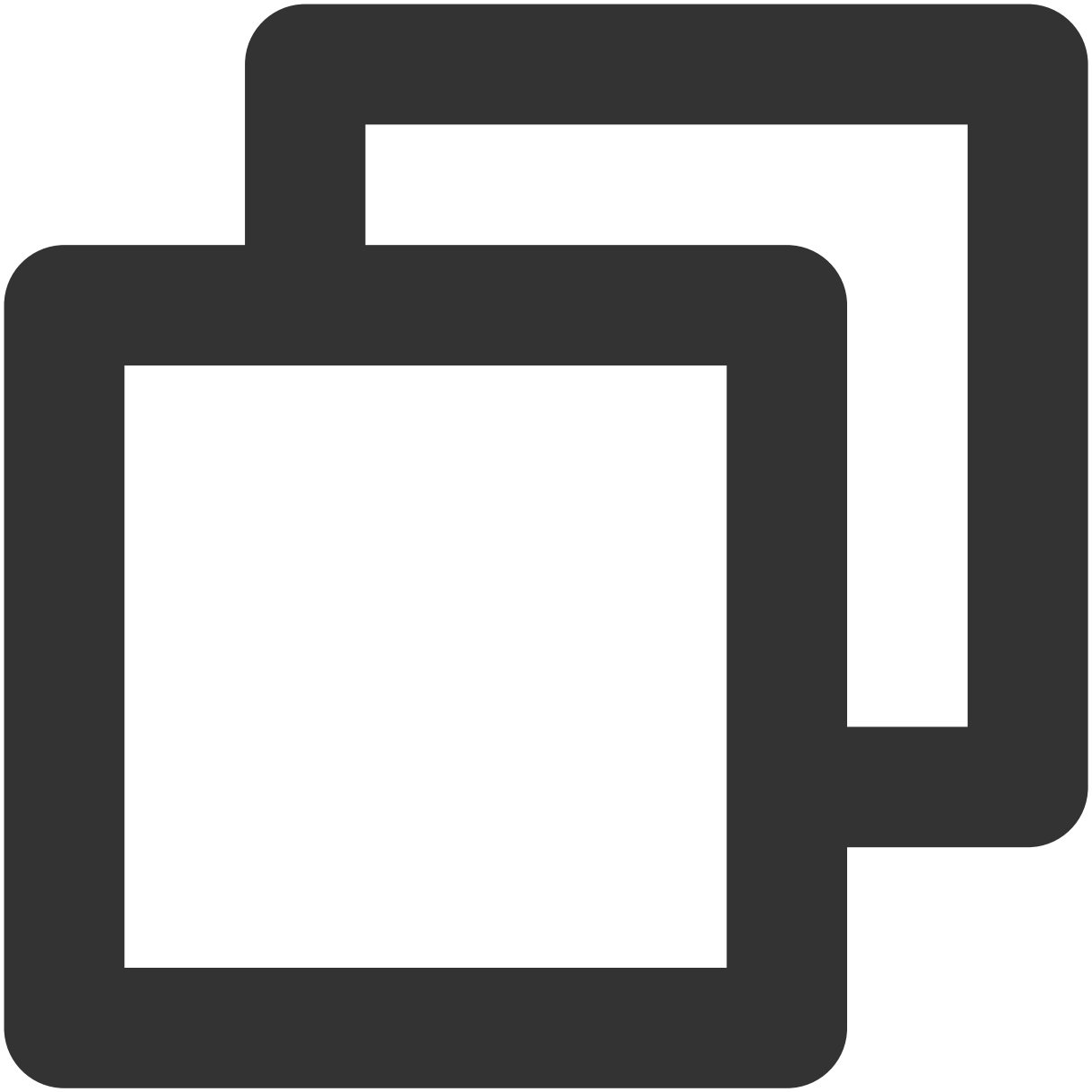
```
[
 "x-tcaplus-target:Tcaplus.DeleteRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-result-flag:1",
 "x-tcaplus-idl-type:protobuf"
]
```

请求 JSON 数据：



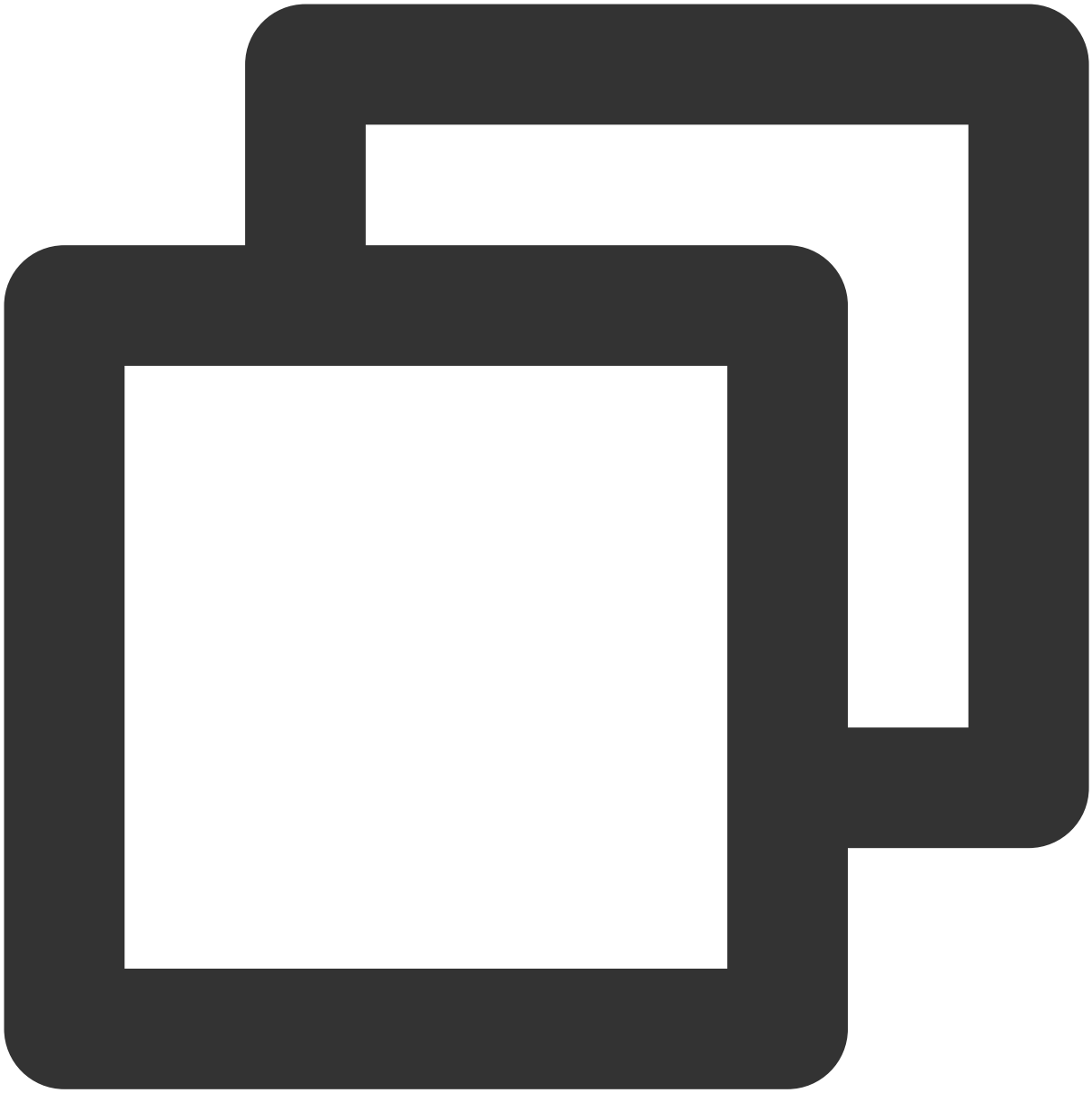
```
{
 "ReturnValues": "Send to tcaplus by calvinshao",
 "Record": {
  "region": 101,
  "name": "calvinshao",
  "uin": 100
 }
}
```

应答 JSON 数据：



```
{
 "ErrorCode": 0,
 "Record": {},
 "RecordVersion": -1,
 "ReturnValues": "Send to tcaplus by calvinshao"
}
```

Tcaplus.FieldGetRecord



```
GET /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records?keys={JSONKey
```

从一个 Tcaplus pb 表中通过指定一条记录的 key 信息查询此记录。本操作只查询和传输用户通过 select 变量指定的字段的值，这将减少网络传输流量，这是与 GetRecord 操作最大的不同之处。如果数据记录不存在，将会返回错误。

必须在 URI 中指定 `keys` 和 `select` 变量。keys 指定所有主键的值，select 指定需要显示的 value 字段的名称。并且用户可以通过点分路径的方式指定嵌套结构中的字段，例如：“pay.total_money”。

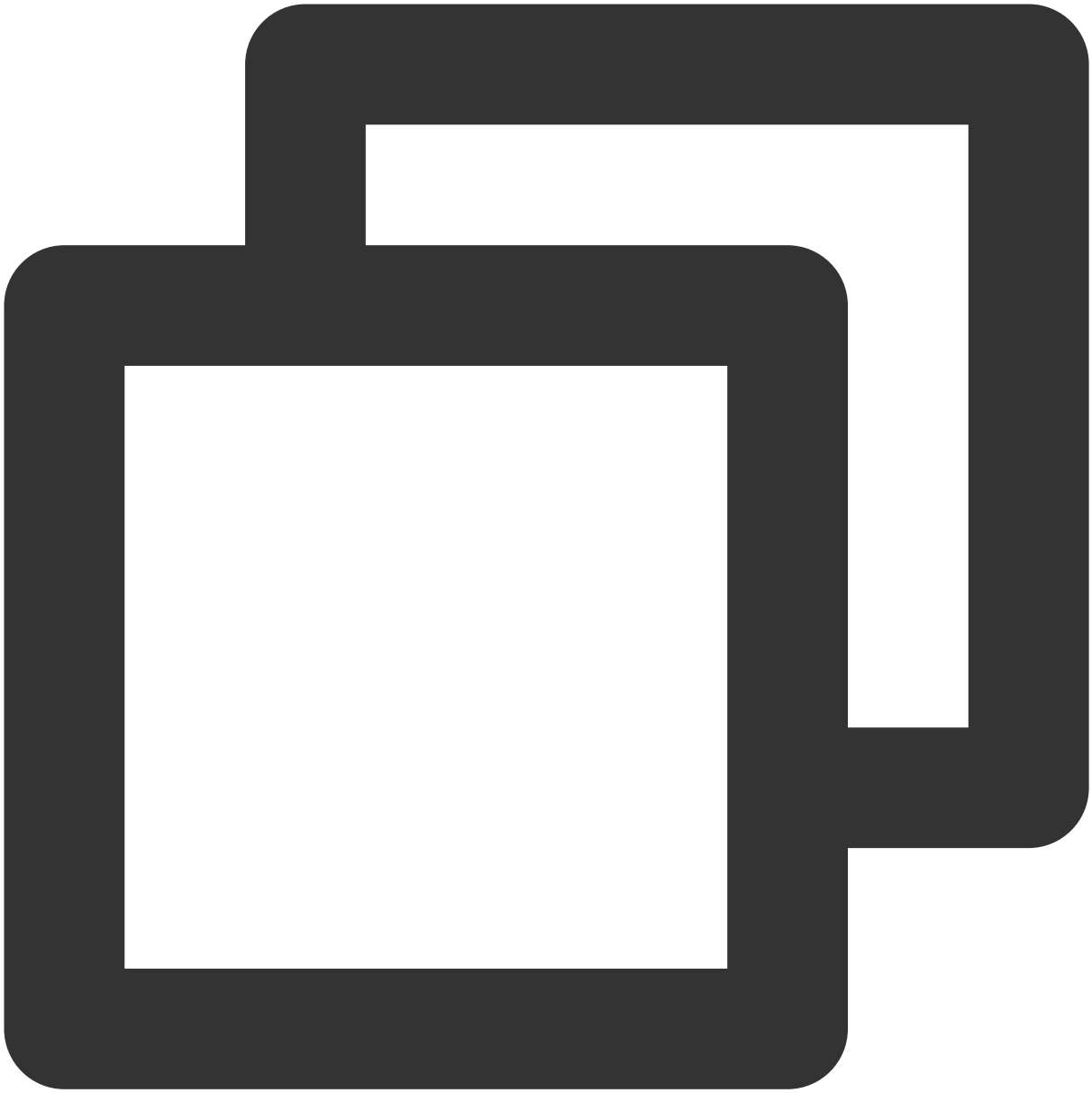
注意：

请求的变量必须通过 urlencode 编码，请将url中的空格编码为"%20"而不是"+"。

名称	类型	取值
x-tcaplus-target	String	Tcaplus.FieldGetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-idl-type	String	protobuf

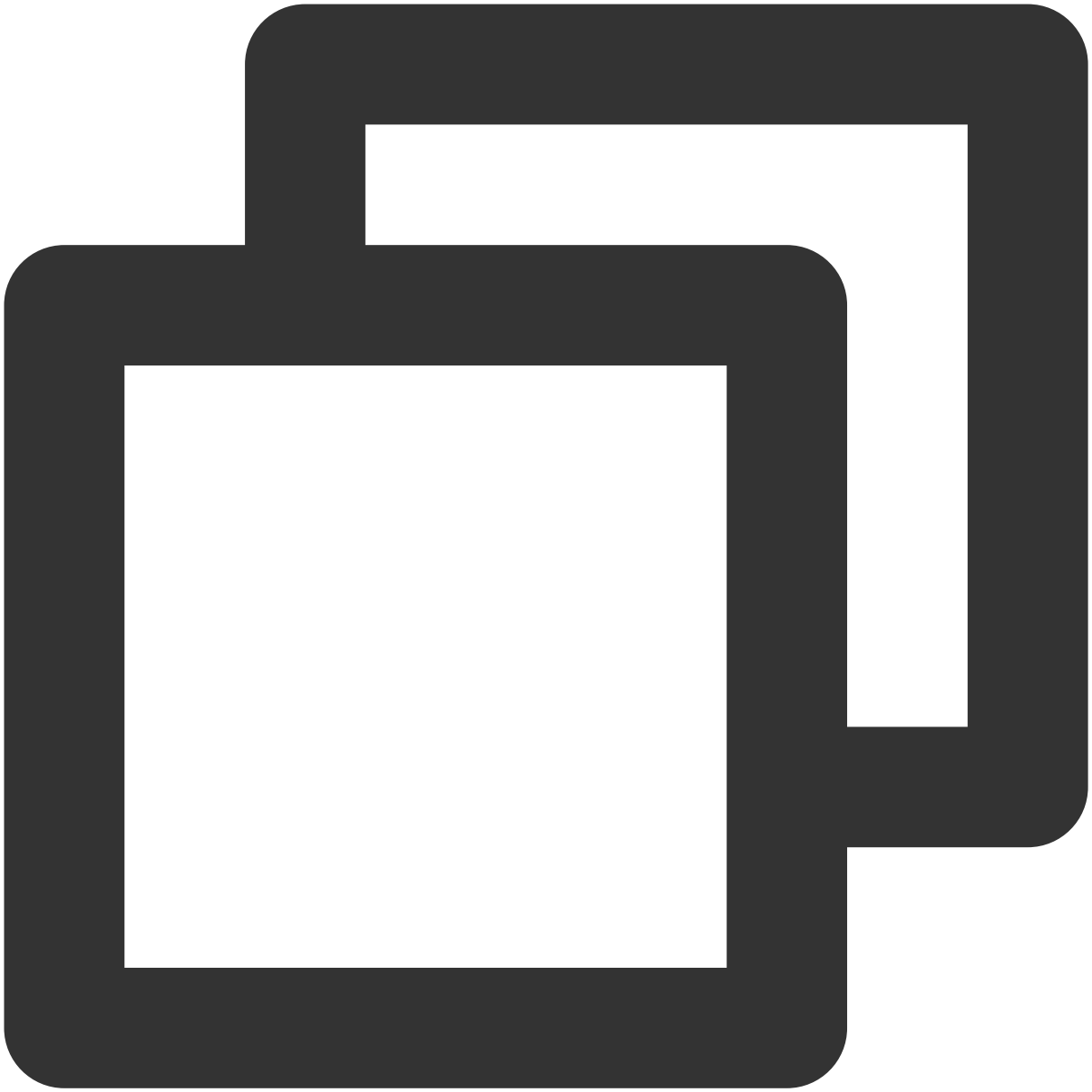
示例：

URL 未 UrlEncode 结果:



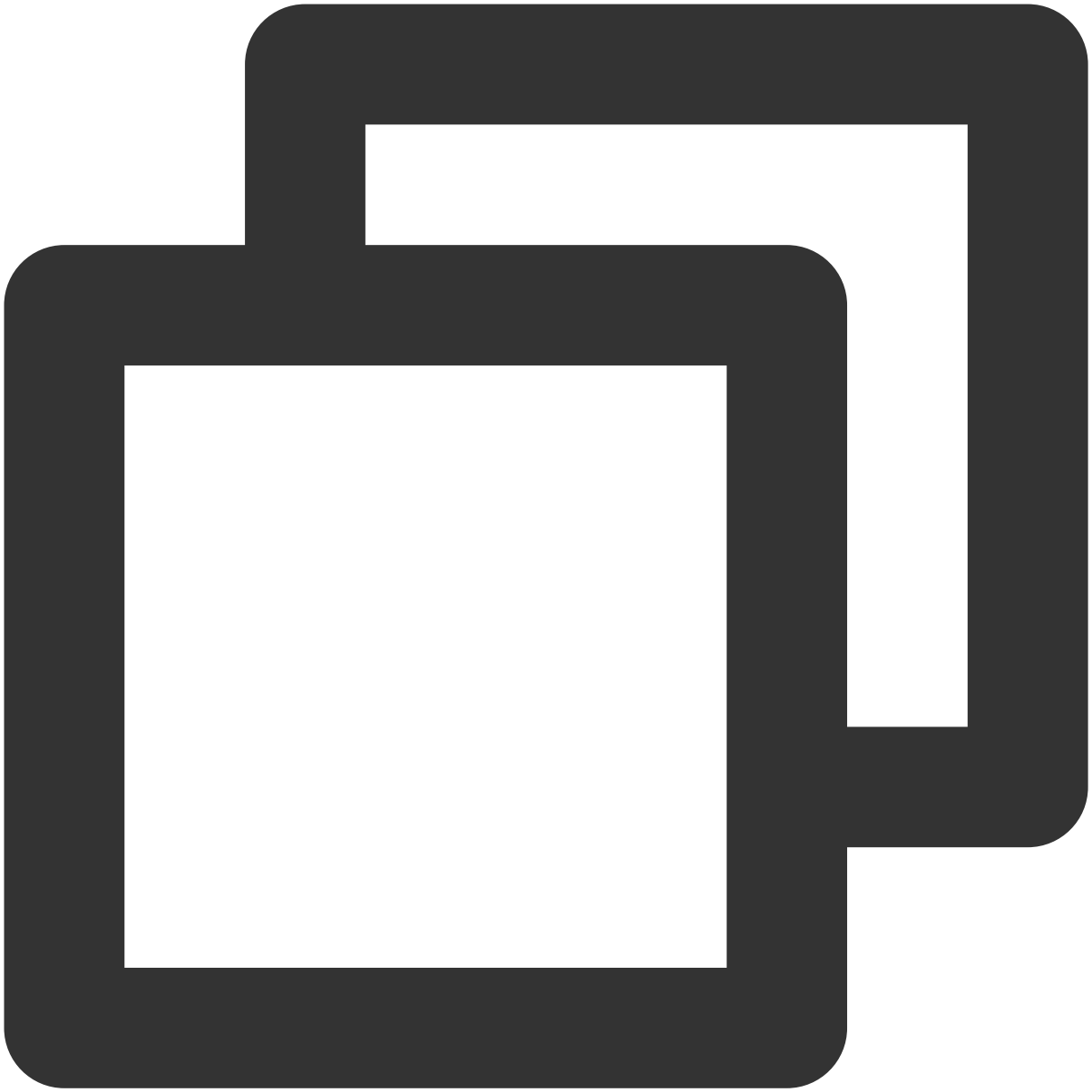
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records?keys={'region':
```

URL UrlEncode 结果：



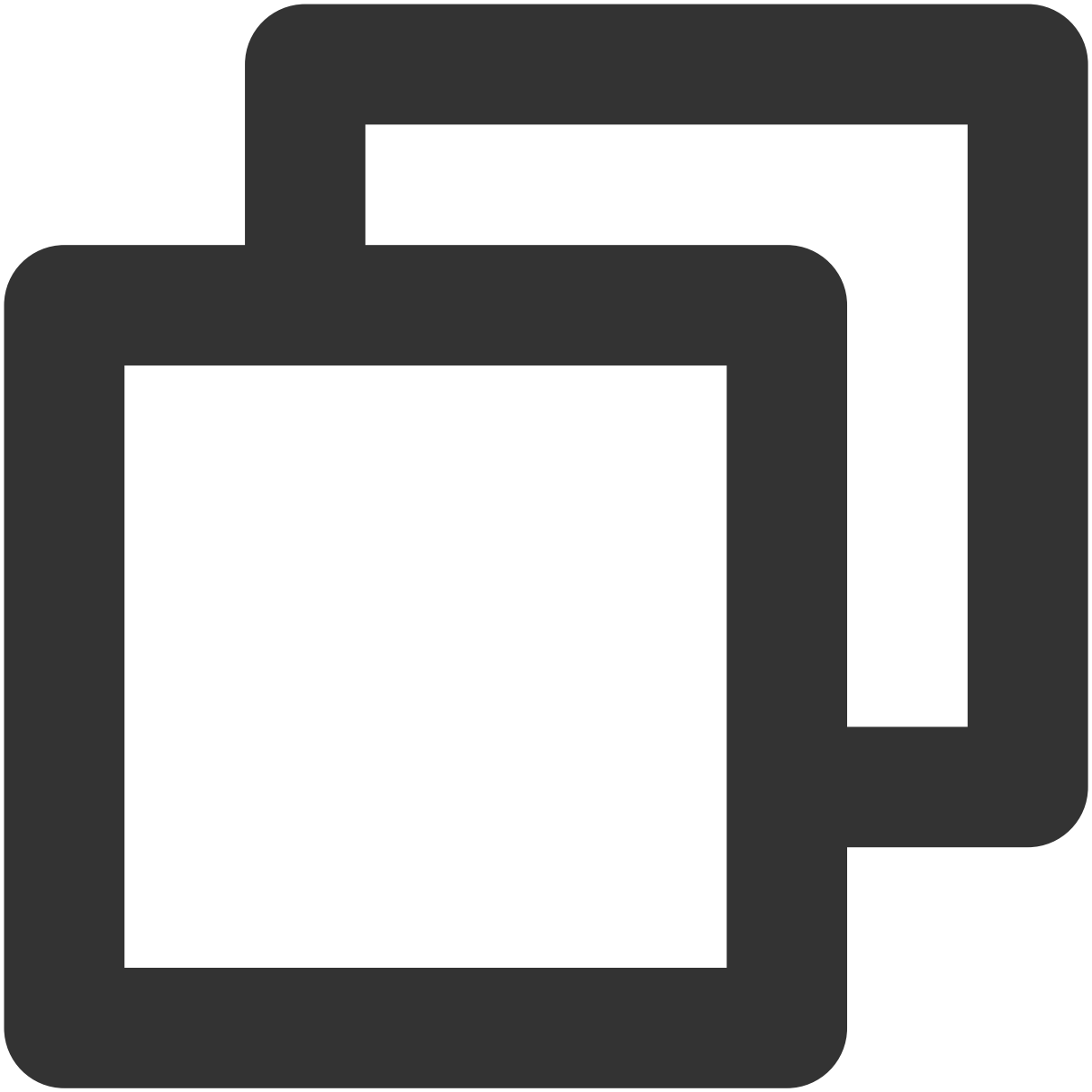
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records?keys=%7B%22regio
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.FieldGetRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-idl-type:protobuf"
]
```

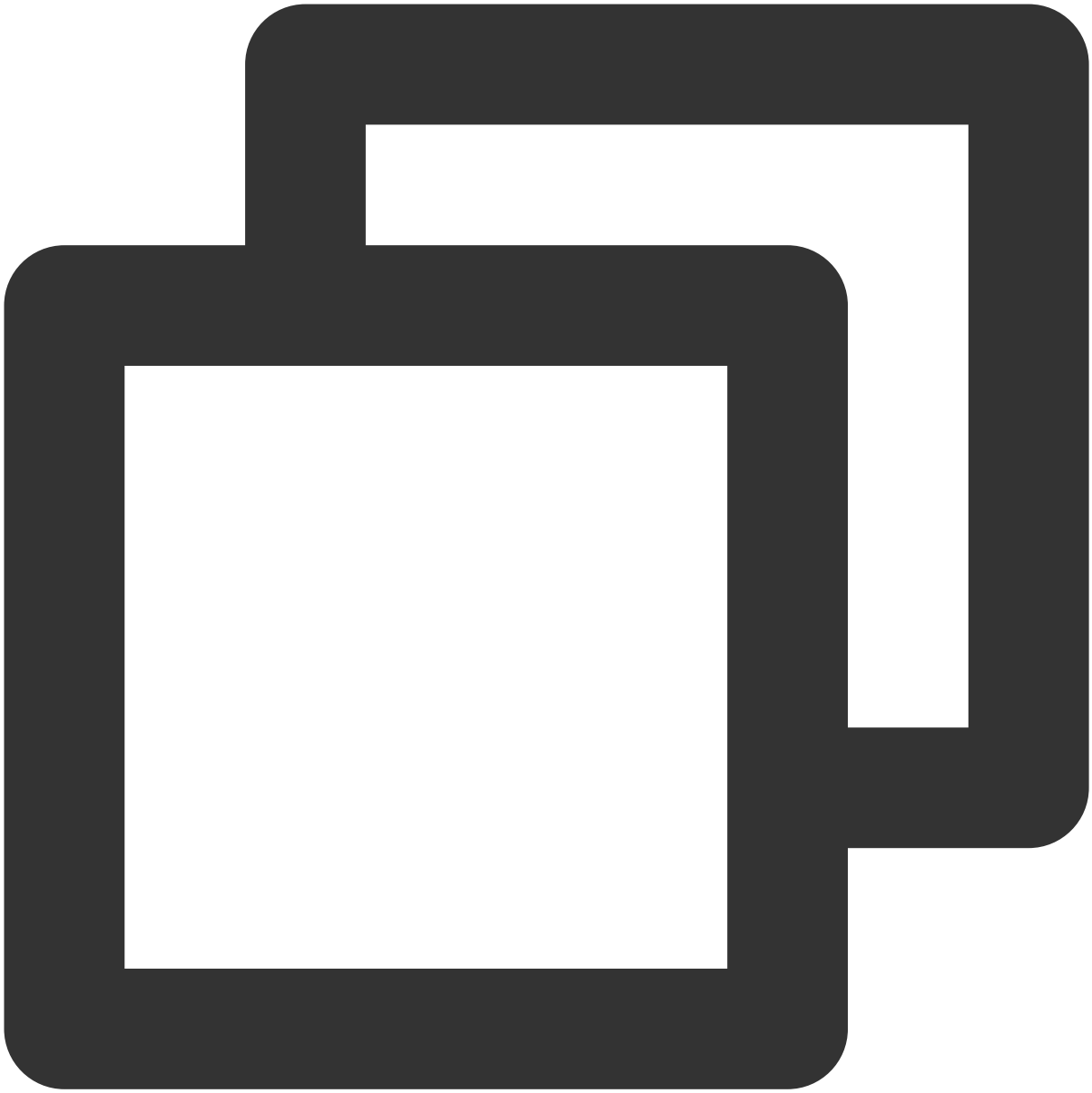
应答 JSON 数据：




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "RecordVersion": 1,
 "Record": {
  "name": "calvinshao",
  "lockid": [
   50,
   60,
   70
  ],
```

```
   "pay": {
    "total_money": 10000,
    "auth": {
     "pay_keys": "adqwacsasafasda"
    }
   },
   "region": 101,
   "uin": 100,
   "gamesvrid": 4099
  }
 }
```

Tcaplus.FieldSetRecord



```
PUT /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records
```

通过指定一条记录的 key 信息修改此记录，与 SetRecord 操作不同的是此操作只传输并设置指定字段的值，并不传输所有字段。这将减轻网络流量。如果数据记录存在，将执行更新操作，否则将会返回错误。

FieldSetRecord 操作支持 resultflag 设置以下取值：

`0` 设置为0，应答中仅包含请求成功或失败

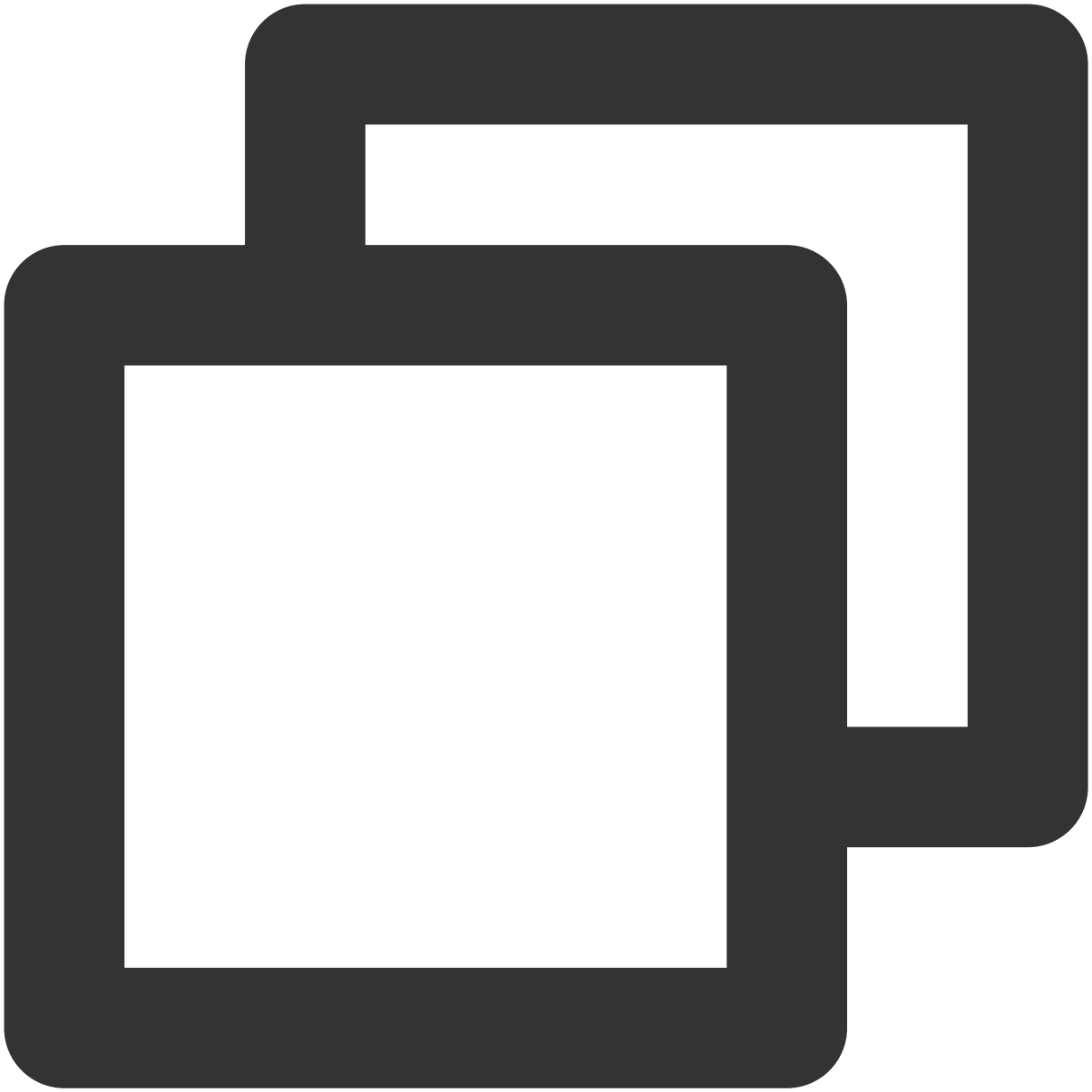
`1` 设置为1，应答中包含与请求一致的值

名称	类型	取值

x-tcaplus-target	String	Tcaplus.SetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-result-flag	Int	2
x-tcaplus-data-version-check	Int	2
x-tcaplus-data-version	Int	-1
x-tcaplus-idl-type	String	protobuf

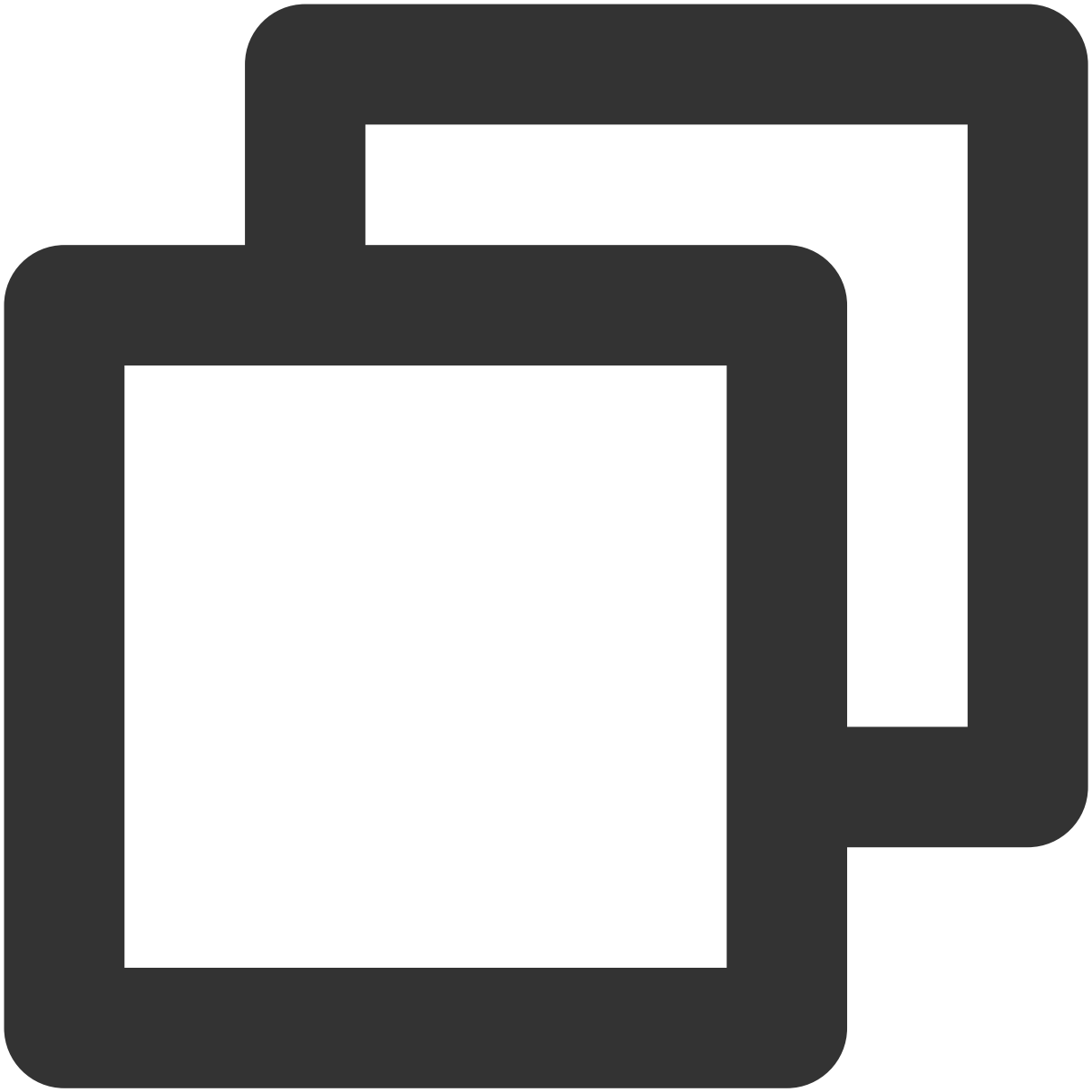
示例：

URL：



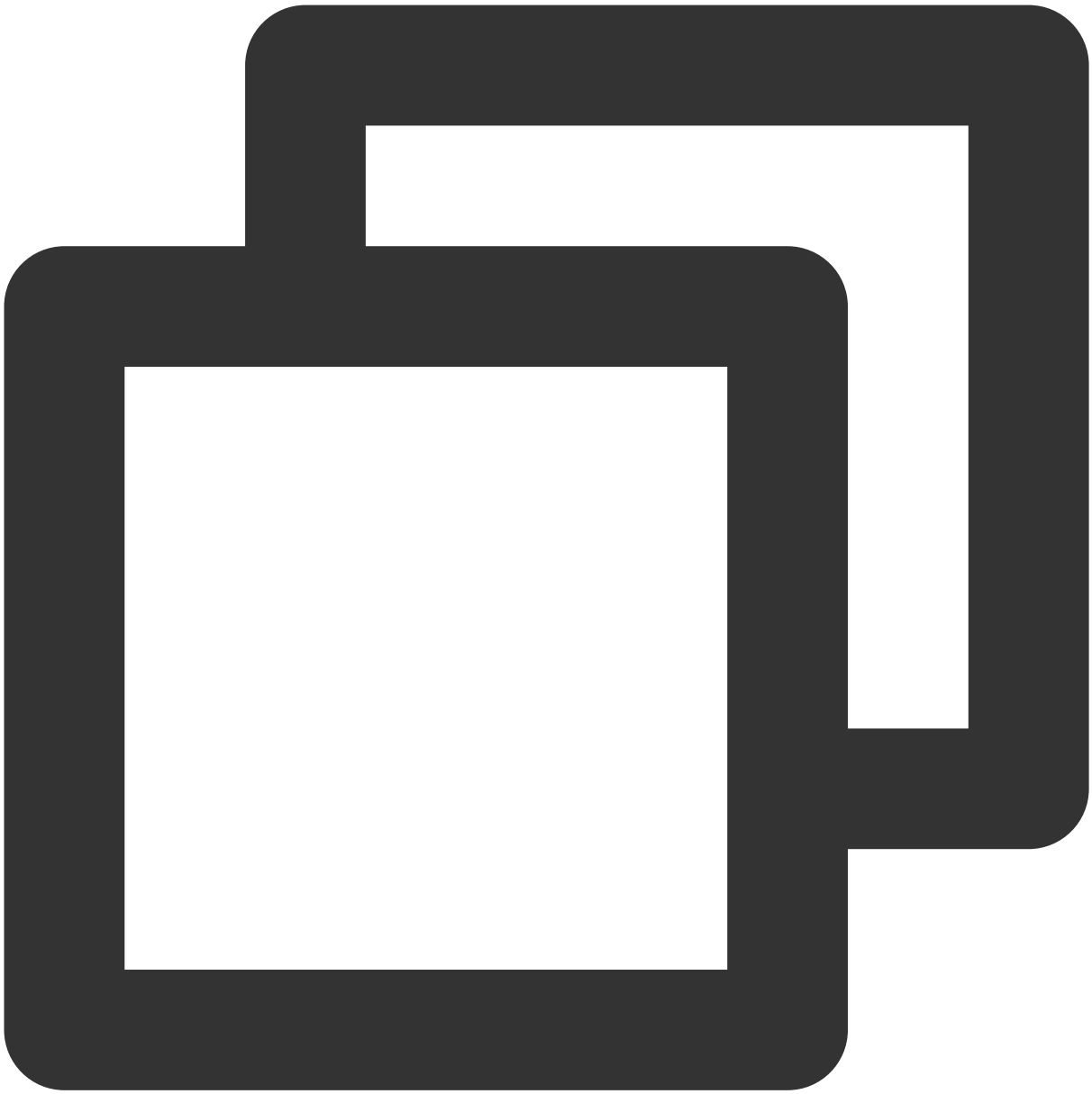
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.FieldSetRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-result-flag:1",
 "x-tcaplus-data-version-check:1",
 "x-tcaplus-data-version:-1",
 "x-tcaplus-idl-type:Protobuf"
]
```

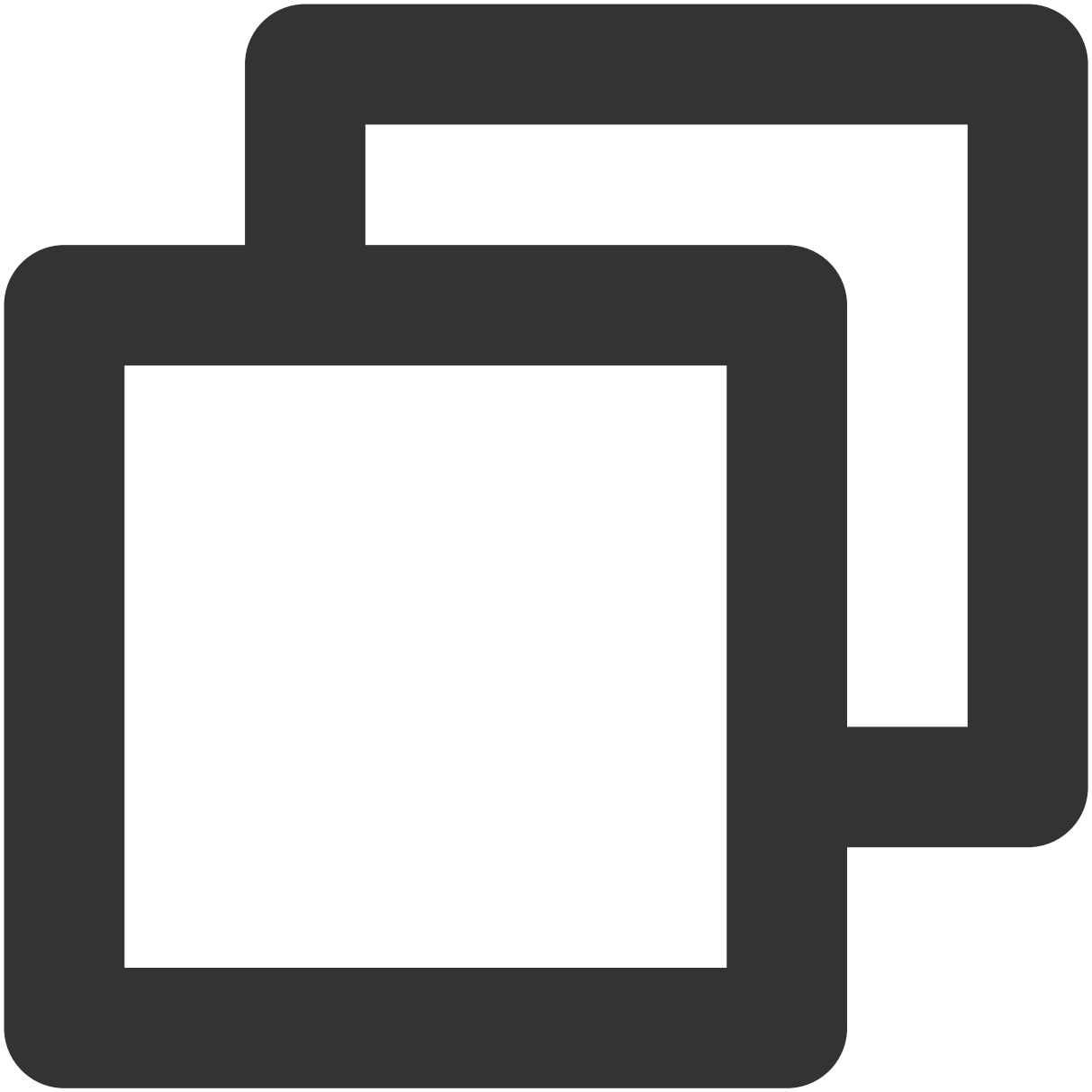
请求 JSON 数据：



```
{
 "ReturnValues": "aaaaaaaaaa",
 "FieldPath": [ // 显式指定需要更新的字段路径
    "gamesvrid",
    "logintime",
    "pay.total_money",
    "pay.auth.pay_keys"
 ],
 "Record": { // 设置需要更新的字段新值
```

```
   "name": "calvinshao",
   "pay": {
    "total_money": 17190,
    "auth": {
     "pay_keys": "bingo"
    }
   },
   "region": 101,
   "uin": 100,
   "gamesvrid": 1719,
   "logintime": 1719
  }
 }
```

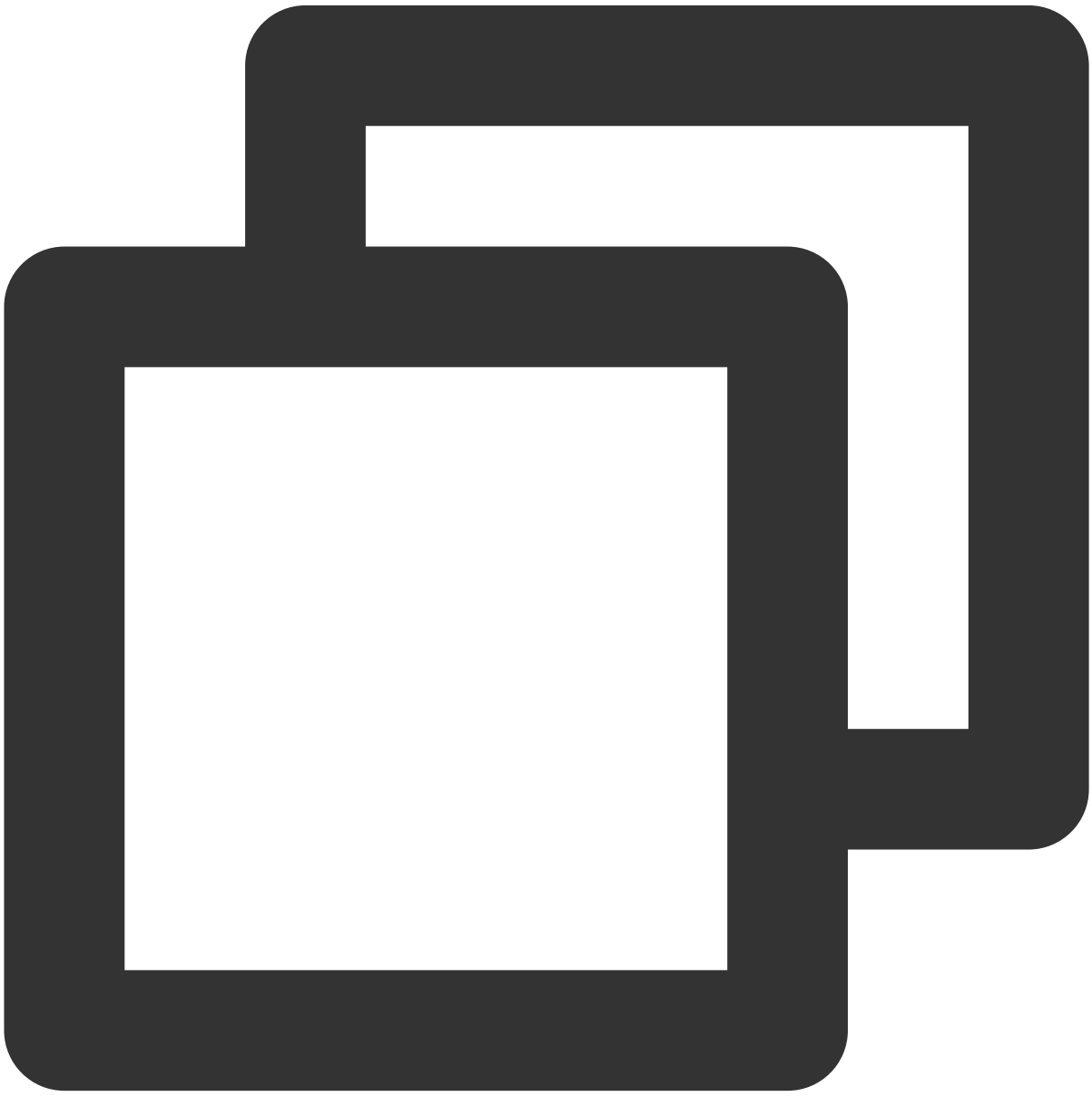
应答 JSON 数据：




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "RecordVersion": 3,
 "ReturnValues": "aaaaaaaaaa",
 "Record": {
  "name": "calvinshao",
  "pay": {
   "total_money": 17190,
   "auth": {
    "pay_keys": "bingo"
```

```
    }
   },
   "region": 101,
   "uin": 100,
   "gamesvrid": 1719,
   "logintime": 1719
  }
 }
```

Tcaplus.FieldIncRecord



```
PUT /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records
```

通过指定一条记录的 key 信息对指定的字段进行自增操作，此命令字仅支持 `int32`， `int64`， `uint32` 和 `uint64` 类型字段。特性与 FieldSetRecord 类似。

SetRecord 操作支持 resultflag 设置以下取值：

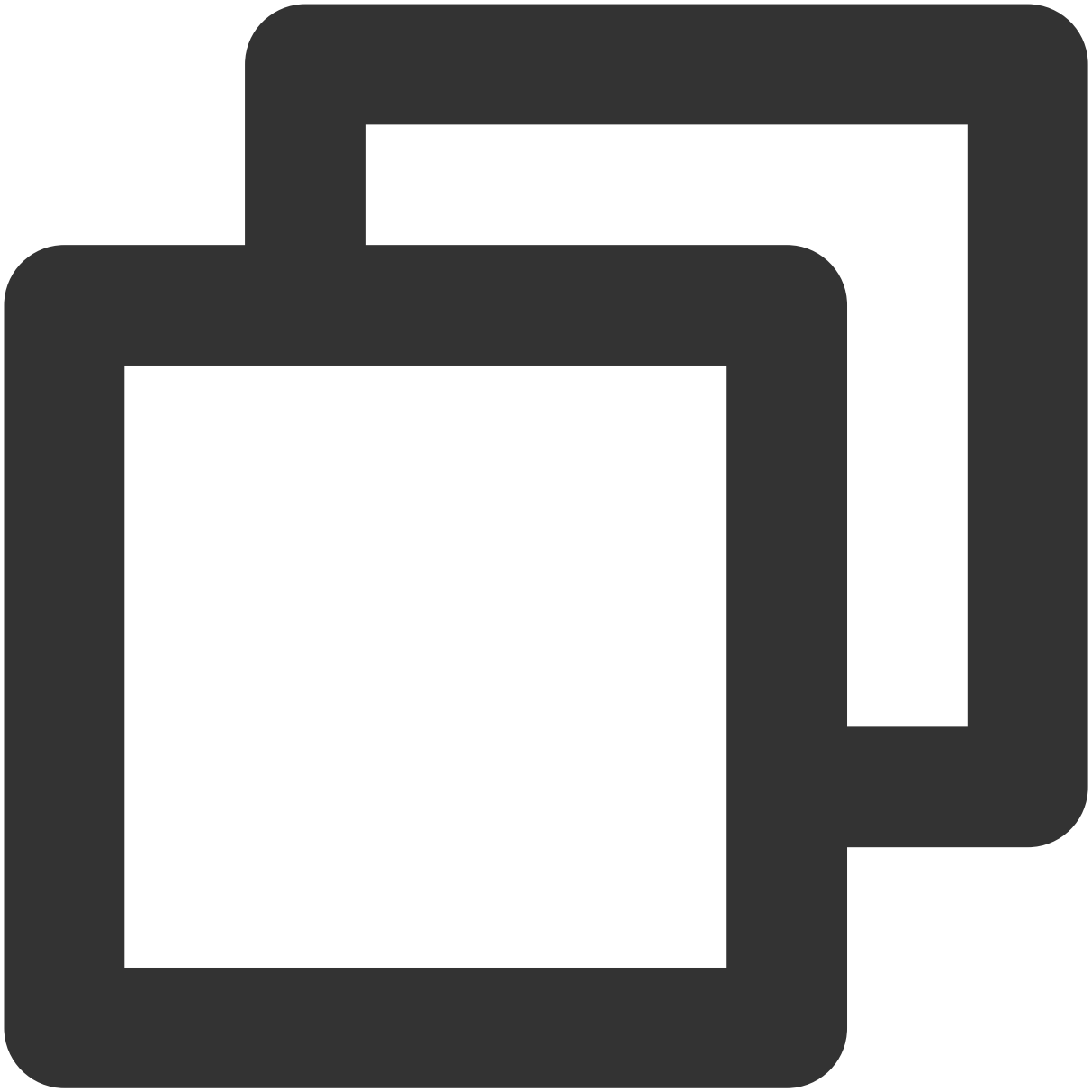
`0` 设置为0，应答中仅包含请求成功或失败

`1` 设置为1，应答中包含指定字段修改后的值

名称	类型	取值
x-tcaplus-target	String	Tcaplus.SetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-result-flag	Int	2
x-tcaplus-data-version-check	Int	2
x-tcaplus-data-version	Int	-1
x-tcaplus-idl-type	String	protobuf

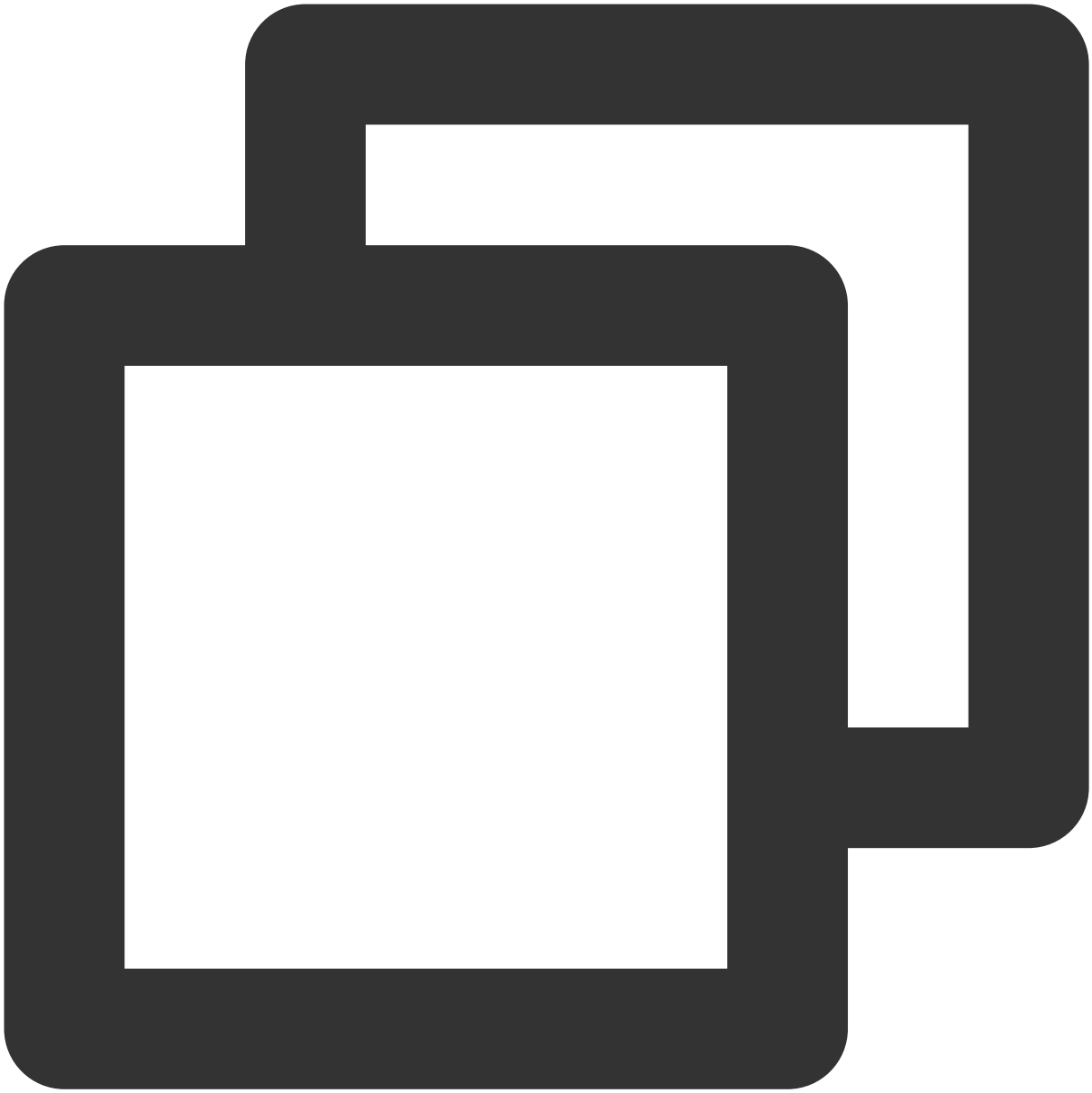
示例：

URL：



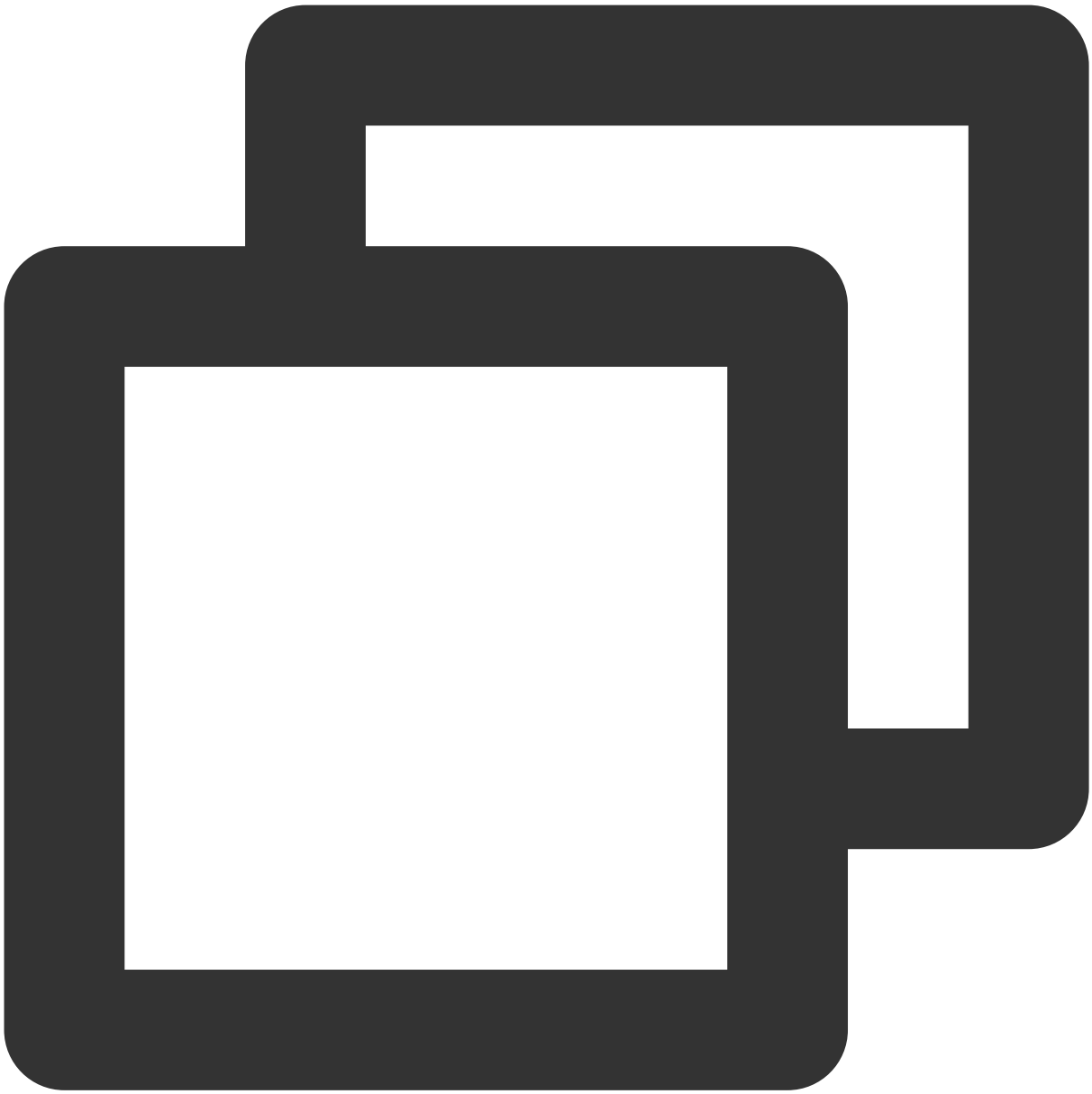
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records
```

请求 HTTP 头：



```
[
 "x-tcaplus-target:Tcaplus.FieldIncRecord",
 "x-tcaplus-version:Tcaplus3.32.0",
 "x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
 "x-tcaplus-result-flag:1",
 "x-tcaplus-data-version-check:1",
 "x-tcaplus-data-version:-1",
 "x-tcaplus-idl-type:Protobuf"
]
```

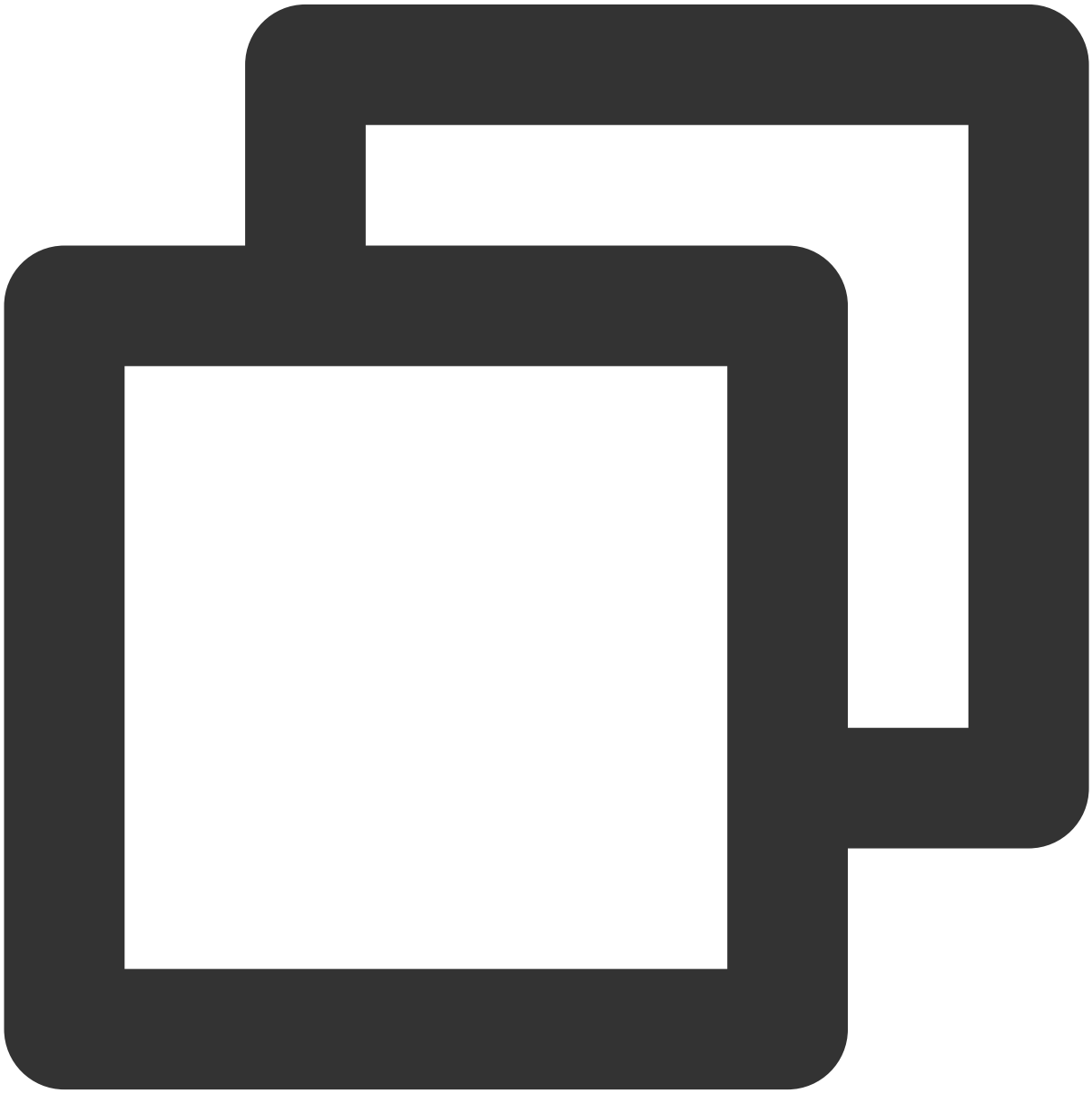
请求 JSON 数据：



```
{
 "ReturnValues": "aaaaaaaaaa",
 "Record": {
  "name": "calvinshao",
  "pay": {
   "total_money": -1,
   "auth": {
    "update_time": -1
   }
```

```
    },
    "region": 101,
    "uin": 100,
    "gamesvrid": 2,
    "logintime": -2
  }
}
```

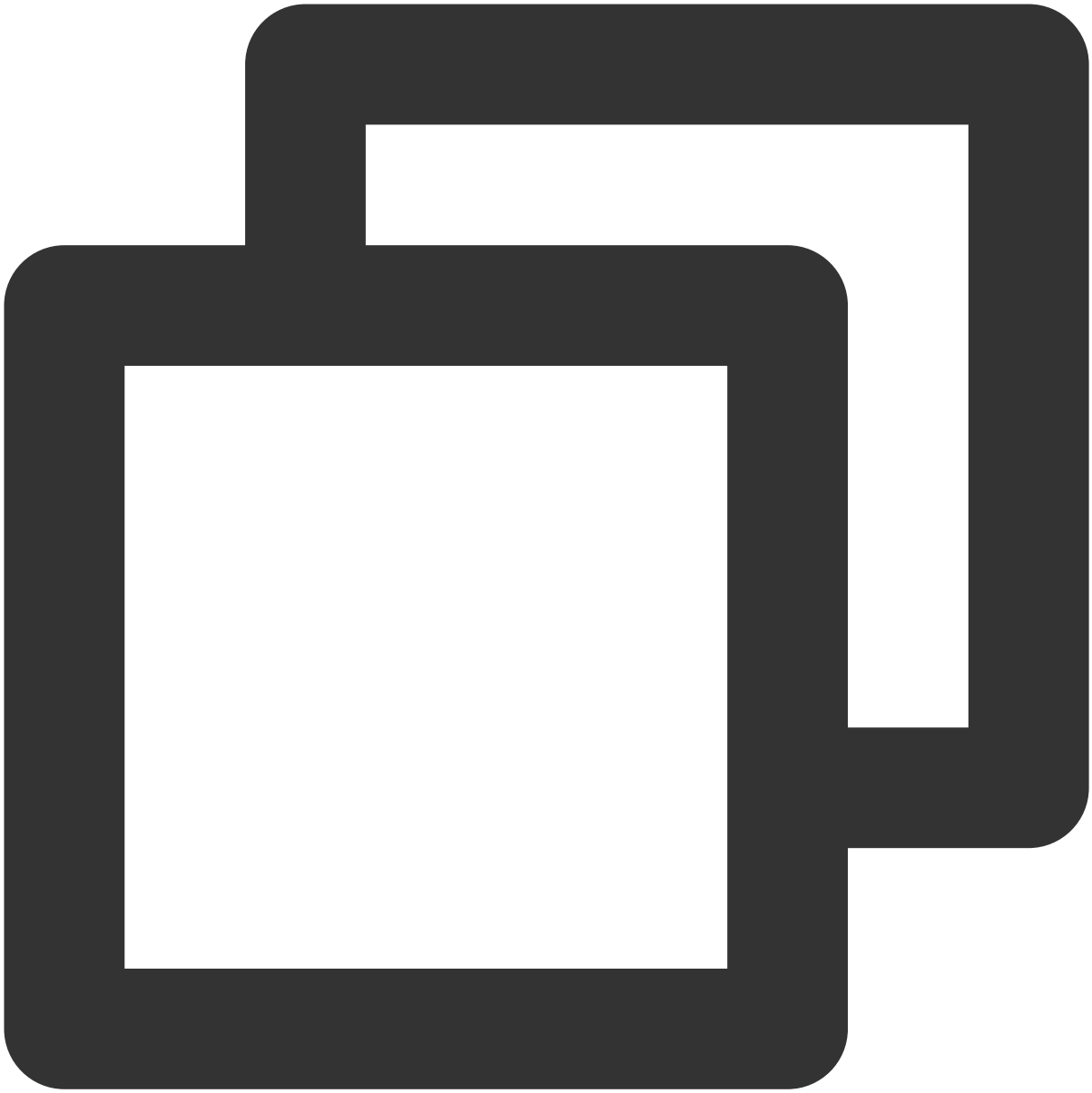
应答 JSON 数据：



```
{
```

```
  "ErrorCode": 0,
  "ErrorMsg": "Succeed",
  "RecordVersion": 9,
  "ReturnValues": "aaaaaaaaaa",
  "Record": {
   "name": "calvinshao",
   "pay": {
    "total_money": 11999,
    "auth": {
     "update_time": 921
    }
   },
   "region": 101,
   "uin": 100,
   "gamesvrid": 4101,
   "logintime": 98
  }
 }
```

Tcaplus.PartkeyGetRecord



```
GET /ver1.0/apps/{APP_ID}/zones/{ZONE_ID}/tables/{TABLE_NAME}/records?keys={JSONKey
```

通过指定部分主键的值查询多条记录。这个操作将返回多条数据，并且通过 select 变量指定的字段名显示。此操作的前提是指定的主键集合必须在建表的时候创建了索引，否则会返回错误。

必须在 URI 中指定 `keys` 变量，而 select 变量则是可选项。keys 指定所有主键的值，select 指定需要显示的 value 字段的名称。并且用户可以通过点分路径的方式指定嵌套结构中的字段，例如：“pay.total_money”。

limit 和 offset 是用于记录部分返回控制的参数。

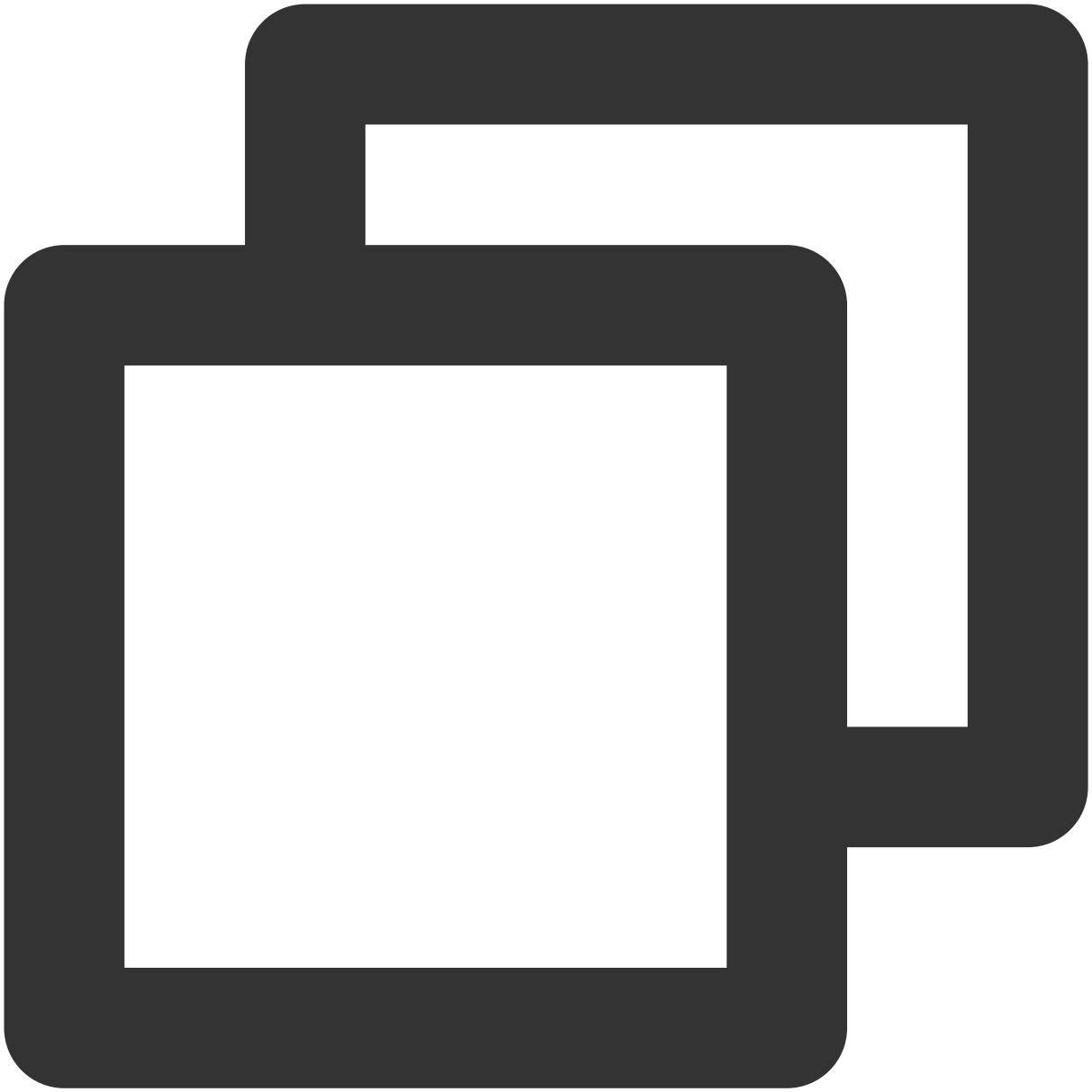
注意：

请求的变量必须通过 urlencode 编码，请将 url 中的空格编码为“%20”而不是“+”；请在 HEADER中 通过 `x-tcaplus-index-name` 指定想要访问的索引名，索引名在表定义文件中可以找到。

名称	类型	取值
x-tcaplus-target	String	Tcaplus.GetRecord
x-tcaplus-version	String	Tcaplus3.32.0
x-tcaplus-pwd-md5	String	MD5 of AppKey(Password)
x-tcaplus-idl-type	String	protobuf
x-tcaplus-index-name	String	{index_name}

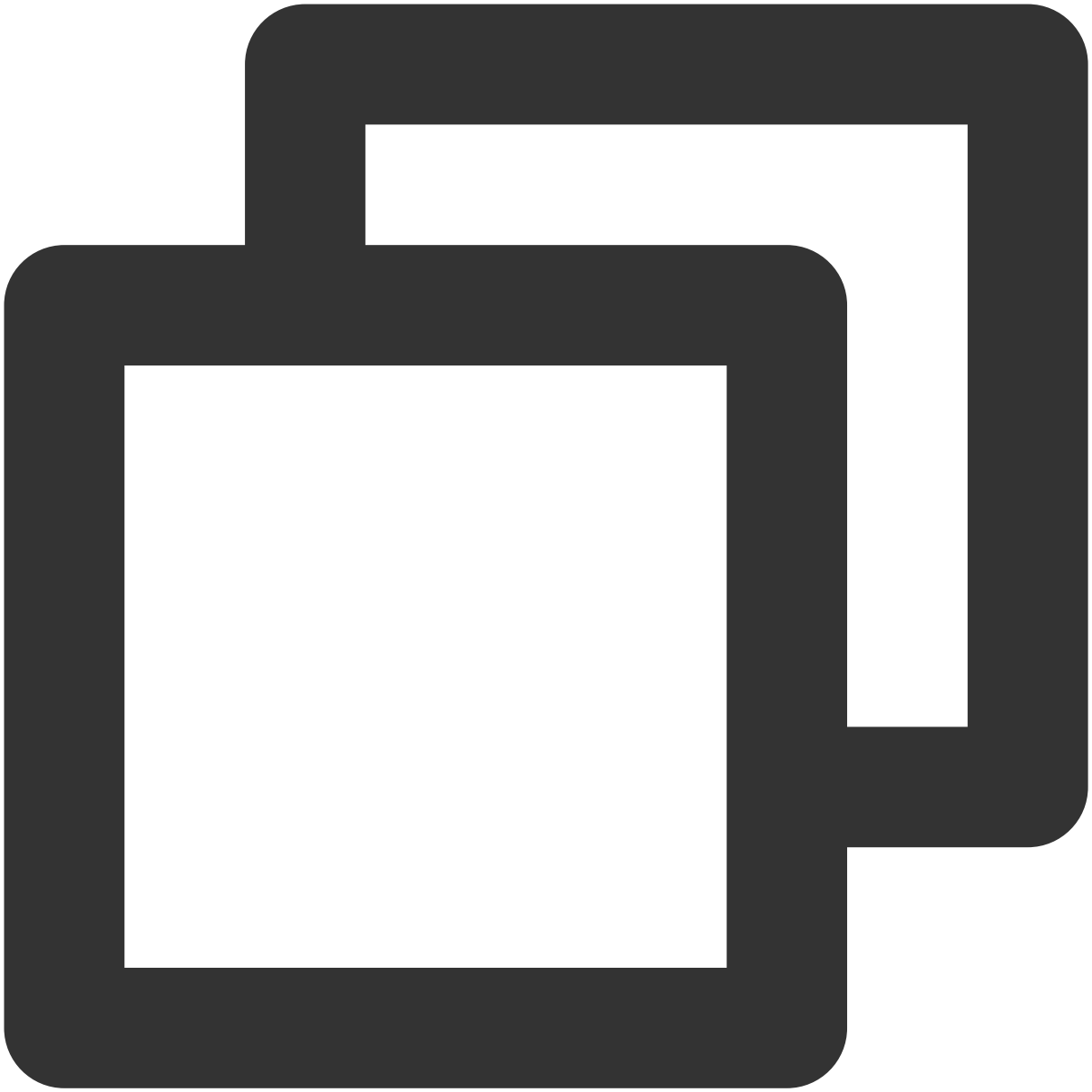
示例：

URL 未 UrlEncode 结果：



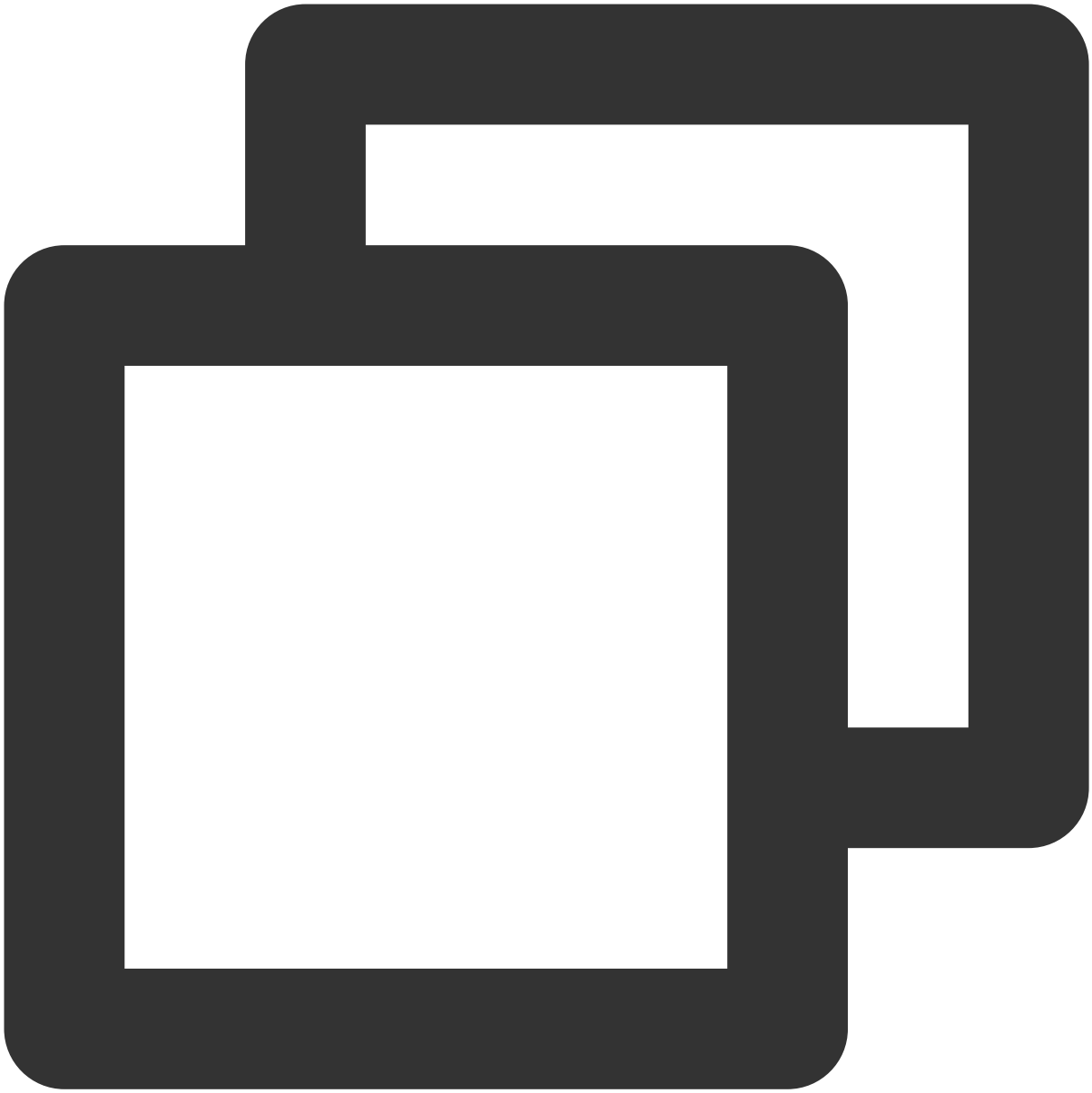
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records?keys={'name': 'c
```

URL UrlEncode 结果：



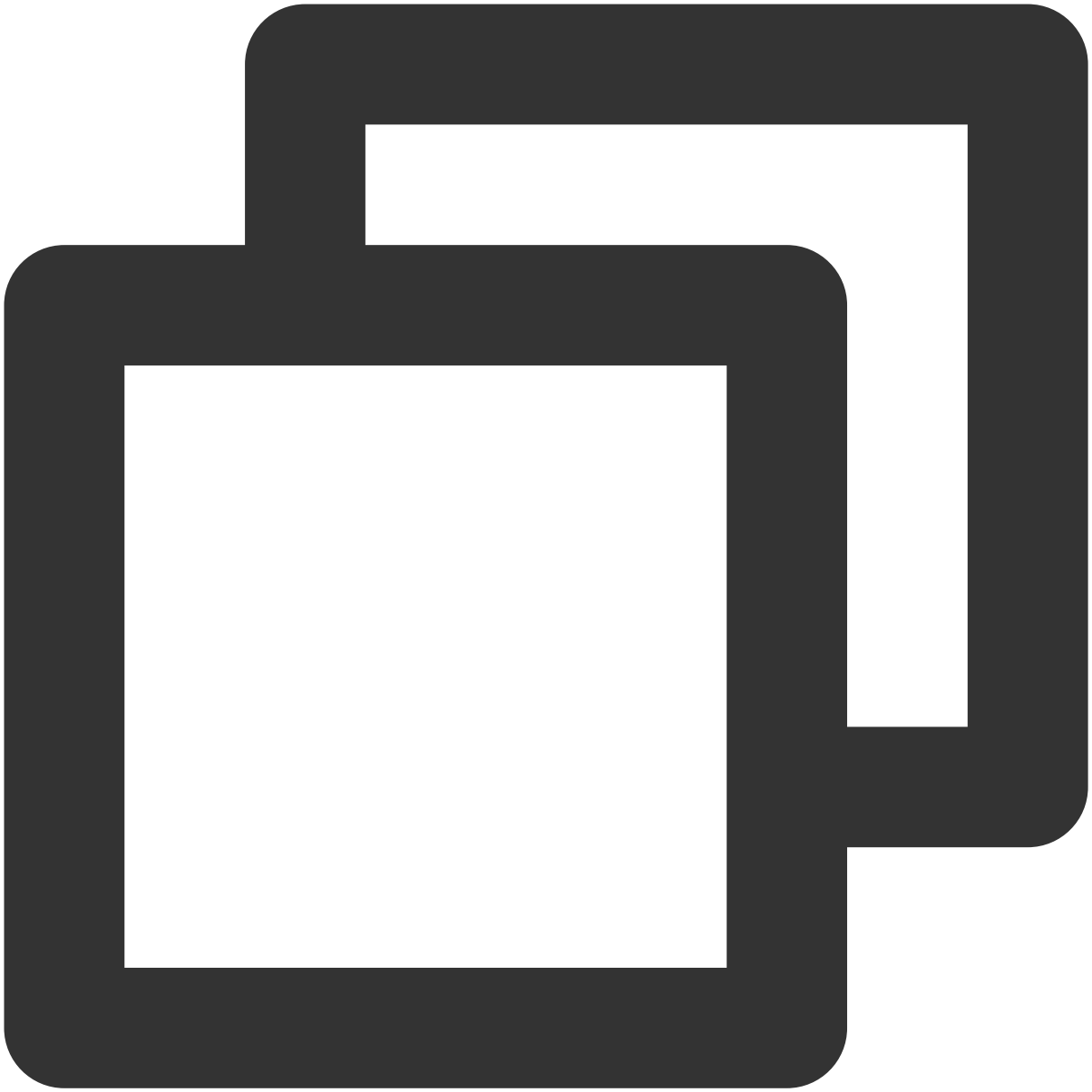
```
http://10.123.9.70/ver1.0/apps/2/zones/1/tables/tb_example/records?keys=%7B%22name%
```

请求 HTTP 头：



```
[
"x-tcaplus-target:Tcaplus.PartkeyGetRecord",
"x-tcaplus-version:Tcaplus3.32.0",
"x-tcaplus-pwd-md5:c3eda5f013f92c81dda7afcdc273cf82",
"x-tcaplus-idl-type:protobuf"
"x-tcaplus-index-name:index_name"
]
```

应答数据：




```
{
 "ErrorCode": 0,
 "ErrorMsg": "Succeed",
 "MultiRecords": [
  {
   "RecordVersion": 9,
   "Record": {
    "name": "calvinshao",
    "lockid": [
     50,
     60,
```

```
      70,
      80,
      90,
      100
     ],
     "pay": {
      "total_money": 11999,
      "auth": {
       "pay_keys": "adqwacsasafasda"
      }
     },
     "region": 101,
     "uin": 100,
     "gamesvrid": 4101
    }
   },
   {
    "RecordVersion": 1,
    "Record": {
     "name": "calvinshao",
     "lockid": [
      50,
      60,
      70,
      80
     ],
     "pay": {
      "total_money": 10000,
      "auth": {
       "pay_keys": "adqwacsasafasda"
      }
     },
     "region": 102,
     "uin": 100,
     "gamesvrid": 4100
    }
   },
   {
    "RecordVersion": 1,
    "Record": {
     "name": "calvinshao",
     "lockid": [
      60,
      70,
      80,
      90
     ],
```

```
      "pay": {
       "total_money": 10000,
       "auth": {
        "pay_keys": "adqwacsasafasda"
       }
      },
      "region": 103,
      "uin": 100,
      "gamesvrid": 4101
     }
    }
  ],
  "RemainNum": 0,
  "TotalNum": 3
}
```

Go RESTful API 接口说明

最近更新时间：2021-04-06 15:40:12

为满足用户通过 Golang 语言来操作 TcaplusDB，基于 RESTful API 封装了关于 TcaplusDB 表操作的接口，涵盖增删查改场景。本文主要为您介绍基于此 RESTful API 来操作 TcaplusDB PB 表。

准备工作

1. 创建 TcaplusDB 表

创建 TcaplusDB 示例表，示例表为 `game_players.proto`，请参见 创建表格。

2. 创建 CVM 实例

- 创建一台 CVM 实例来运行 SDK 示例程序，配置建议为2核4GB、硬盘50GB，该 CVM 需创建在 TcaplusDB 实例所在 VPC 网络中。
- 通过 SDK 下载 Go RESTful API SDK 安装包。

3. 准备 Go 环境

安装 Golang 执行环境，安装命令如下：

```
yum install -y golang
```

4. 编译程序

SDK 示例程序通过 make 编译，在 src 目录下有 Makefile 文件，直接执行 `make build` 即可。编译好后，会生成一个可执行文件 `example`，直接执行此文件即可演示所有示例接口。

使用步骤

1. 配置表参数

在 TcaplusDB 控制台 查看所创建表的相关信息，在示例中进行配置，如下所示：

```
//TcaplusDB RESTful API 的连接参数
const (
//服务接入点，表所在集群 RESTful 连接地址，默认端口80
EndPoint = "http://172.xx.xx.12/"
//应用接入 ID，表所在集群接入ID
```

```
AccessID = 310
//应用密码,表所在集群访问密码
AccessPassword = "Tcaplus2020"
//表格组ID
TableGroupID = 1
//表名称
TableName = "game_players"
)
```

2. 创建表连接

通过 NewTcaplusClient 指定 EndPoint、AccessID、AccessPassword 参数，创建 TcaplusRestClient 的对象 client。

```
//通过指定 EndPoint、AccessID、AccessPassword参数, 创建 TcaplusClient 的对象 client
client, err := tcaplus_client.NewTcaplusClient(EndPoint, AccessID, AccessPasswor
d)
if err != nil {
fmt.Println(err.Error())
return
}
```

3. 表示例数据

插入数据为 JSON 格式，定义如下所示：

```
//AddRecord 插入记录
//用户可将 record 定义成结构体/map/slice, 需可转成 json
record := map[string]interface{}{
"player_id": 10805514,
"player_name": "Calvin",
"player_email": "calvin@test.com",
"game_server_id": 10,
"login_timestamp": []string{"2019-12-12 15:00:00"},
"logout_timestamp": []string{"2019-12-12 16:00:00"},
"is_online": false,
"pay": map[string]interface{}{
"pay_id": 10101,
"amount": 1000,
"method": 1,
},
}
status, resp, err := client.AddRecord(record, tcaplus_client.RetAllLatestField,
"userBuffer", TableGroupID, TableName)
if err != nil {
fmt.Println(err.Error())
```

```
return
}
```

接口列表

GetRecord 记录查询

```
/**
@brief 根据 Key 字段查询记录，可以根据 selectFiled 过滤出部分字段
@param [IN] key 需要查询的记录的 key 字段信息，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] selectFiled 需要查询的记录的字段列表，结构体嵌套则设为点分式，为 nil 表示查询全部字段
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) GetRecord(key interface{}, selectFiled []string, groupID int, tableName string) (string, []byte, error)
```

AddRecord 插入记录

```
/**
@brief 添加一条记录到表中，该记录若存在，则会报错
@param [IN] record 需要添加的记录，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] resultFlag 返回值标记位，可设置为以下值
RetOnlySucOrFail：应答中仅包含请求成功或失败
RetEqualReq：应答中包含与请求一致的值
RetAllLatestField：应答中包含被修改的数据的所有字段最新值
RetAllOldField：应答中包含记录被修改前的值
@param [IN] userBuffer 用户自定义信息，在响应信息中原样返回，不关注则填""
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) AddRecord(record interface{}, resultFlag int, userBuffer string, groupID int, tableName string) (string, []byte, error)
```

SetRecord 设置记录

```
/**
@brief 更新/插入一条记录到表中，该记录若存在，则更新；不存在，则插入
@param [IN] record 需要添加的记录，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] resultFlag 返回值标记位，可设置为以下值
RetOnlySucOrFail：应答中仅包含请求成功或失败
RetEqualReq：应答中包含与请求一致的值
RetAllLatestField：应答中包含被修改的数据的所有字段最新值
RetAllOldField：应答中包含记录被修改前的值
@param [IN] versionPolicy 记录的版本号校验策略，与 version 配合使用，用于乐观锁，不关注则设置为 NoCheckDataVersionAutoIncrease
CheckDataVersionAutoIncrease：检测记录版本号，只有当 version 与服务器端的版本号相同时，操作成功，记录版本号自增
NoCheckDataVersionOverwrite：不检测记录版本号，强制把记录版本号 version 写入到服务器中
NoCheckDataVersionAutoIncrease：不检测记录版本号，服务器端的版本号自增
@param [IN] version 记录的版本号，用于版本号校验，不校验则设置为-1
@param [IN] userBuffer 用户自定义信息，在响应信息中原样返回，不关注则填""
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) SetRecord(record interface{}, resultFlag int, versionPolicy int, version int, userBuffer string, groupID int, tableName string) (string, []byte, error)
```

DeleteRecord 删除记录

```
/**
@brief 删除一条记录，不存在，则报错
@param [IN] record 需要删除的记录，包含 key 字段即可；可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] resultFlag 返回值标记位，可设置为以下值
RetOnlySucOrFail：应答中仅包含请求成功或失败
RetEqualReq：应答中包含与请求一致的值
RetAllLatestField：应答中包含被修改的数据的所有字段最新值
RetAllOldField：应答中包含记录被修改前的值
@param [IN] userBuffer 用户自定义信息，在响应信息中原样返回，不关注则填""
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) DeleteRecord(record interface{}, resultFlag int, userBuffer string, groupID int, tableName string) (string, []byte, error)
```

FieldGetRecord 指定字段查询

```
/**
@brief 记录的部分字段查询，根据 key 字段查询记录，根据 selectFiled 过滤字段内容
@note 该接口与 GetRecord 的区别：GetRecord 是查询的整条记录然后按 selectFiled 过滤；而 FieldGetRecord 是在 svr 端过滤，流量负载更低
@param [IN] key 需要查询的记录的 key 字段信息，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] selectFiled 需要查询的记录的字段列表，不能为空，结构体嵌套则设为点分式
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) FieldGetRecord(key interface{}, selectFiled []string, groupID int, tableName string) (string, []byte, error)
```

FieldSetRecord 指定字段设置

```
/**
@brief 更新一条记录，该记录若不存在则报错
@param [IN] record 需要添加的记录，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] setField 需要更新的字段列表，不能为空，结构体嵌套则设为点分式
@param [IN] resultFlag 返回值标记位，可设置为以下值
RetOnlySucOrFail：应答中仅包含请求成功或失败
RetEqualReq：应答中包含与请求一致的值
@param [IN] versionPolicy 记录的版本号校验策略，与 version 配合使用，用于乐观锁，不关注则设置为 NoCheckDataVersionAutoIncrease
CheckDataVersionAutoIncrease：检测记录版本号，只有当 version 与服务器端的版本号相同时，操作成功，记录版本号自增
NoCheckDataVersionOverwrite：不检测记录版本号，强制把记录版本号 version 写入到服务器中
NoCheckDataVersionAutoIncrease：不检测记录版本号，服务器端的版本号自增
@param [IN] version 记录的版本号，用于版本号校验，不校验则设置为-1
@param [IN] userBuffer 用户自定义信息，在响应信息中原样返回，不关注则填""
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) FieldSetRecord(record interface{}, setField []string, resultFlag int, versionPolicy int, version int, userBuffer string, groupID int, tableName string) (string, []byte, error)
```

FieldIncRecord 指定字段自增/自减

```
/**
@brief 对记录中的整型字段进行自增/自减，此命令字仅支持 int32，int64，uint32 和 uint64类型字段
@param [IN] record 需要更新的的记录，记录中的字段值为正，则表示自增，累加该值；为负，则表示自减，累减该值
@param [IN] resultFlag 返回值标记位，可设置为以下值
RetOnlySucOrFail：应答中仅包含请求成功或失败
RetEqualReq：应答中包含与请求一致的值
@param [IN] versionPolicy 记录的版本号校验策略，与 version 配合使用，用于乐观锁，不关注则设置为 NoCheckDataVersionAutoIncrease
CheckDataVersionAutoIncrease：检测记录版本号，只有当 version 与服务器端的版本号相同时，操作成功，记录版本号自增
NoCheckDataVersionOverwrite：不检测记录版本号，强制把记录版本号 version 写入到服务器中
NoCheckDataVersionAutoIncrease：不检测记录版本号，服务器端的版本号自增
@param [IN] version 记录的版本号，用于版本号校验，不校验则设置为-1
@param [IN] userBuffer 用户自定义信息，在响应信息中原样返回，不关注则填""
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) FieldIncRecord(record interface{}, resultFlag int, versionPolicy int, version int, userBuffer string, groupID int, tableName string) (string, []byte, error)
```

PartkeyGetRecord 索引查询，仅指定索引键

```
/**
@brief 按索引进行批量查询
@param [IN] key 需要查询的记录，仅需要索引包含的 key 字段，可以是结构体，map，slice，从而转成 json，与 proto 中定义的类型保持一致
@param [IN] indexName 索引名称
@param [IN] selectFiled 需要查询的记录的字段列表，结构体嵌套则设为点分式；为 nil 表示查询全部字段
@param [IN] limit 批量返回的记录上限，>0有效
@param [IN] offset 批量返回的记录的偏移，>=0有效
@param [IN] groupID 表格组 ID
@param [IN] tableName 表名
@retval(3) http 响应码，http 响应内容，错误信息
**/
func (c *TcaplusClient) PartKeyGetRecord(key interface{}, indexName string, selectFiled []string, limit int, offset int, groupID int, tableName string) (string, []byte, error)
```

对于 PartKeyGetRecord 接口，1个请求返回的最大包大小为 `256KB` ，limit 的设置依赖于单条记录大小。推荐设置策略：

- 单条记录小于256KB：limit 参考设置为 256KB / [单条记录大小]，如记录大小为10KB，则 limit 推荐设置20 - 25左右。
- 单条记录大于等于256KB：limit 设置为1，即一次请求只返回一条记录。

对于设置 limit 和 offset 的场景，如果要根据索引键获取全量的数据，则需要依据响应包中返回的 `TotalNum` 和 `RemainNum` 标识来判断数据是否获取完全。

Java RESTful API 接口说明

最近更新时间：2021-04-06 15:40:37

为满足用户使用 Java 操作 TcaplusDB 表，TcaplusDB 基于 RESTful API 封装了 Java SDK。本文主要为您介绍基于此 Java RESTful API 来操作 TcaplusDB PB 表（基于 protobuf 协议）。

准备工作

1. 创建 TcaplusDB 表

创建 TcaplusDB 示例表，示例表为 `game_players.proto`，请参见 创建表格。

2. 创建 CVM 实例

- 创建一台 CVM 实例来运行 SDK 示例程序，配置建议为1核2GB、Centos 7，该 CVM 需创建在 TcaplusDB 实例所在 VPC 网络中。
- 通过 SDK 下载 Java RESTful API SDK 安装包。

3. 准备 Java 环境

SDK 依赖 Java 1.8 以上环境，CVM 操作系统为 CentOS 7 及以上版本时，执行 `yum install -y java` 即可。

4. 接口说明

目前 SDK 示例支持8个接口，详情如下：

接口名	接口用途
addRecord	增加一条 TcaplusDB 记录，记录必须包含所有主键字段值
getRecord	查询一条 TcaplusDB 记录，必须指定主键字段查询
setRecord	更新一条 TcaplusDB 记录，必须指定主键字段更新，若记录不存在则插入一条新记录
deleteRecord	删除一条 TcaplusDB 记录，必须指定主键字段删除
fieldSetRecord	更新指定字段，必须指定主键字段更新
fieldGetRecord	获取指定字段，必须指定主键字段获取
fieldIncRecord	更新指定字段，只针对数值类型字段，自增或自减场景，必须指定主键字段更新
partkeyGetRecord	根据指定的索引名和对应的索引字段值获取相应 TcaplusDB 记录（一条或多条），必须指定对应的索引字段值

使用步骤

上传整个 SDK 包 `tcaplusdb-restapi-java-sdk-1.0-assembly.zip` 到 CVM 目录，请参见 如何将本地文件拷贝到云服务器，解压后目录名 `tcaplusdb-restapi-java-sdk-1.0` 。

1. 配置准备

配置文件在 `tcaplusdb-restapi-java-sdk-1.0/conf/config.properties` 。配置内容如下：

```
#replace with your table endpoint
endpoint=http://10.xx.xx.16

#replace with your table access id
access_id=60

#replace with your table access password
access_password=Tcaxxx19

#replace with your table group id
table_group_id=1

#replace with your table name
table_name=game_players

#optional, configure the returning fields for GetRecord API, multiple fields with
comma delimeter
get_select_fields=

#required, configure the returing fields for FieldGetRecord API
field_get_select_fields=game_server_id,is_online

#optional, configure the returning fields for PartkeyGetRecord API
part_key_get_select_fields=game_server_id,is_online

#required, configure the updating fields for FieldSetRecord API
field_set_fields=game_server_id,pay.amount

#required, configure the index keys for PartkeyGetRecord API
part_key_index_keys=player_id,player_name

#required, configure the index name for PartkeyGetRecord API
part_key_index_name=index_1

#optional, configure the limit number of records per request to return for Partke
yGetRecord API, value range: >0
part_key_limit = 10
```

```
#optional, configure the offset position to return for PartkeyGetRecord API, valu
e range: >=0
part_key_offset = 0
```

2. 准备数据

示例为每个 RESTful API 接口单独准备测试数据，以 json 文件方式放在 `tcaplusdb-restapi-java-sdk-1.0/conf` 目录。

文件名	文件说明
add_data.json	用于 addRecord 示例接口，增加一条记录
set_data.json	用于 updateRecord 示例接口，更新一条记录字段值
delete_data.json	用于 deleteRecord 示例接口，删除记录的主键值
get_data.json	用于 getRecord 示例接口，获取记录的主键值
field_get_data.json	用于 fieldGetRecord 示例接口，获取部分字段的值
field_set_data.json	用于 fieldSetRecord 示例接口，更新部分字段的值
field_inc_data.json	用于 fieldIncRecord 示例接口，增加部分字段的值，针对数值类型字段
partkey_get_data.json	用于 partkeyGetRecord 示例接口，根据索引获取记录值

3. 执行脚本

主要执行脚本在 `tcaplusdb-restapi-java-sdk-1.0/bin/run.sh` ，执行方法如下：

```
#增加一条记录
sh bin/run.sh add
#获取一条记录
sh bin/run.sh get
#更新一条记录
sh bin/run.sh set
#删除一条记录
sh bin/run.sh delete
#更新部分字段
sh bin/run.sh field_set
#获取部分字段
sh bin/run.sh field_get
#更新部分字段值（数据类型）
sh bin/run.sh field_inc
```

```
#根据索引获取一条记录
sh bin/run.sh partkey_get
```

接口数据说明

addRecord 接口

增加一条记录，插入的数据格式为 JSON 格式，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com", "game_server
_id":2,"login_timestamp":[],"logout_timestamp":[],"is_online":true,"pay":{"pay_i
d":1,"amount":20,"method":3}}
```

getRecord 接口

查询一条记录，指定记录的主键字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com"}
```

查询接口还支持指定要返回的字段列表，在配置文件 `config.properties` 中定义了 `get_select_fields` 配置项，用于指定要返回的字段名，如果配置为空则默认返回记录所有字段，如果不为空则返回指定的字段值。

setRecord 接口

更新一条记录，更新记录必须包含主键字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com", "game_server
_id":2,"login_timestamp":[],"logout_timestamp":[],"is_online":false,"pay":{"pay_i
d":1,"amount":30,"method":3}}
```

deleteRecord 接口

删除一条记录，输入数据必须包含主键字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com"}
```

fieldSetRecord 接口

更新部分字段值，输入数据必须包含主键字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com", "game_server
_id":3,"is_online":false, "login_timestamp":[],"logout_timestamp":[],"pay":{"pay_
```

```
id":1,"amount":40,"method":3}}
```

同时还需要指定更新的部分字段名，在配置文件 `config.properties` 中定义了 `field_set_fields` 配置项，指定对应的字段名即可，用逗号隔开，**不能为空**。

fieldGetRecord 接口

获取部分字段值，输入数据必须包含主键字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com"}
```

同时还需要指定要获取的部分字段名，在配置文件 `config.properties` 中定义了 `field_get_select_fields` 配置项， 指定需要返回的字段值，**不能为空**。

fieldIncRecord 接口

更新部分字段值，与 fieldSetRecord 接口不同的是，此接口只支持更新数值类型的字段值，如增加或减少相应值，输入数据必须包含主键字段和要更新数值字段，数据格式为 JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com", "pay":{"amou
nt":50}}
```

假设 pay.amount 初始值为100，请求传参值为50，则最终 pay.amount 值为150。

partkeyGetRecord 接口

根据指定索引获取对应记录，可能对应多条记录，支持返回指定字段的值，输入数据必须包含索引字段，数据格式为JSON，示例数据如下所示：

```
{"player_id":1,"player_name":"test","player_email":"test@email.com"}
```

同时还必须指定对应的索引名和索引字段名，在配置文件 `config.properties` 中定义： `part_key_index_name` 表示索引名， `part_key_index_keys` 表示索引字段名，**这两个配置项不能为空**。

如果要返回指定字段，支持在配置文件中配置 `part_key_get_select_fields` 来指定要返回的字段，可以为空。

对于此接口，1个请求返回的最大包大小为 `256KB` ，limit 的设置依赖于单条记录大小。推荐设置策略：

- 单条记录小于256KB：limit 参考设置为 256KB / [单条记录大小]，如记录大小为10KB，则 limit 推荐设置20 - 25左右。
- 单条记录大于等于256KB：limit 设置为1，即一次请求只返回一条记录。

对于设置 limit 和 offset 的场景，如果要根据索引键获取全量的数据，则需要依据响应包中返回的 `TotalNum` 和 `RemainNum` 标识来判断数据是否获取完全。

设置 limit 和 offset 的场景演示可按如下步骤进行：

```
#step1，准备批量演示数据，参考 conf/batch_add_data.json. 设置 limit 和 offset 值
part_key_limit = 2
part_key_offset = 0

#step2，批量增加演示数据，用 batch_add 命令，增加3条记录
sh bin/run.sh batch_add

#step3，调用 partkey_get 命令，会执行 partkeyGetRecord 两次：一次不带 limit 和 offset，一次带 limit 和 offset，观察两次执行的结果差异。
sh bin/run.sh partkey_get

#step4，不带 limit 和 offset 的响应数据如下，总共4条数据
{"MultiRecords":[{"RecordVersion":3,"Record":{"game_server_id":3,"is_online":true,"player_email":"test@email.com","player_id":1,"player_name":"test"}},{"RecordVersion":1,"Record":{"game_server_id":3,"is_online":true,"player_email":"test1@email.com","player_id":1,"player_name":"test"}},{"RecordVersion":1,"Record":{"game_server_id":4,"is_online":true,"player_email":"test2@email.com","player_id":1,"player_name":"test"}},{"RecordVersion":1,"Record":{"game_server_id":5,"is_online":true,"player_email":"test3@email.com","player_id":1,"player_name":"test"}}],"TotalNum":4,"RemainNum":0,"ErrorCode":0,"ErrorMsg":"Succeed"}

#step5，带 limit 和 offset 的响应数据如下，只返回2条和 limit 配置值一致
{"MultiRecords":[{"RecordVersion":3,"Record":{"game_server_id":3,"is_online":true,"player_email":"test@email.com","player_id":1,"player_name":"test"}},{"RecordVersion":1,"Record":{"game_server_id":3,"is_online":true,"player_email":"test1@email.com","player_id":1,"player_name":"test"}}],"TotalNum":4,"RemainNum":2,"ErrorCode":0,"ErrorMsg":"Succeed"}
```

PHP RESTful API 接口说明

最近更新时间：2023-12-18 14:57:47

准备工作

1. 创建 TcaplusDB 表

创建 TcaplusDB 示例表，示例表为 `game_players.proto` ，请参见 创建表格。

2. 创建 CVM 实例

创建一台 CVM 实例来运行 SDK 示例程序，配置建议为1核2GB、Centos 7，该 CVM 需创建在 TcaplusDB 实例所在 VPC 网络中。
通过 SDK 下载 PHP RESTful API SDK 安装包。

3. 准备 PHP 环境

新申请实例未安装 PHP，需要执行 `yum install php` 安装默认版本的 PHP 5.4.16，PHP 版本要求大于5.3。

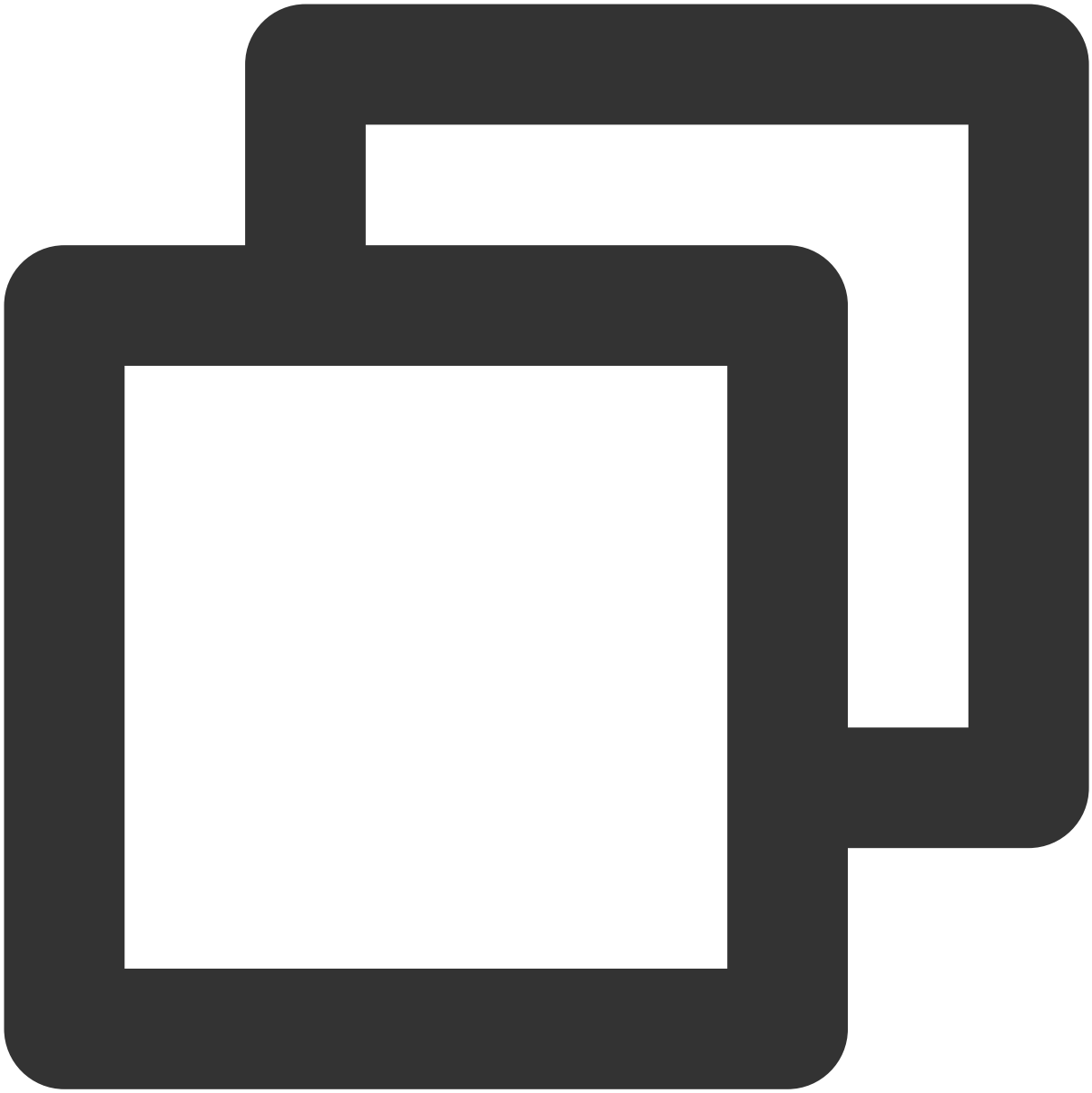
使用方法

1. 修改配置

所有基础配置统一放在 SDK 安装目录下的 `config.php` 中，用户需要根据申请的表信息进行相应更改。
连接配置：如 url、endpoint、table_group_id、access_id、access_passwd 等，修改成对应值，连接信息获取请参考 获取连接信息。
默认数据源：此部分变量都为大写，用于 demo.php 中具体接口函数的数据源，可以统一在此修改。

2. 调用方式

为方便测试单个接口的功能，TcaplusDB 还把所有 SDK 接口函数单独成了程序，分别对应 demo 中的8个接口函数，采用的也是数据源与程序代码分离管理方式。
所有接口函数的数据源有两个地方，一个是默认的数据源，统一在 `config.php` 中，另一个是每个接口函数自己的数据源，在 `data` 子目录下，用户可以修改该数据文件来执行对应的接口函数，方便批量操作时避免在程序代码中定义数据源。如果数据文件不存在则会加载默认数据源。数据源统一采用数据列表方式，支持批量加载。
sample 中的程序文件执行方法类似：



```
php -f <具体的接口函数文件>
```

3. 接口函数

所有 API 接口封装成函数统一在 demo.php 中，总共有8个接口函数，所有接口函数支持从配置中加载数据源进行批量操作，具体函数如下所示：

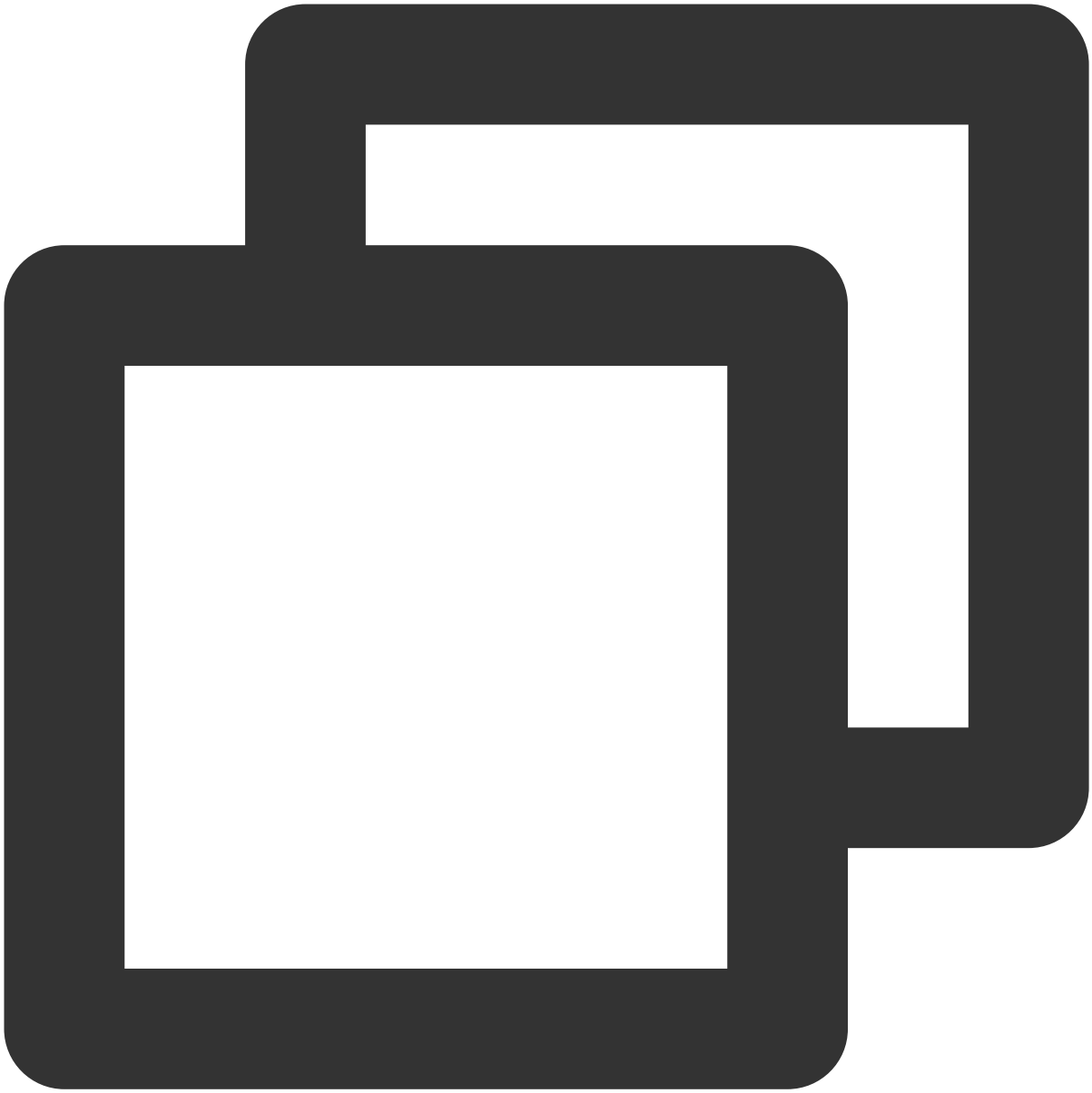
接口函数名	接口函数功能
GetRecord	获取表记录

AddRecord	插入表记录
SetRecord	更新表记录
DeleteRecord	删除记录
FieldGetRecord	获取指定记录属性字段
FieldSetRecord	更新指定记录属性字段值
FieldIncRecord	更新指定记录属性字段数值，如增加/减少数值类型值
PartKeyGetRecord	根据指定索引获取指定记录属性字段值

SDK 接口列表

add

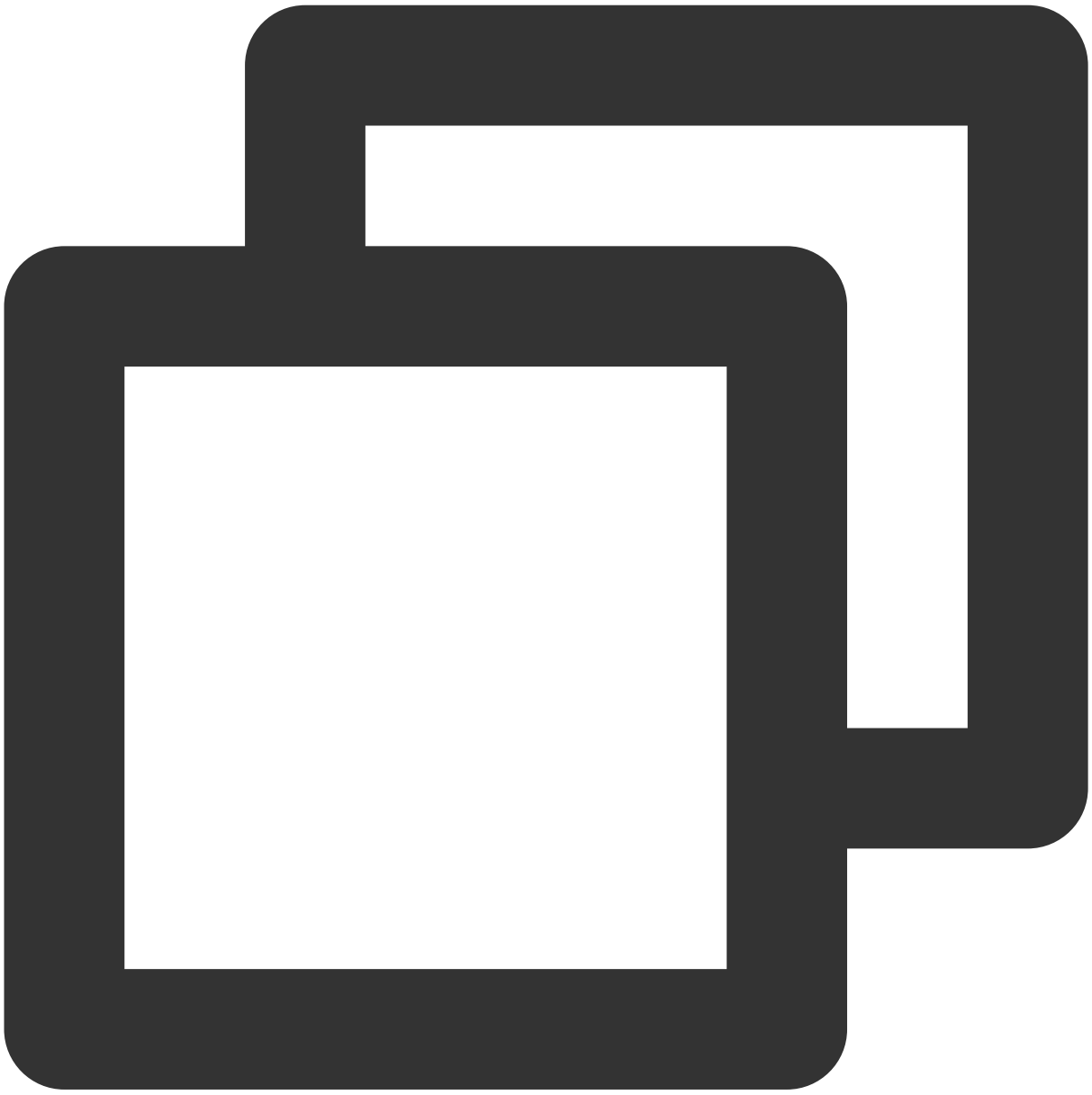
插入一条表记录，如果记录存在则报错。



```
@param $table_group_id （必须） - 目标表 table_group_id
@param $table_name （必须） - 目标表 table_name
@param $record （必须） - 待插入的目标记录数组
@param $ReturnValues （可选） - 指定返回的自定义值
@param $resultflag （可选） - 指定返回响应包的内容，取值范围如下：
            0：应答中仅包含请求成功或失败
            1：应答中包含与请求一致的值
            2：应答中包含被修改的数据的所有字段最新值
            3：应答中包含记录被修改前的值

public function add($table_group_id, $table_name, array $record, $ReturnValues = 'T
```

get

根据主键查询一条记录，支持指定要返回的属性字段列表，参考 `$params["select"]` 定义。

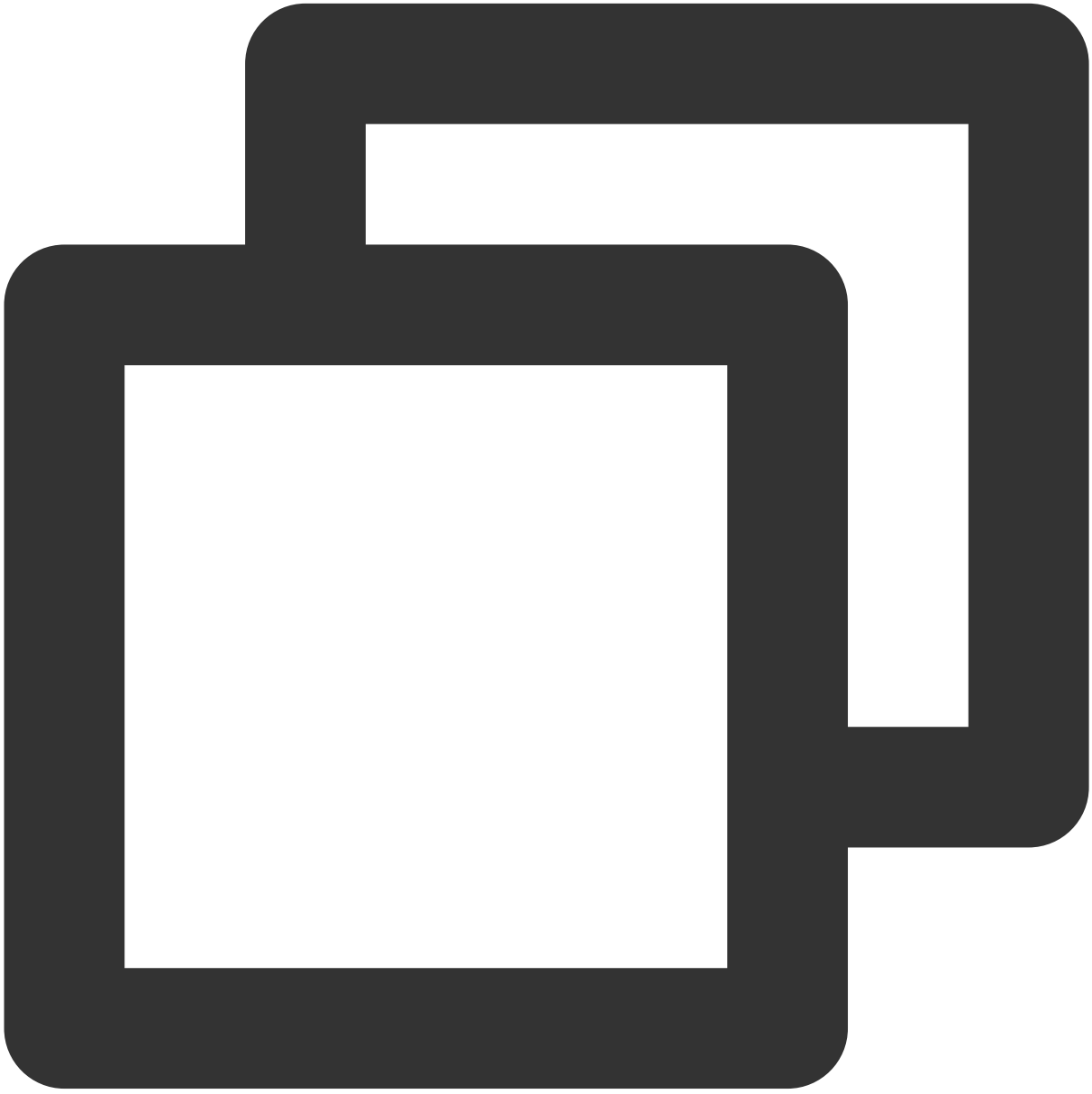


```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param $params （必须）
   $params\\["select"\\] （可选） 查询的字段 数组
   $params \\["keys"\\] （必须） 查询目标记录的主键字段(Primary key)
   $params \\["limit"\\] （可选） 返回记录数限制量
```

```
    $params \\["offset"\\] (可选) 返回记录数偏移量

public function get($table_group_id, $table_name, $params)
```

fieldGet

根据主键查询返回指定字段的值，指定字段列表参考 `$param["select"]`，该参数必须为非空。

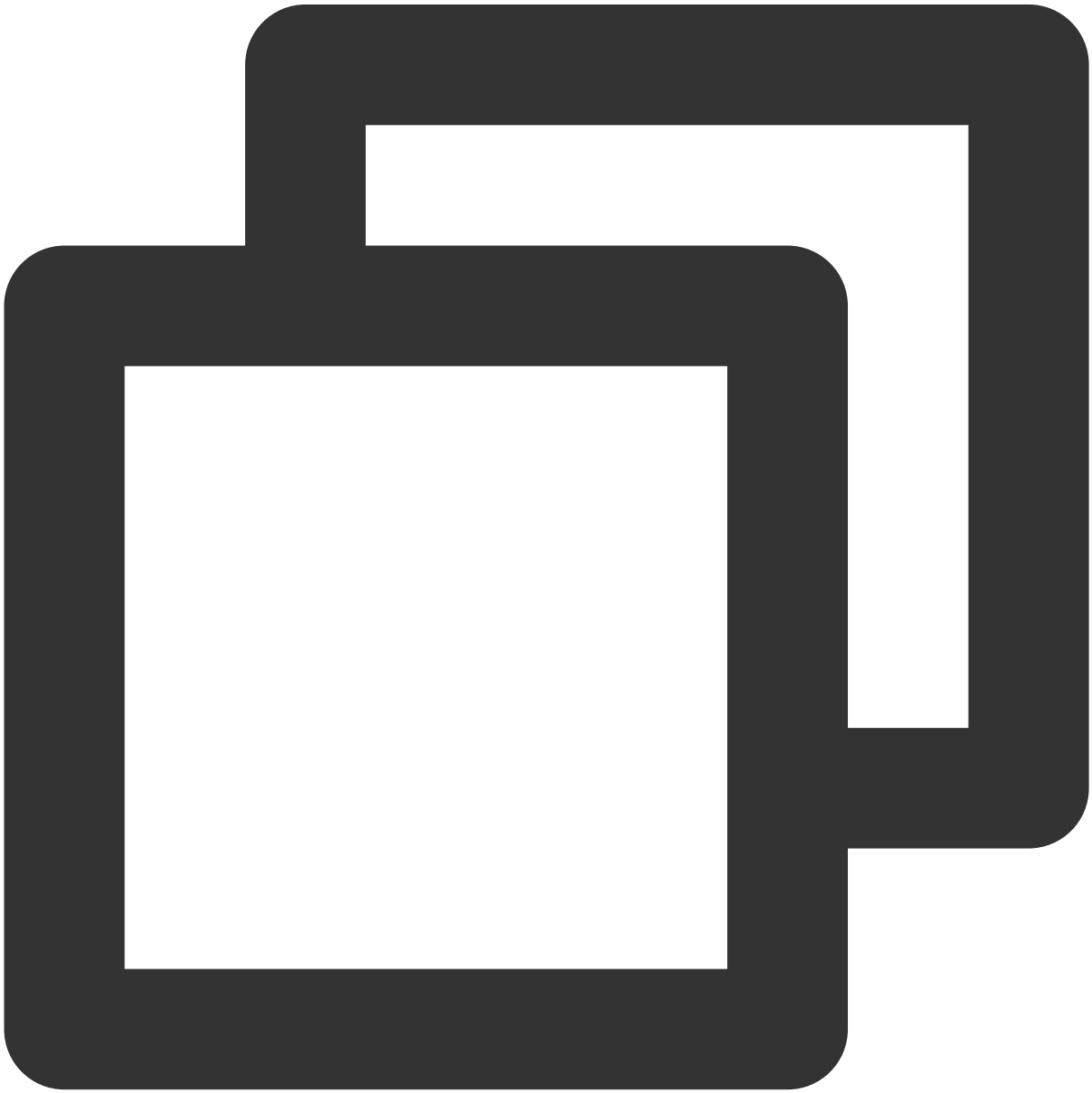


```
@param $table_group_id (必须) 游戏区 ID
@param $table_name (必须) 表名
@param  $params (必须)
```

```
    $params\\["select"\\] （必须） 查询的字段 数组
    $params\\["keys"\\] （必须） 查询目标记录的主键字段(Primary key)
    $params\\["limit"\\] （可选） 返回记录数限制量
    $params\\["offset"\\] （可选） 返回记录数偏移量

public function fieldGet($table_group_id, $table_name, $params)
```

partKeyGet

根据表定义的索引查询返回一条或多条记录，支持指定字段列表返回，支持 limit 和 offset 参数控制返回包的记录数。



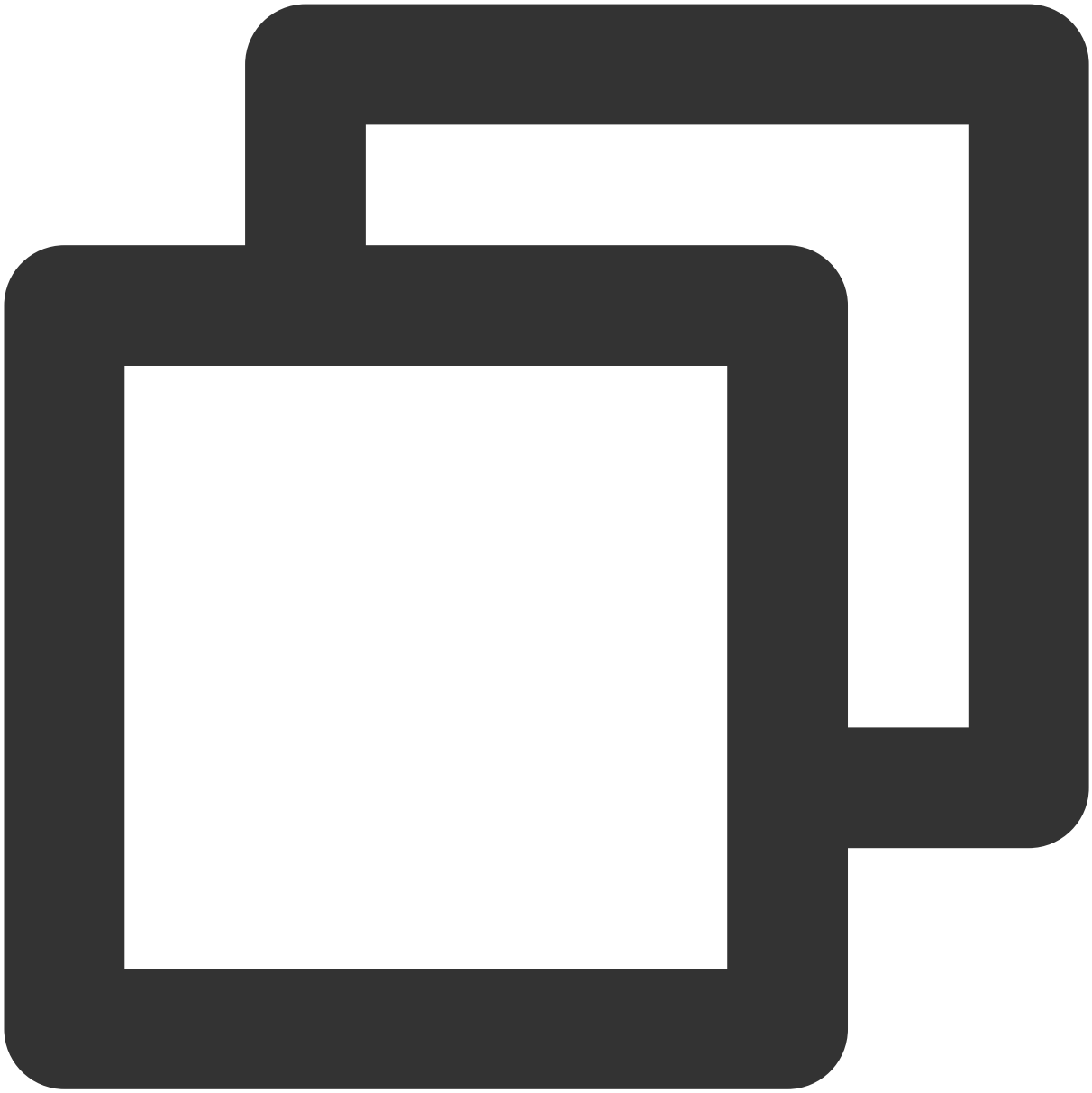
```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param  $params （必须）
    $params\\["select"\\] （可选） 查询的字段 数组
    $params\\["index"\\] （必须） 查询的索引名称
    $params\\["keys"\\] （必须） 查询目标记录的主键字段(Primary key)
    $params\\["limit"\\] （可选） 返回记录数限制量， 默认值-1
    $params\\["offset"\\] （可选） 返回记录数偏移量

public function partKeyGet($table_group_id, $table_name, $params)
```

对于 partkeyGet 接口，1个请求返回的最大包大小为 `256KB` , limit 的设置依赖于单条记录大小。推荐设置策略：

单条记录小于256KB：limit 参考设置为 256KB / [单条记录大小]，如记录大小为10KB，则 limit 推荐设置20 - 25左右。

单条记录大于等于256KB：limit 设置为1，即一次请求只返回一条记录。

关于 limit 和 offset 设置的响应示例如下:

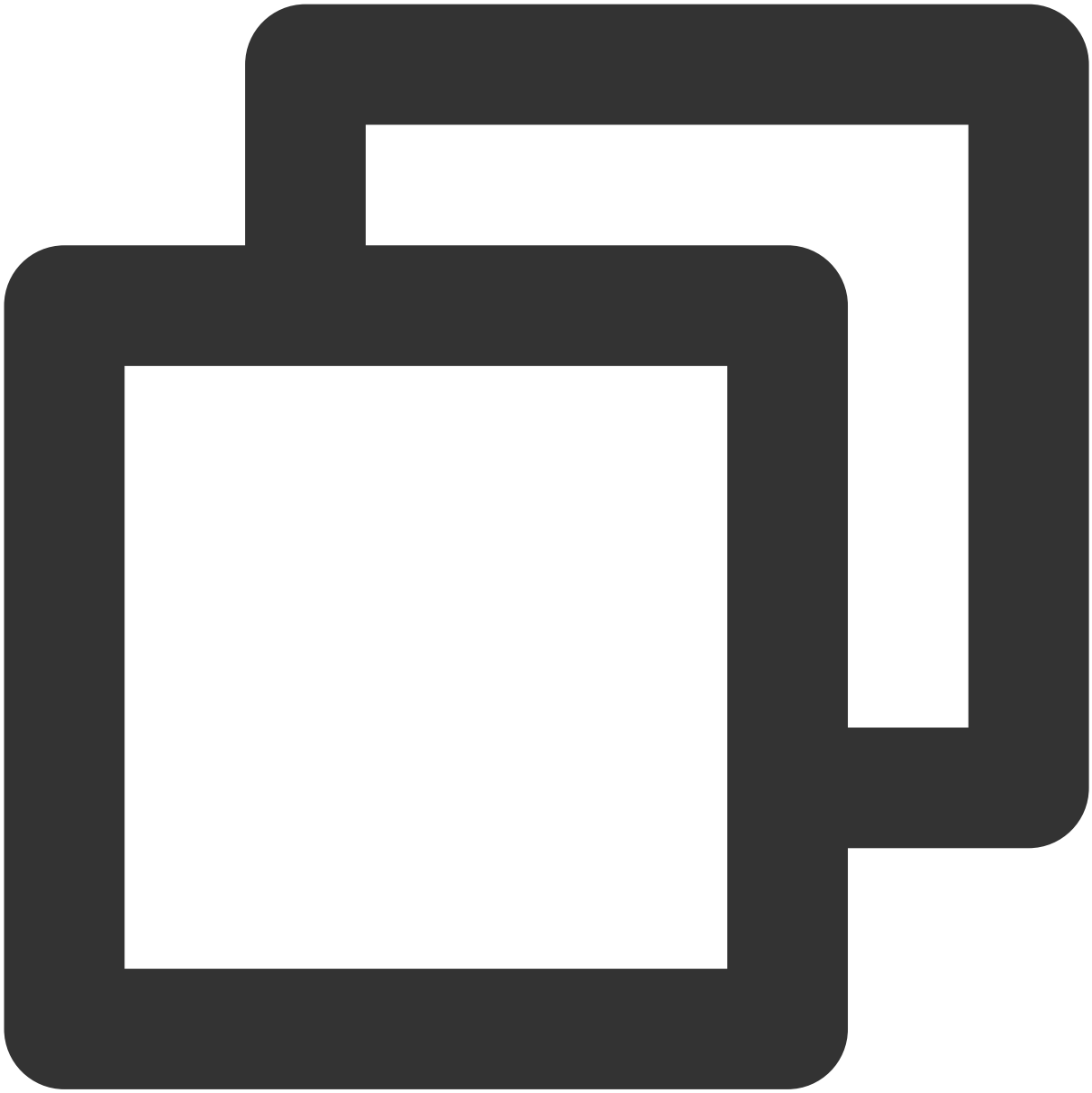


```
#请求设置 limit=-1, offset=0
{"MultiRecords":[{"RecordVersion":1,"Record":{"pay":{"amount":1000,"pay_id":10101},

#请求设置limit=2, offset=0
{"MultiRecords":[{"RecordVersion":1,"Record":{"pay":{"amount":1000,"pay_id":10101},
```

从上面响应包来看，设置了 limit 和 offset 的结果返回的条数和设定的 limit 的大小保持一致。
如果用户想获取全量的索引数据，则可根据响应包中的 `RemainNum` 和 `TotalNum` 这两个标识来判断数据是否获取完全。

set

更新记录或插入记录, 如果记录存在则进行更新，如果记录不存在则进行插入。

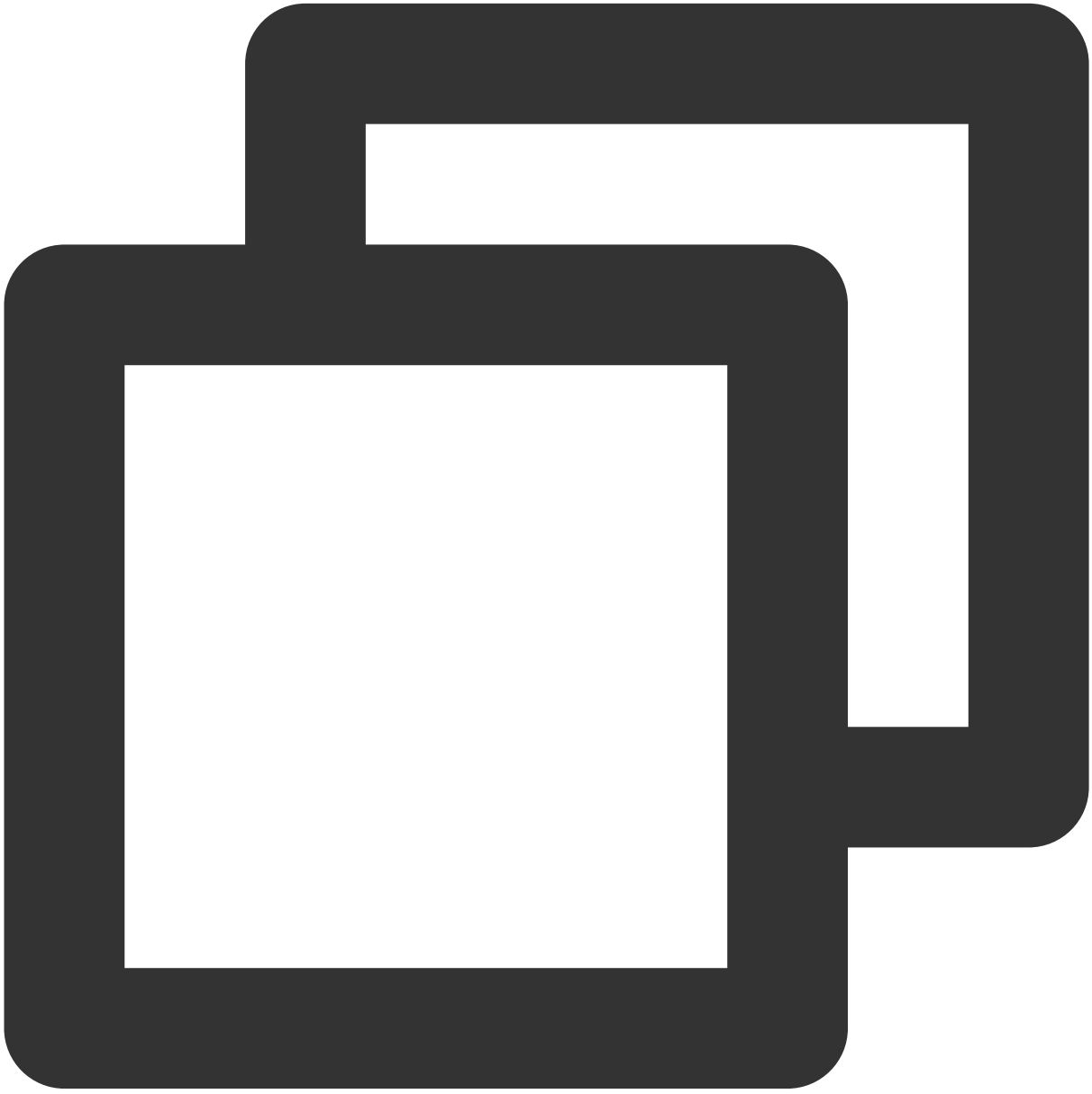


```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param $record （必须） 待设置的目标记录 该操作包含插入和修改的语义
@param $ReturnValues （可选） 校验字符串，服务端返回同样的字符串
@param $resultflag （可选） 默认值1，设置响应包的内容模式，取值范围如下：
            0：应答中仅包含请求成功或失败
            1：应答中包含与请求一致的值
            2：应答中包含被修改的数据的所有字段最新值
```

```
            3: 应答中包含记录被修改前的值

public function set($table_group_id, $table_name, array $record, $ReturnValues = 'T
```

fieldSet

根据主键更新指定字段的值，指定更新字段参考 `$field_path` 。



```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param $record （必须） 待设置的目标记录 该操作包含插入和修改的语义
```

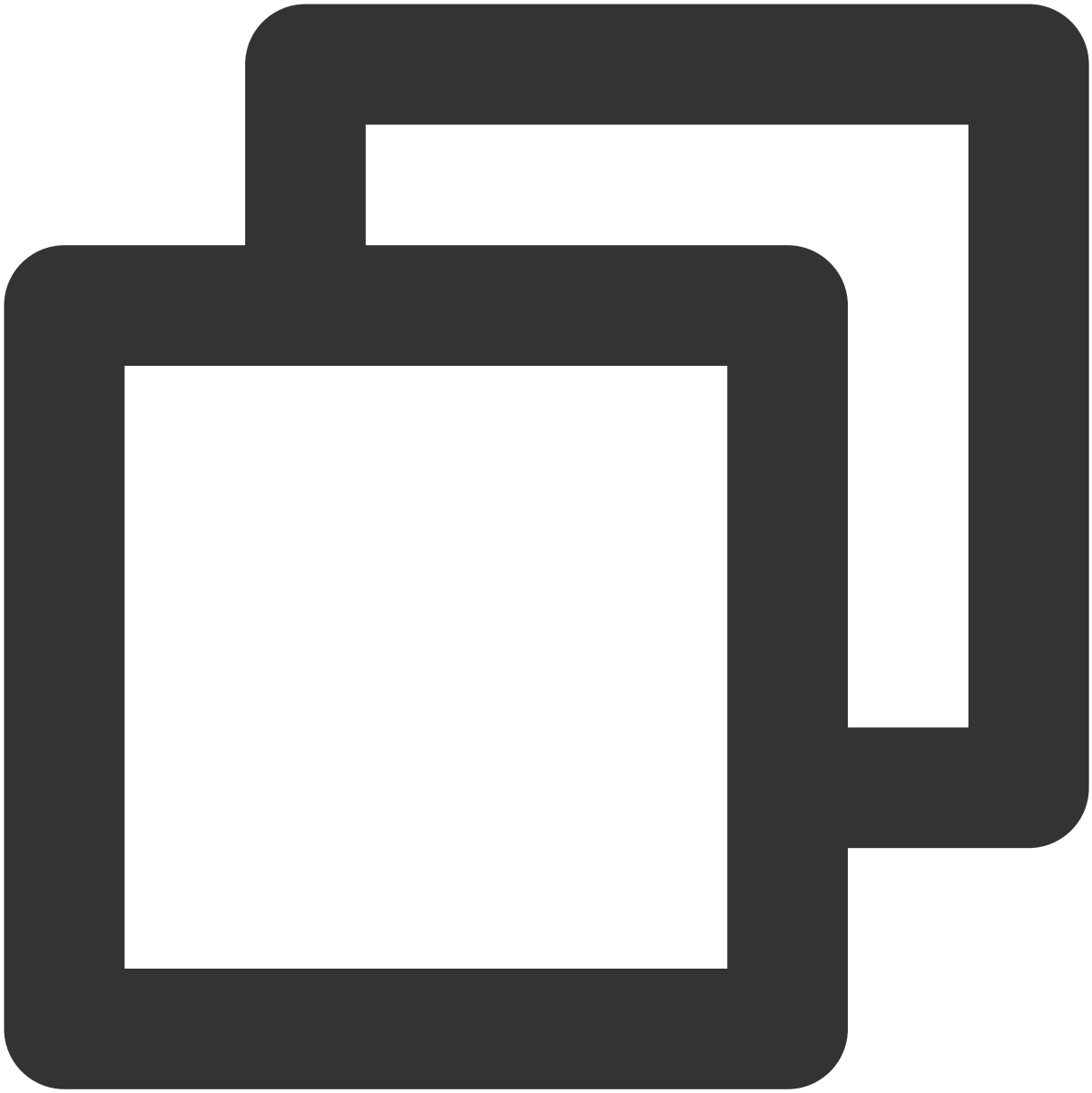
```
@param $field_path （必须） - 待设置的字段名(路径)数组，嵌套字段可以通过点分路径的方式指定
@param $ReturnValues （可选） 校验字符串，服务端返回同样的字符串
@param $resultflag （可选） 默认值1，设置响应包的内容模式，取值范围如下：
            0：应答中仅包含请求成功或失败
            1：应答中包含与请求一致的值
            2：应答中包含被修改的数据的所有字段最新值
            3：应答中包含记录被修改前的值

public function fieldSet($table_group_id, $table_name, array $record, $field_path,
```

fieldInc

根据主键更新指定字段的值（自增或自减），仅针对数值类型字段。如 pay.method 原始值为200，请求值若为50，则最终值为250；请求值若为-50，则最终值为150。

注意：

自减仅针对有符号型字段，如 int32、int64。

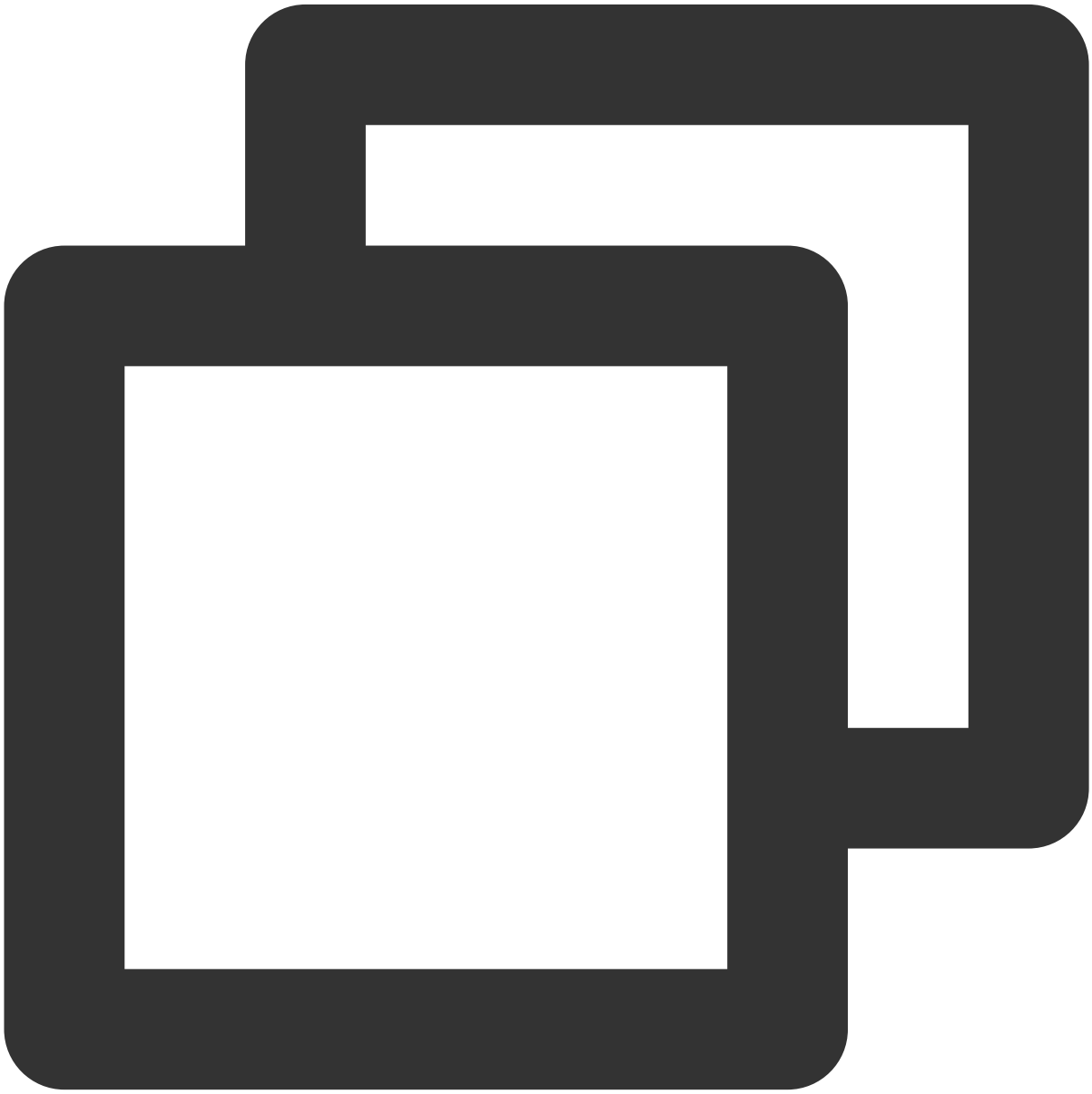


```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param $record （必须） 待自增、自减的记录，必须保证记录中包含主键，待自增、自减的记录必须是整数类
@param $ReturnValues （可选） 校验字符串，服务端返回同样的字符串
@param $resultflag （可选），默认值1，设置响应包的内容模式，取值范围如下：
            0：应答中仅包含请求成功或失败
            1：应答中包含与请求一致的值
            2：应答中包含被修改的数据的所有字段最新值
            3：应答中包含记录被修改前的值
@param $dataVersion （可选） 版本号
@param $dataVersionCheck （可选） 数据版本检查策略，取值范围：
```

```
    1 ：表示版本号一致才能会写入
    2 ：不检测版本号，强制将客户端传入的版本号设置到存储层
    3 ：默认值，不校验版本号，写操作会将存储层数据版本号+1

public function fieldInc($table_group_id, $table_name, array $record, $ReturnValues
```

delete

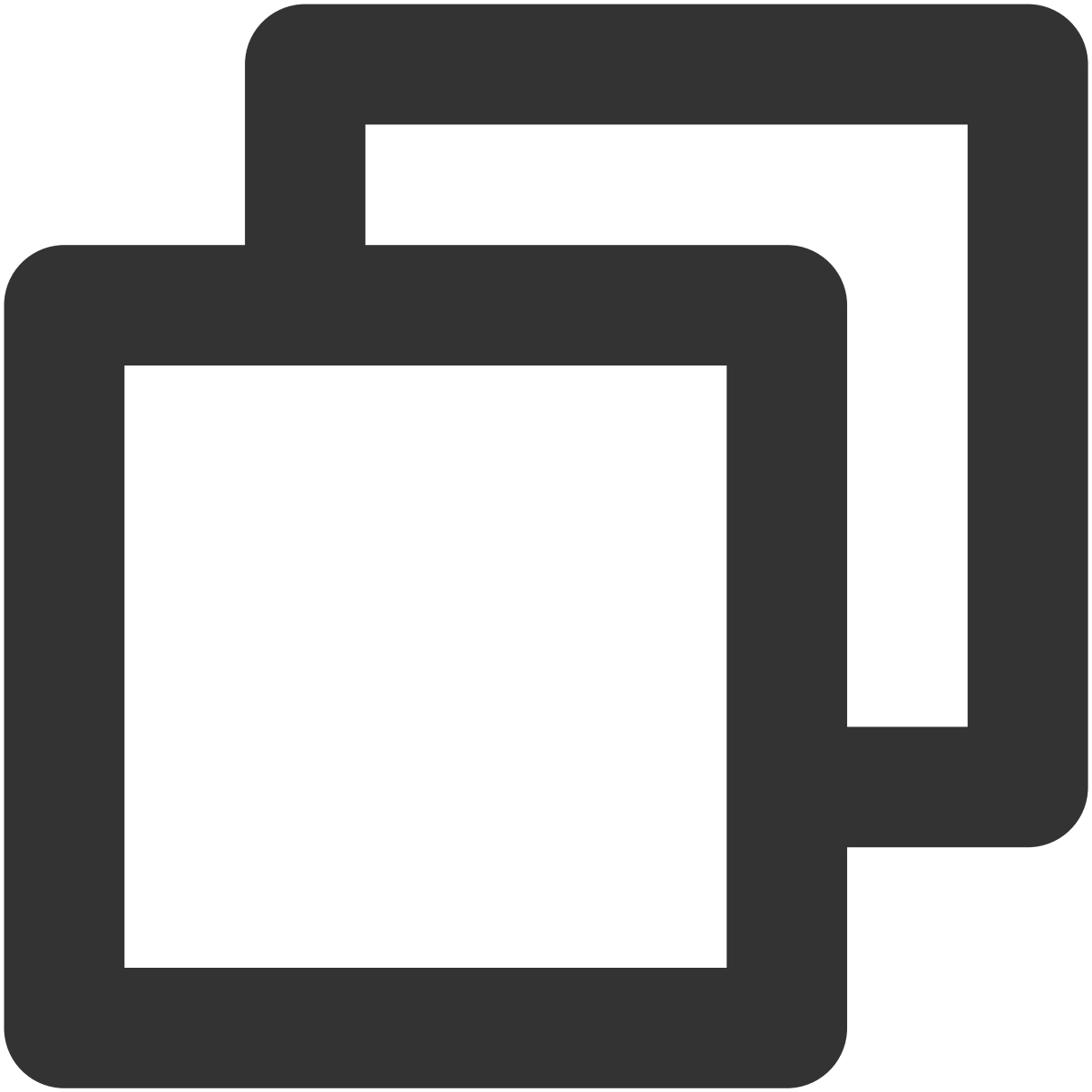
删除指定主键的记录。



```
@param $table_group_id （必须） 游戏区 ID
@param $table_name （必须） 表名
@param $record （必须） 待删除的目标记录，可以只需要设置主键字段
@param  $ReturnValues （可选） 校验字符串，服务端返回同样的字符串
@param $resultflag(可选） 默认值1，设置响应包的内容模式，取值范围如下：
            0：应答中仅包含请求成功或失败
            1：应答中包含与请求一致的值
            2：应答中包含被修改的数据的所有字段最新值
            3：应答中包含记录被修改前的值

public function delete($table_group_id, $table_name, array $record, $ReturnValues =
```

demo.php 使用方法

为方便用户快速体验 TcaplusDB SDK 功能， 目前所有接口已经封闭成函数统一放在 demo.php，共8个函数，可以按需执行对应的函数，只需要注释其它函数即可，执行命令如下：



```
php -f  "demo.php"
```

Python RESTful API 接口说明

最近更新时间：2023-12-18 15:02:26

本 SDK 是基于 RESTful API 封装的一个 Python 语言 SDK，用于进行 TcaplusDB PB 表的增删查改操作。

准备工作

1. 创建 TcaplusDB 表

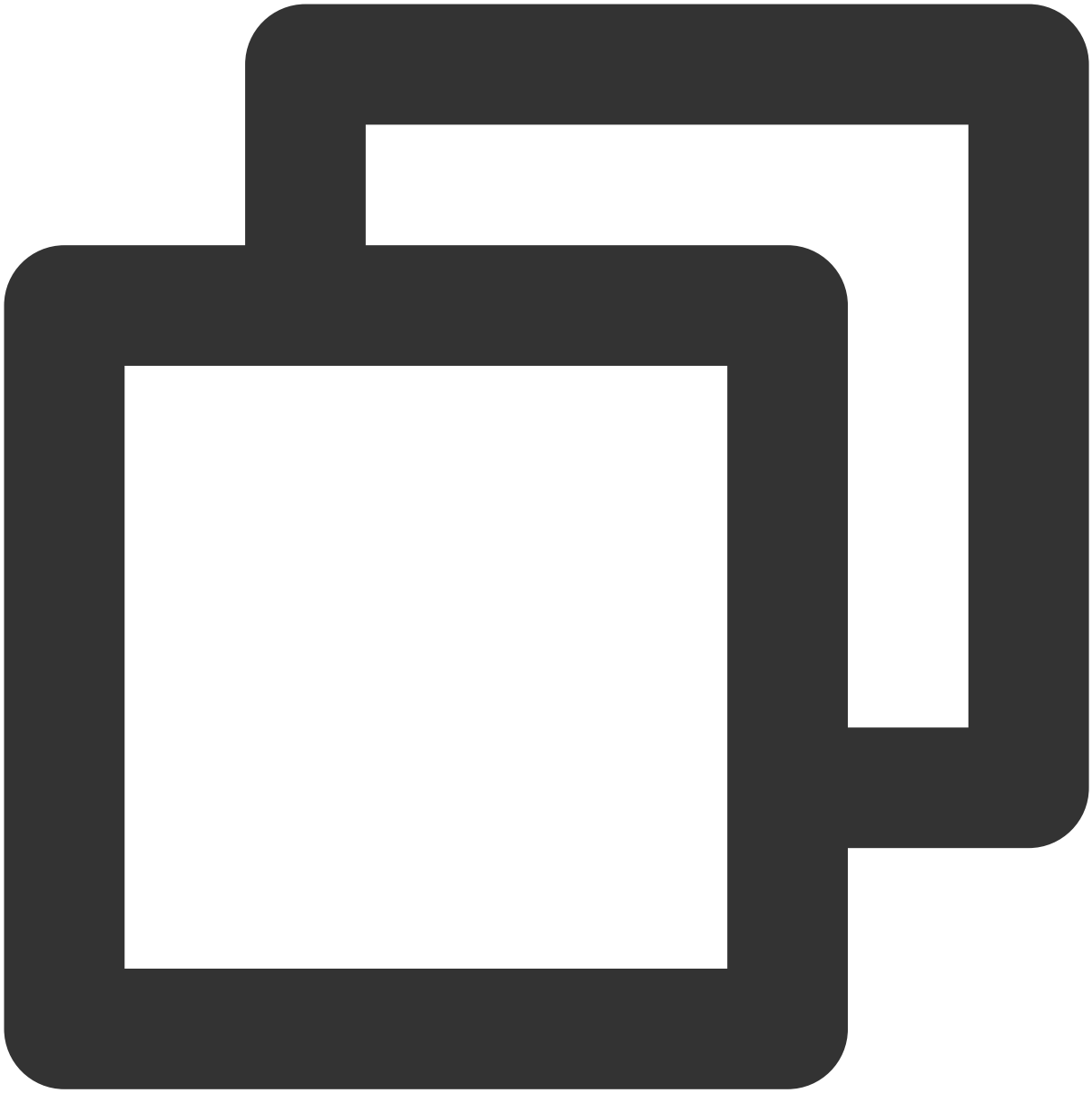
创建 TcaplusDB 示例表，示例表为 `game_players.proto` ，请参见 创建表格。

2. 创建 CVM 实例

创建一台 CVM 实例来运行 SDK 示例程序，配置建议为2核4GB、硬盘50GB，该 CVM 需创建在 TcaplusDB 实例所在 VPC 网络中。
通过 SDK 下载 Python RESTful API SDK 安装包。

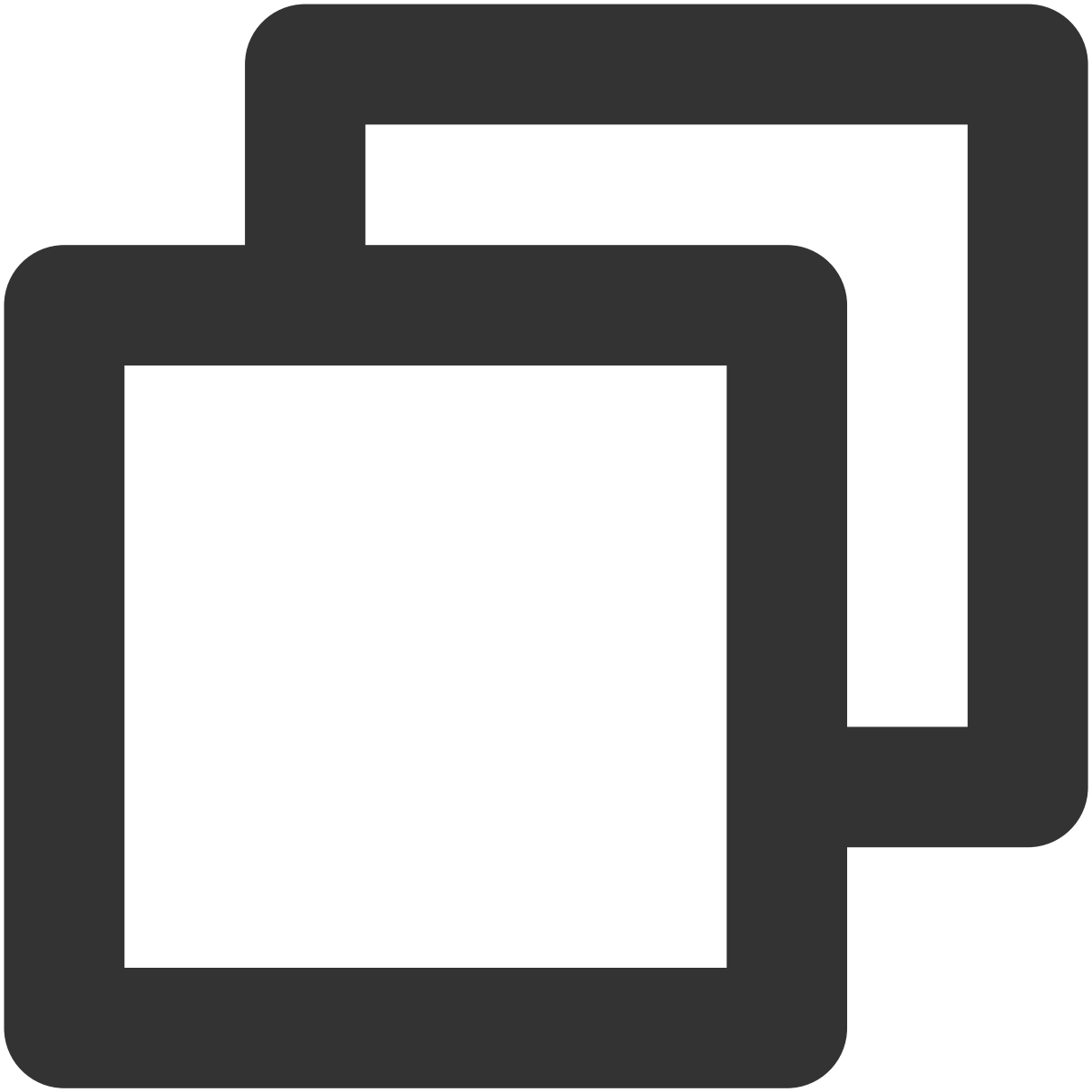
使用步骤

1. 通过指定 endpoint、access_id、access_passwd 参数创建 TcaplusRestClient 的对象 client。
2. 通过 client 对象的 SetTargetTable 方法指定想要访问的目标表。
3. 调用接口函数，发送数据访问请求。



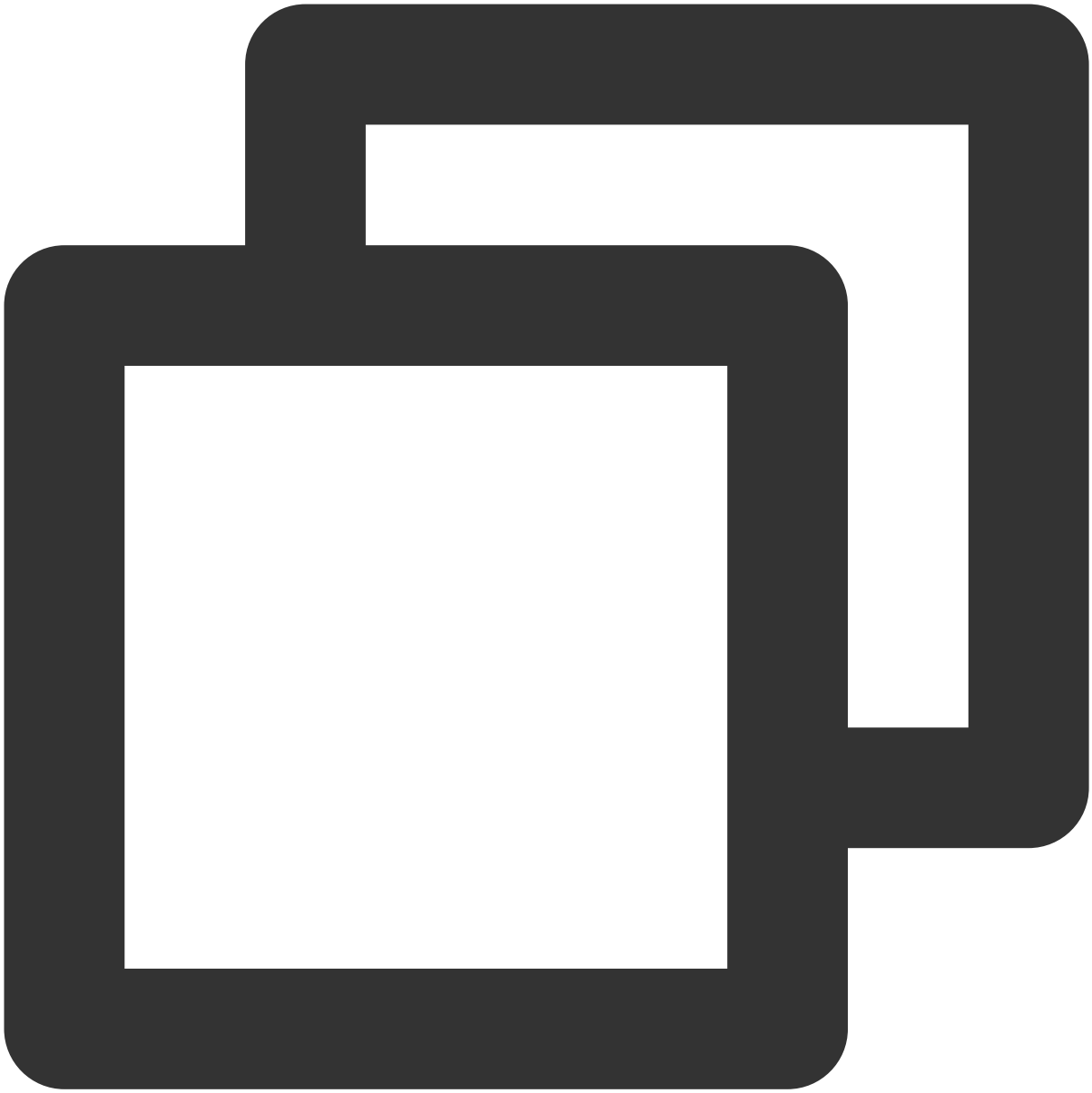
```
#创建 TcaplusRestClient 对象
#endpoint: restful API, such as http://x.x.x.x:80
client = TcaplusRestClient(endpoint, access_id, access_passwd)
#SetTargetTable 指定想要访问的目标表
client.SetTargetTable(table_group_id=1, table_name='game_players')
#发送数据访问请求，以 AddRecord 为例
record = {'player_id': 10805514, 'player_name': 'Calvin', 'player_email': 'calvin@t
                    'login_timestamp': ['2019-12-12 15:00:00'], 'logout_timestamp':
    status, resp = client.AddRecord(record, custom_headers=None, return_values=None
```

4. 用户通过 SetTargetTable 方法指定目标表后，可以通过 GetTargetTable 方法获取目标表。



```
table_group_id, table_name = client.GetTargetTable()
```

也可以通过接口的 table_group_id 和 table_name 参数直接指定目标表，通过这种方式不会改变 SetTargetTable 函数设置的目标表。

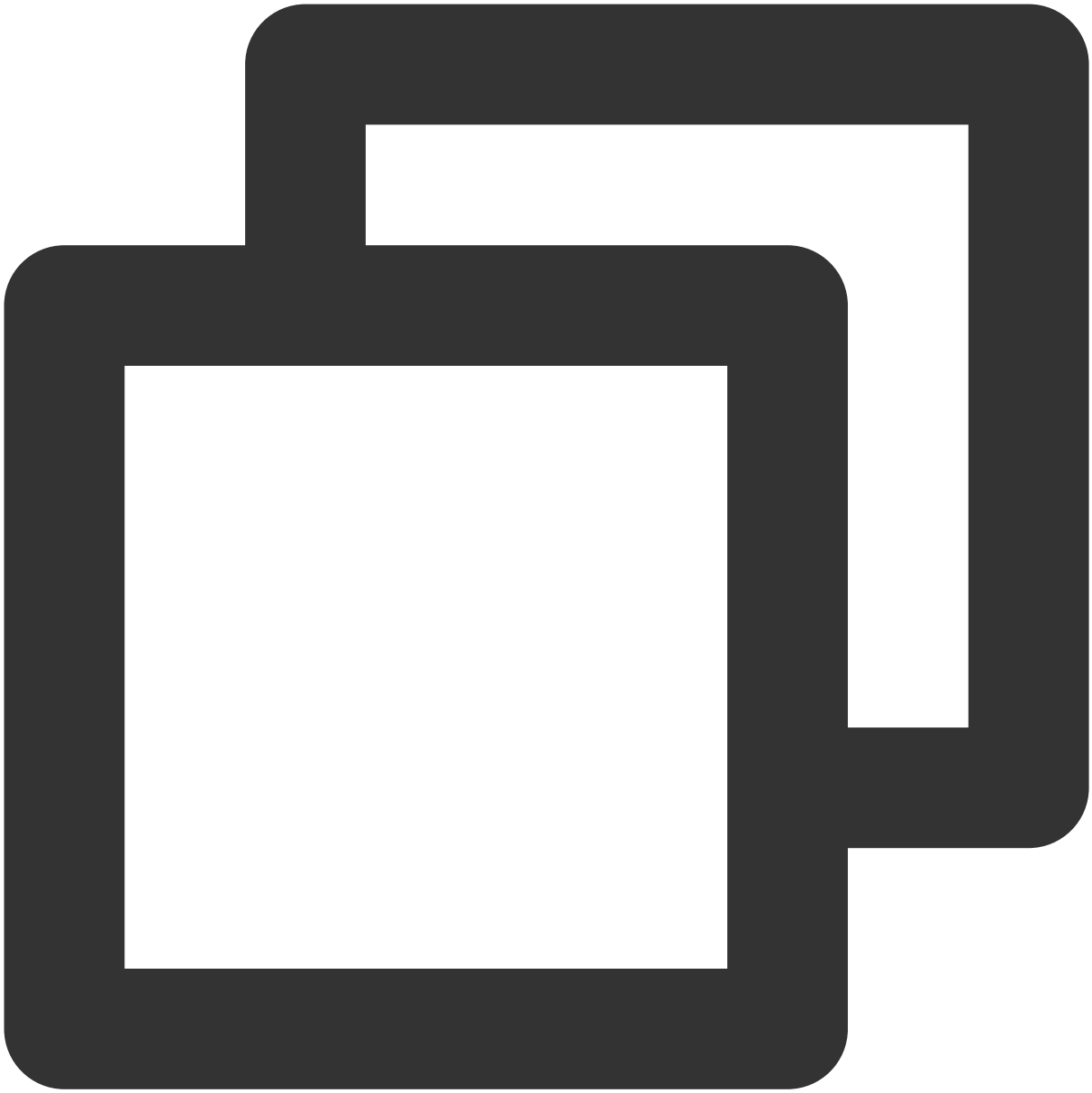


```
status, resp = client.AddRecord(record, custom_headers=None, return_values=None, ta
```

接口列表

GetRecord 记录查询

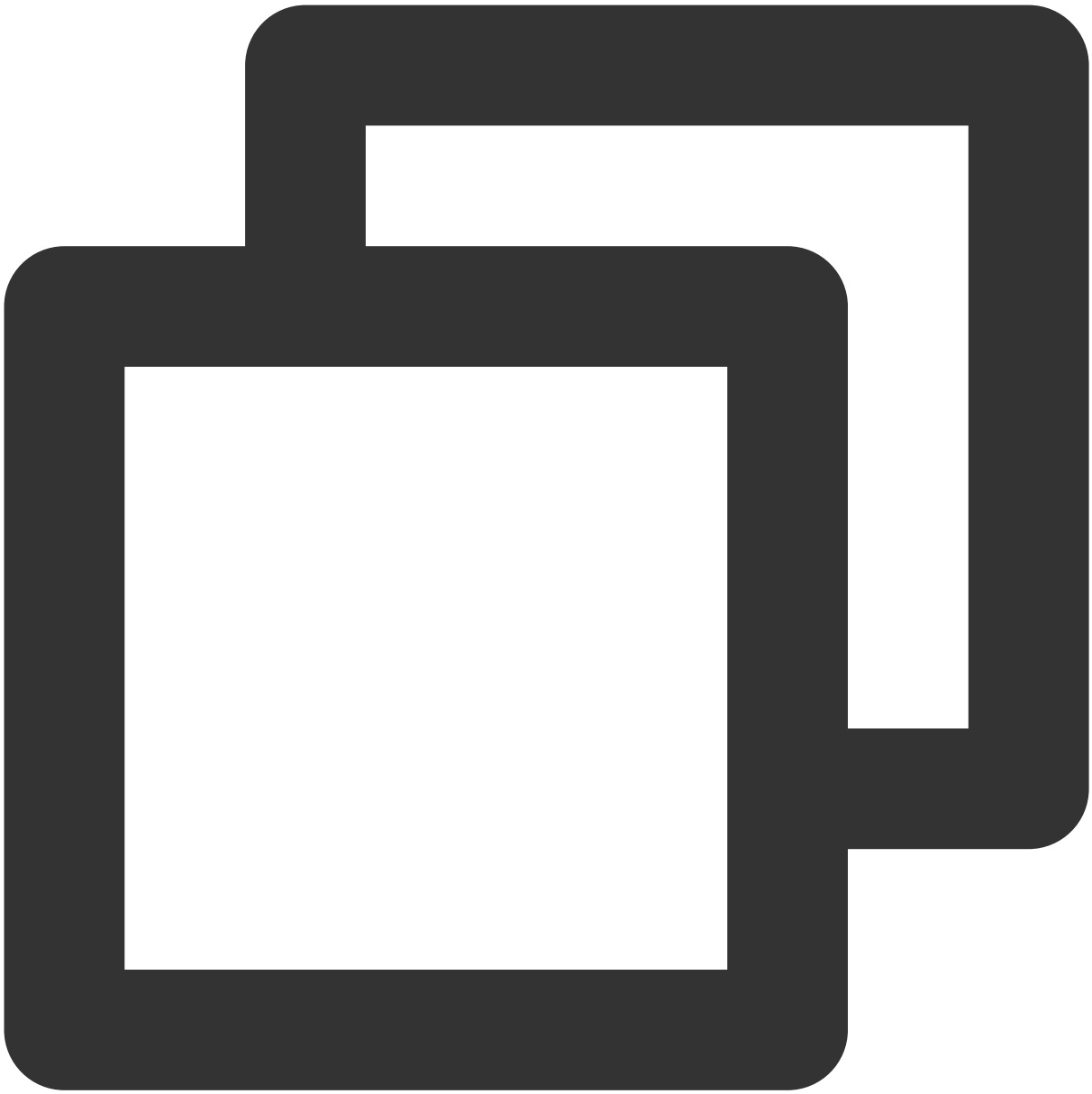
根据主键字段查询表记录，一次返回一条。



```
@param keys （必须） - 查询目标记录的主键字段(Primary key)字典
@param select_fields （可选） - 返回结果中所包含的非主键字段名数组，嵌套字段可以通过点分路径的方
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def GetRecord(self, keys, select_fields=[], custom_headers=None, table_group_id=Non
```

AddRecord 插入记录

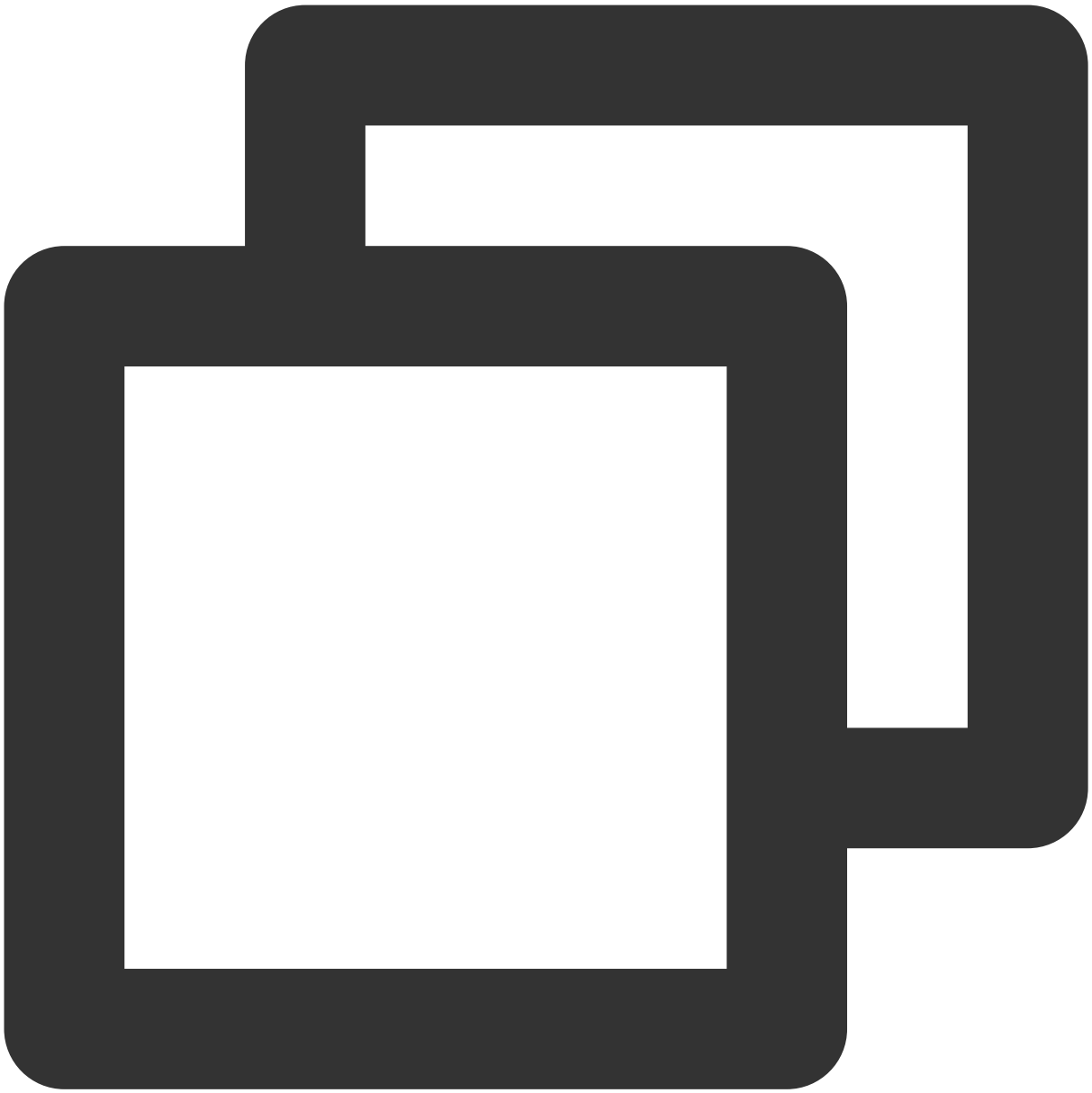
插入一条表记录，如果记录存在则报错。



```
@param record （必须） - 待插入的目标记录 json 对象
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def AddRecord(self, record, custom_headers=None, return_values=None, table_group_id
```

SetRecord 设置记录

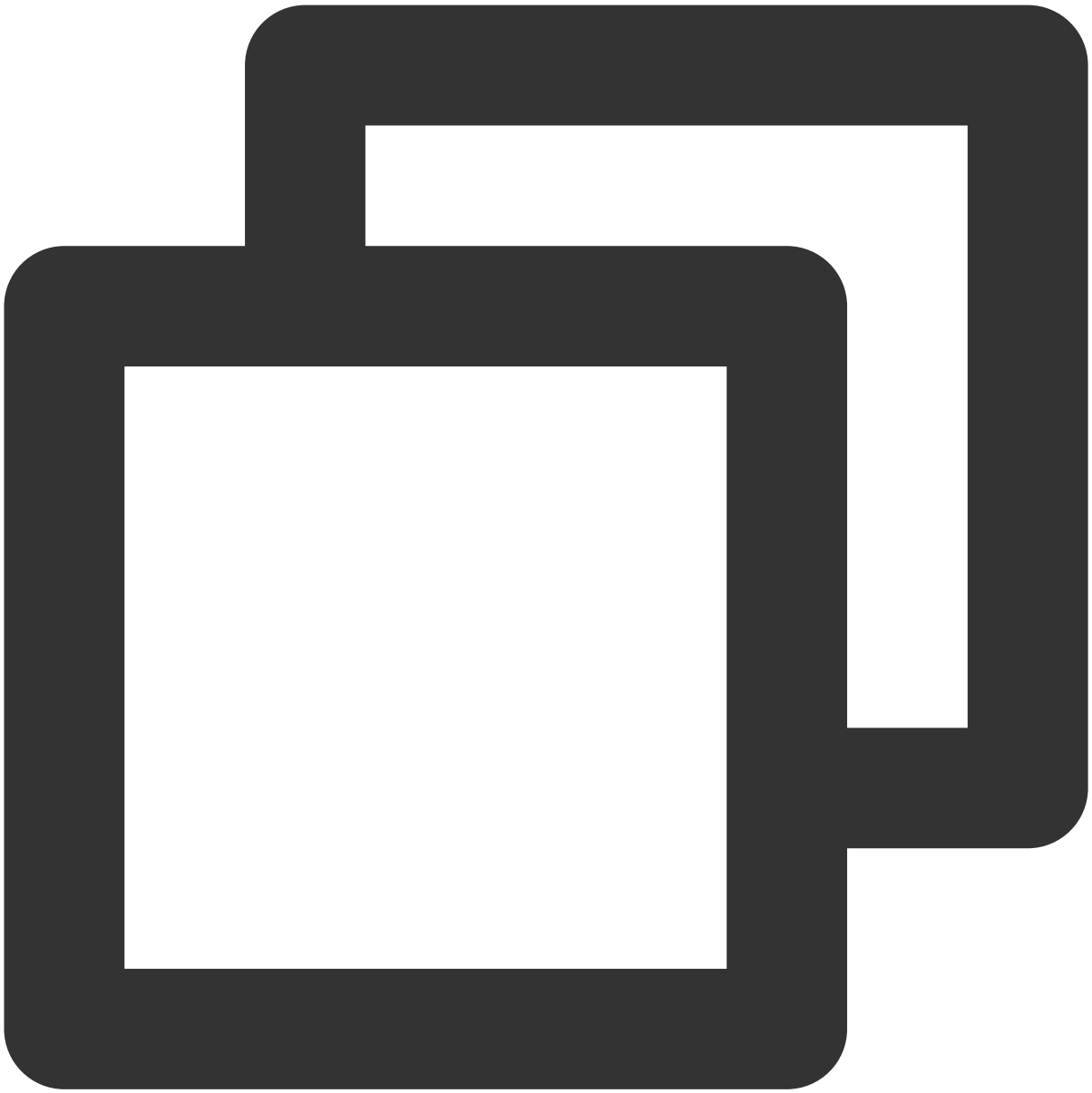
依据主键更新或插入一条记录，如果记录已存在则更新记录相关值，如果记录不存在则插入新的记录。



```
@param record （必须） - 待设置的目标记录 json 对象，该操作包含插入和修改的语义
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def SetRecord(self, record, custom_headers=None, return_values=None, table_group_id
```

DeleteRecord 删除记录

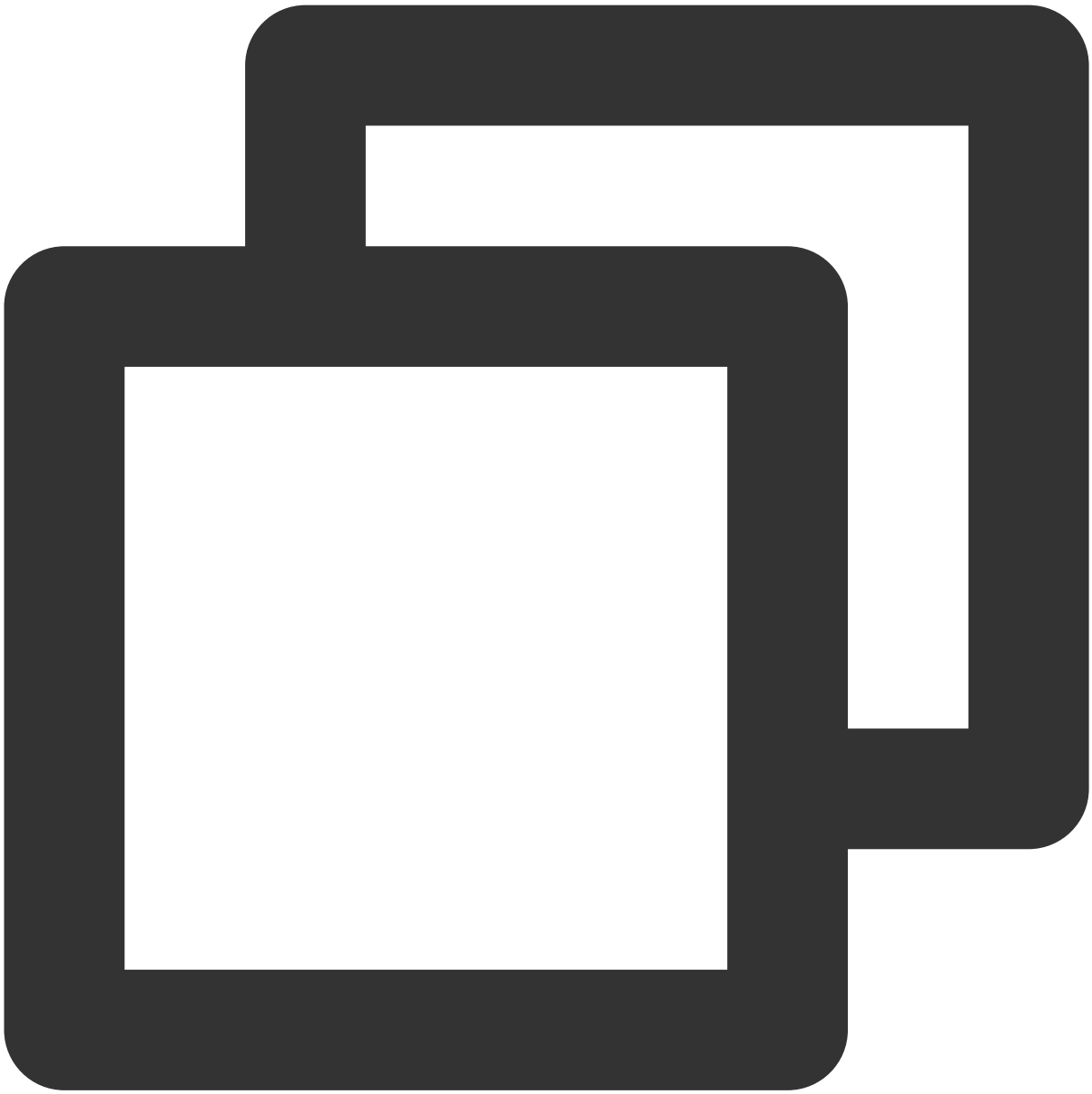
根据主键删除一条记录。



```
@param record （必须） - 待删除的目标记录 json 对象，可以只需要设置主键字段
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def DeleteRecord(self, record, custom_headers=None, return_values=None, table_group
```

FieldGetRecord 指定字段查询

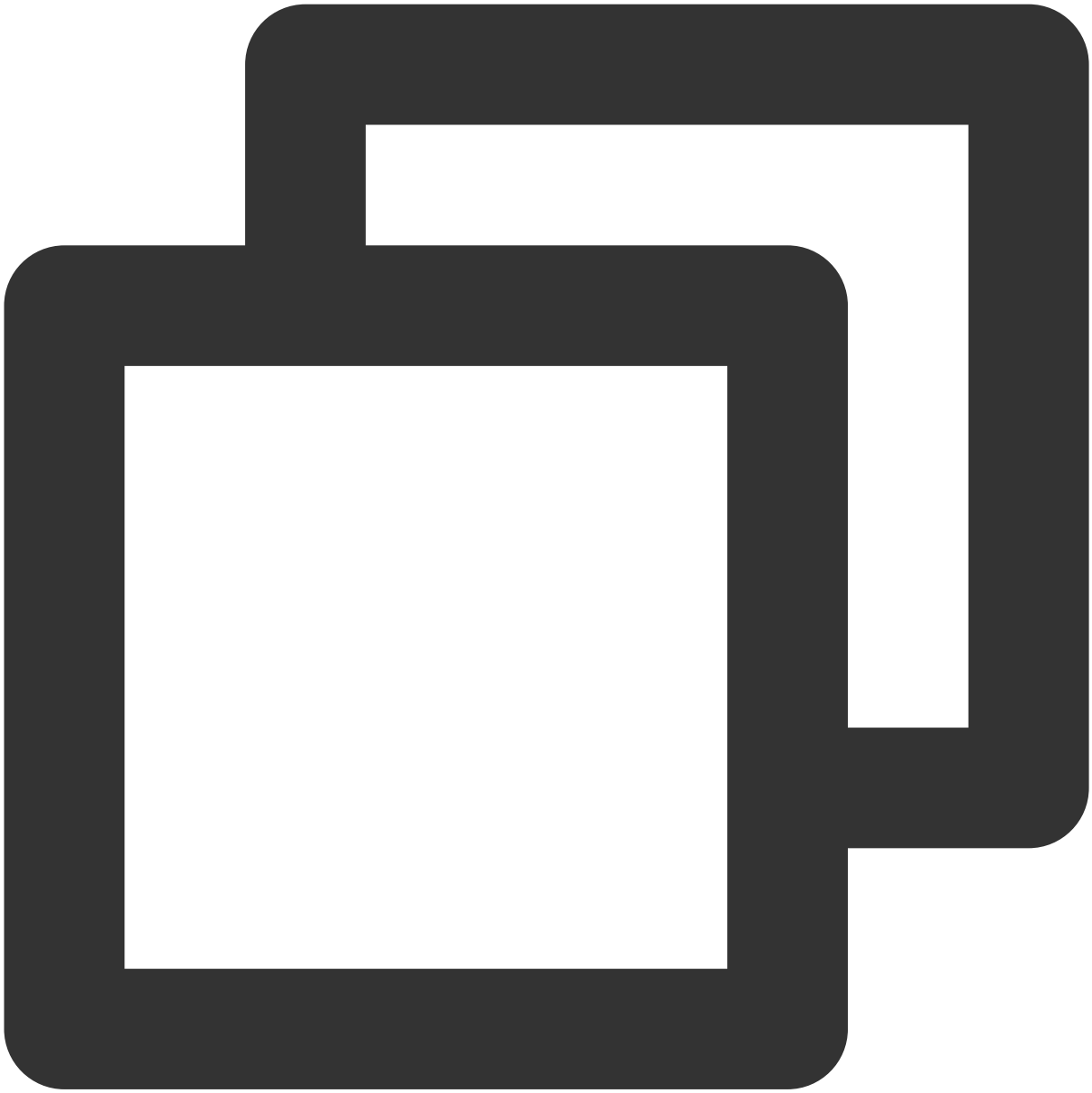
根据主键获取部分字段值，指定需要返回的字段列表（必须为非主键字段），嵌套字段以点分方式表示，如：pay.amount。



```
@param keys （必须） - 查询目标记录的主键字段(Primary key)字典
@param select_fields （必须） - 返回结果中所包含的非主键字段名数组，嵌套字段可以通过点分路径的方
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表table_name

def FieldGetRecord(self, keys, select_fields, custom_headers=None, table_group_id=N
```

FieldSetRecord 指定字段设置

根据主键更新部分字段的值，指定需要更新的字段列表（必须为非主键字段），嵌套字段以点分方式表示，如：pay.amount。



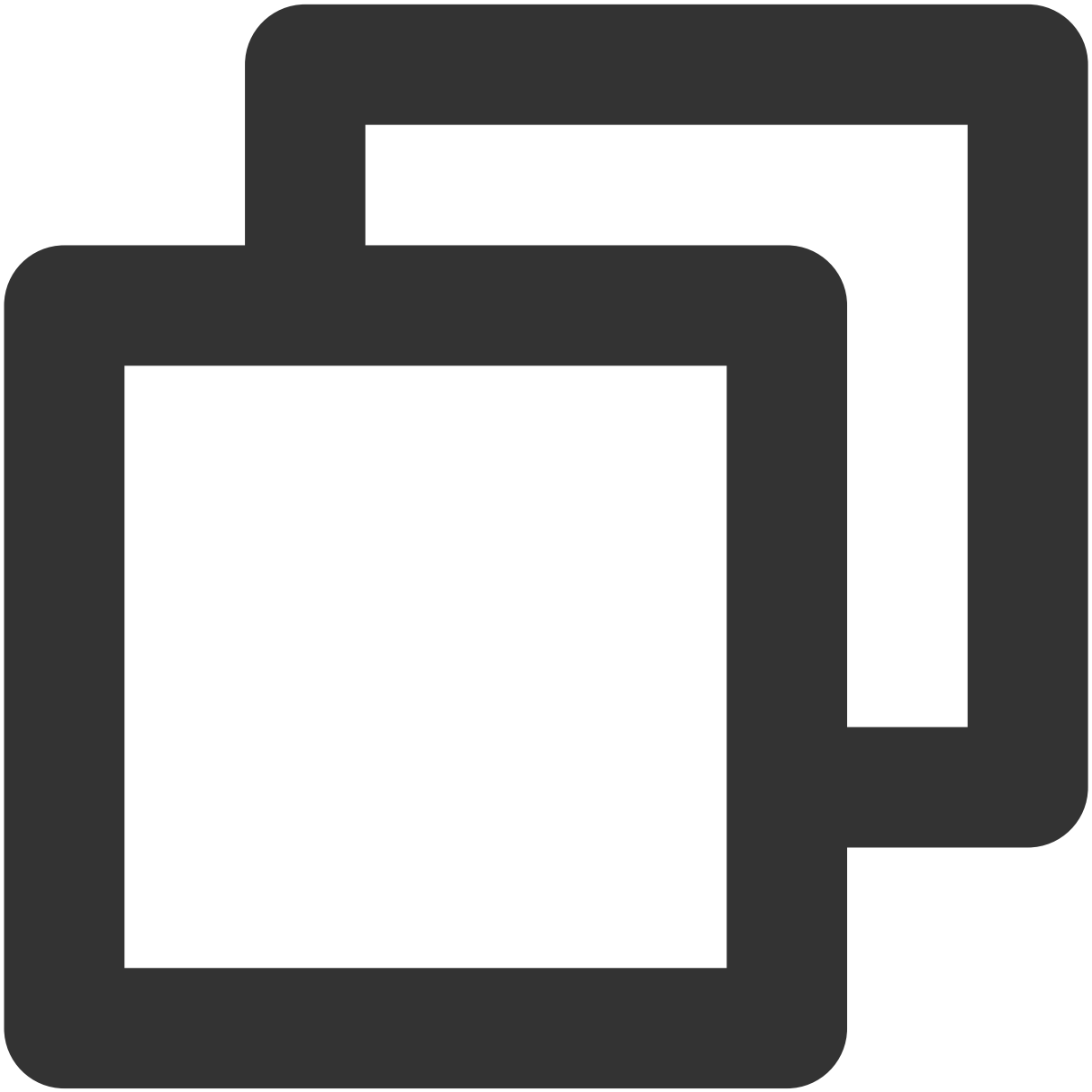
```
@param record （必须） - 待设置的目标记录 json 对象，该操作包含插入和修改的语义
@param field_path （必须） - 待设置的字段名(路径)数组，嵌套字段可以通过点分路径的方式指定
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def FieldSetRecord(self, record, field_path, custom_headers=None, return_values=Non
```

FieldIncRecord 指定字段自增/自减

根据主键更新数值类型字段的值，自增或自减。如 pay.method 原始值为200，请求值为50，则最终 pay.method 值为250，如果请求值为负数-50, 则最终 pay.method 值为150。

注意：

这里如果是自减，那么请求传入的自减字段的类型必须为有符号数值类型，如 int32、int64。

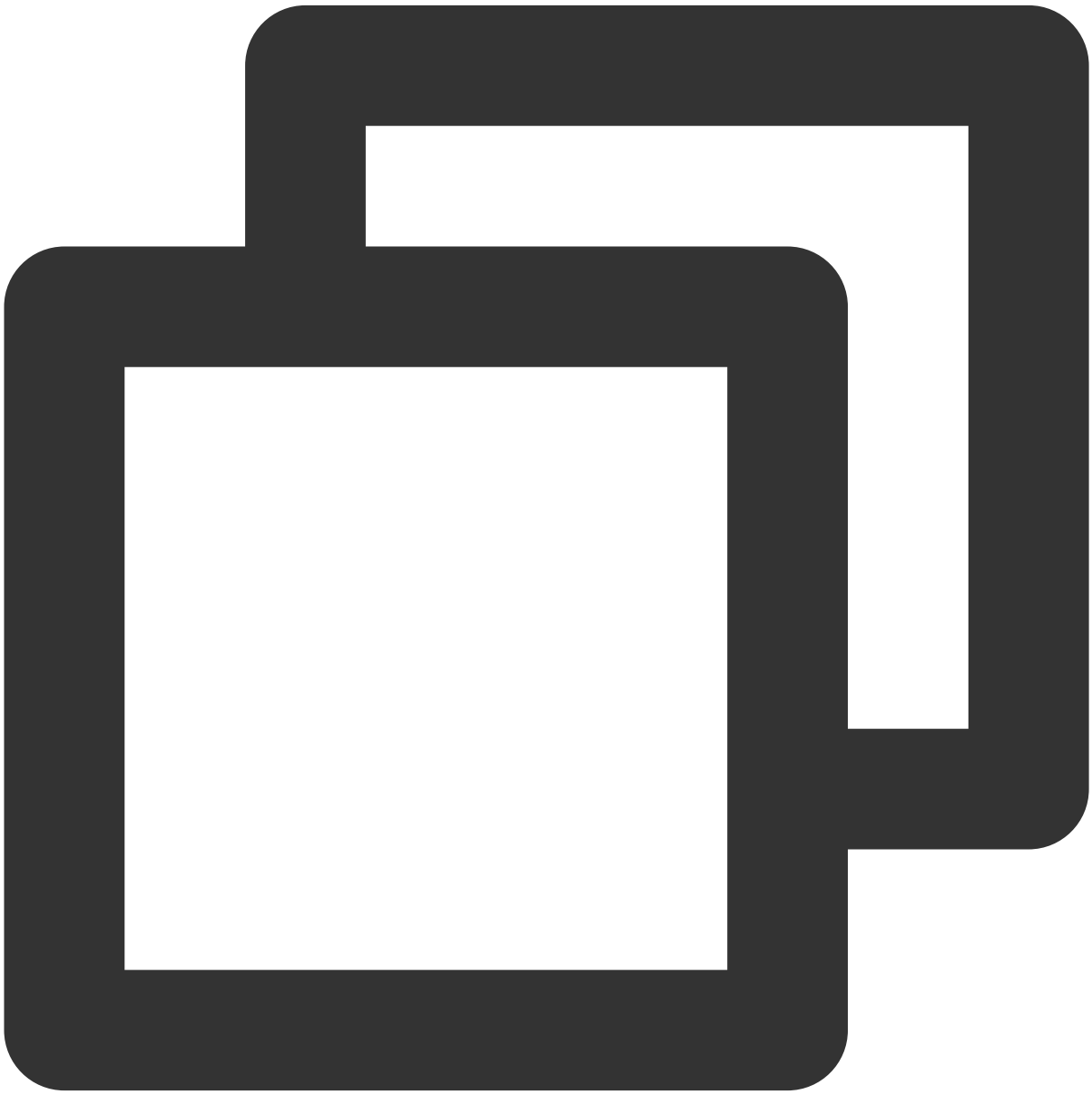


```
@param record （必须） - 待自增、自减的记录，必须保证记录中包含主键，待自增、自减的记录必须是整数类
@param custom_headers （可选） - 用户需要指定的 http header
```

```
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name

def FieldIncRecord(self, record, custom_headers=None, return_values=None, table_gro
```

PartkeyGetRecord 索引查询

根据表定义的索引进行数据查询，返回一条或多条索引键匹配的记录。支持指定字段列表返回。支持指定 limit 和 offset 来控制每个请求返回的包大小。



```
@param index_keys （必须） - 查询目标记录的索引键字典
```

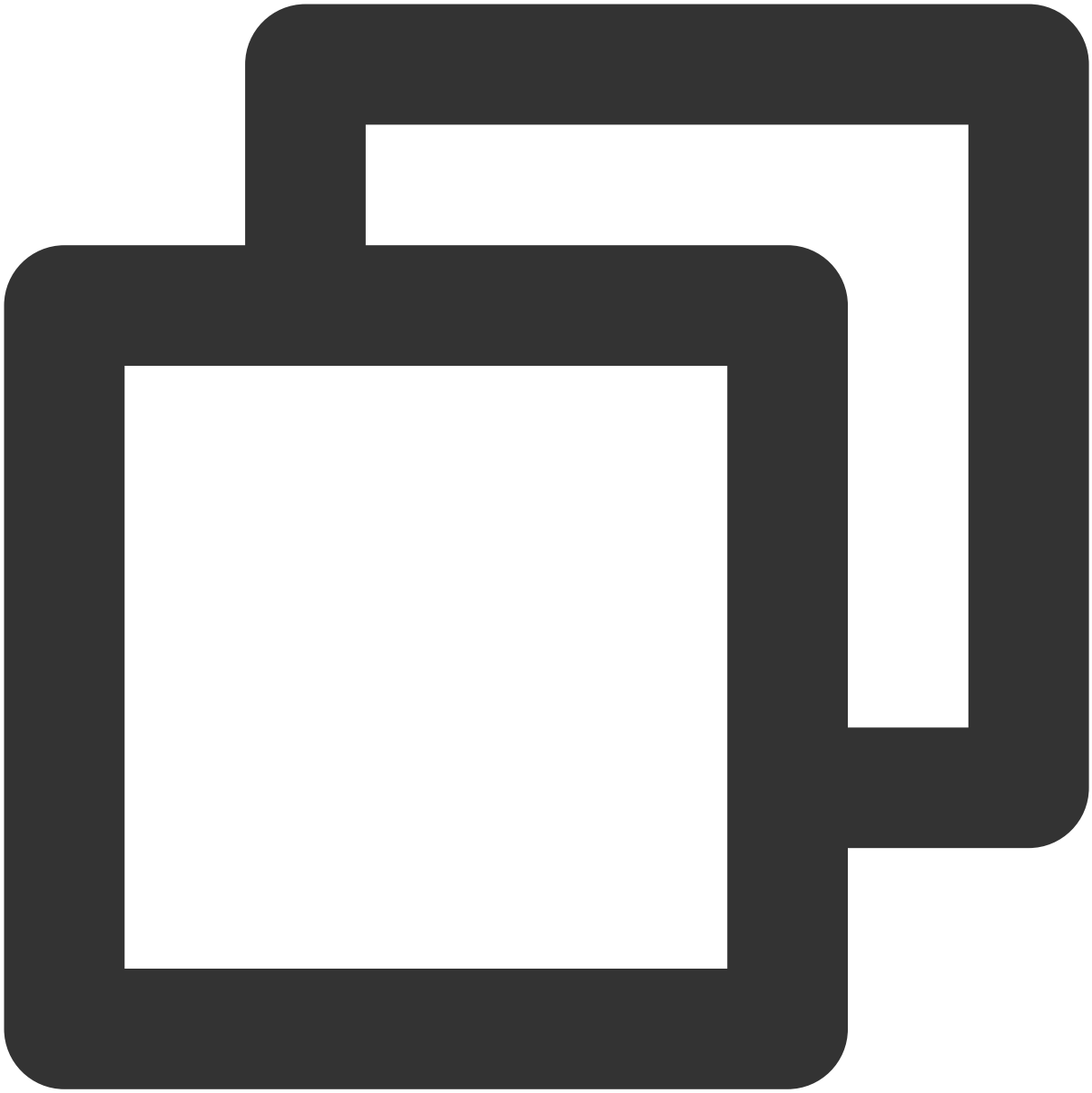
```
@param index_name （必须） - 查询的索引名称
@param select_fields （可选） - 返回结果中所包含的非主键字段名数组，嵌套字段可以通过点分路径的方
@param custom_headers （可选） - 用户需要指定的 http header
@param table_group_id （可选） - 表所在集群表格组 id
@param table_name （可选） - 目标表 table_name
@param limit （可选） -限制返回的记录条数
@param offset （可选） -设定返回记录数的起始偏移量

def PartkeyGetRecord(self, index_keys, index_name, select_fields=[], custom_headers
```

对于该接口，1个请求返回的最大包大小为 `256KB` , limit 的设置依赖于单条记录大小。推荐设置策略：

单条记录小于256KB：limit 参考设置为 256KB / [单条记录大小]，如记录大小为10KB，则 limit 推荐设置20 - 25左右。

单条记录大于等于256KB：limit 设置为1，即一次请求只返回一条记录。

下面是关于设置 limit 和 offset 以及不设置的响应包情况：



```
#索引键情况
{'player_id': 39775502, 'player_name': 'Sara'}

#请求不设置 limit 和 offset，响应包如下
{u'ErrorCode': 0, u'ErrorMsg': u'Succeed', u'MultiRecords': [{u'RecordVersion': 1,

#请求设置 limit 和 offset，其中 limit 设置为2，响应包如下
{u'ErrorCode': 0, u'ErrorMsg': u'Succeed', u'MultiRecords': [{u'RecordVersion': 1,
```

从上面响应包来看，设置了 limit 和 offset 的结果返回条数和设定 limit 大小保持一致。
如果用户想获取全量的索引数据，则可根据响应包中的 `RemainNum` 和 `TotalNum` 这两个标识来判断数据是否获取完全。

RESTful API 各语言示例下载

最近更新时间：2023-06-15 17:03:23

各语言 RestFul 封装 SDK 下载

语言	安装包名	下载
GO	tcaplusdb-restapi-go-sdk.zip	下载
Java	tcaplusdb-restapi-java-sdk.zip	下载
PHP	tcaplusdb-restapi-php-sdk.zip	下载
Python 2.x	tcaplusdb-restapi-python-sdk-2.x.zip	下载
Python 3.x	tcaplusdb-restapi-python-sdk-3.x.zip	下载