前端性能监控

接入指南

产品文档

腾讯云

文档目录

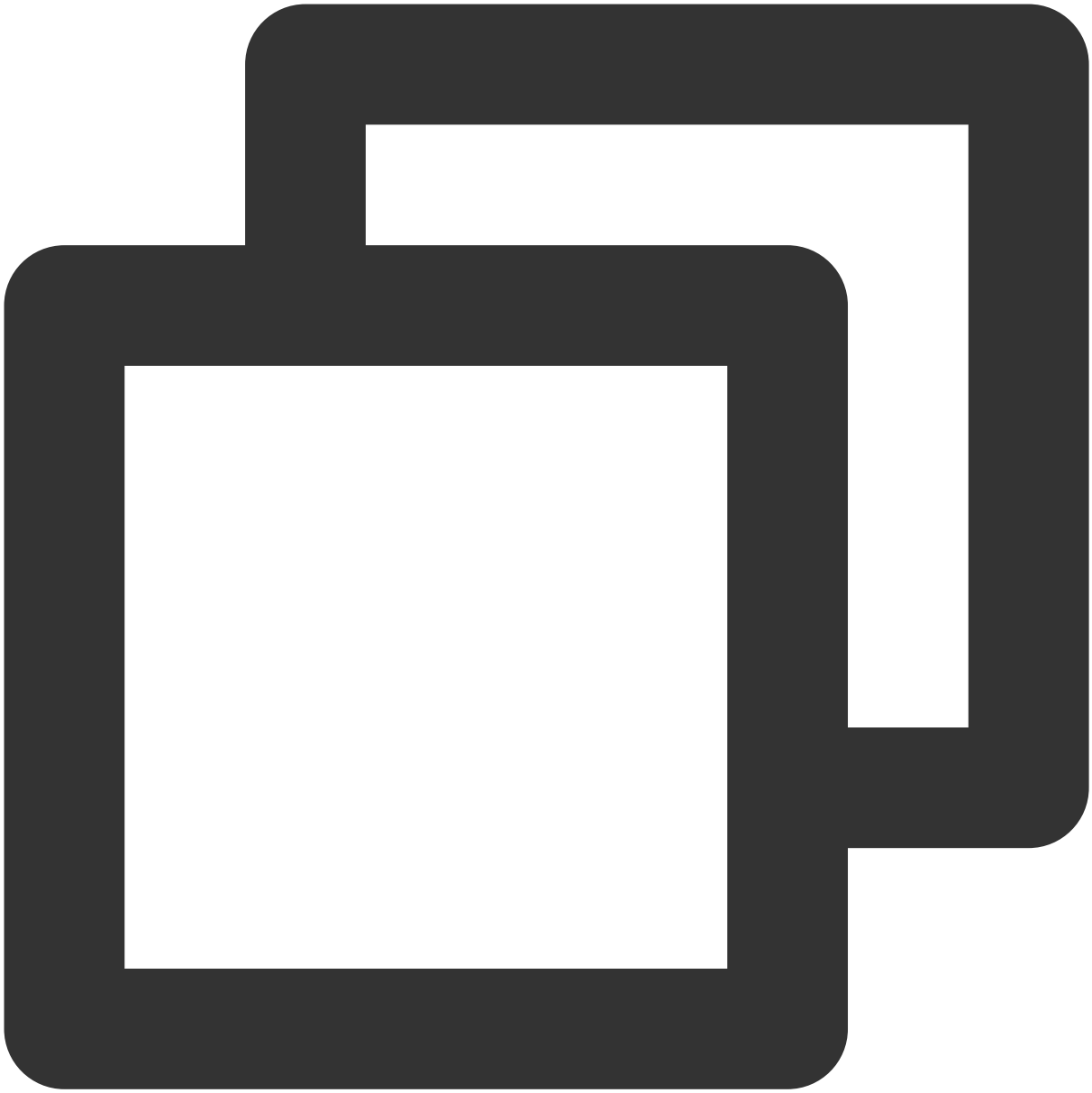
接入指南
Web 应用场景
安装和初始化

最近更新时间：2024-01-22 19:39:30

支持 CDN 和 NPM 两种方式安装 SDK 。

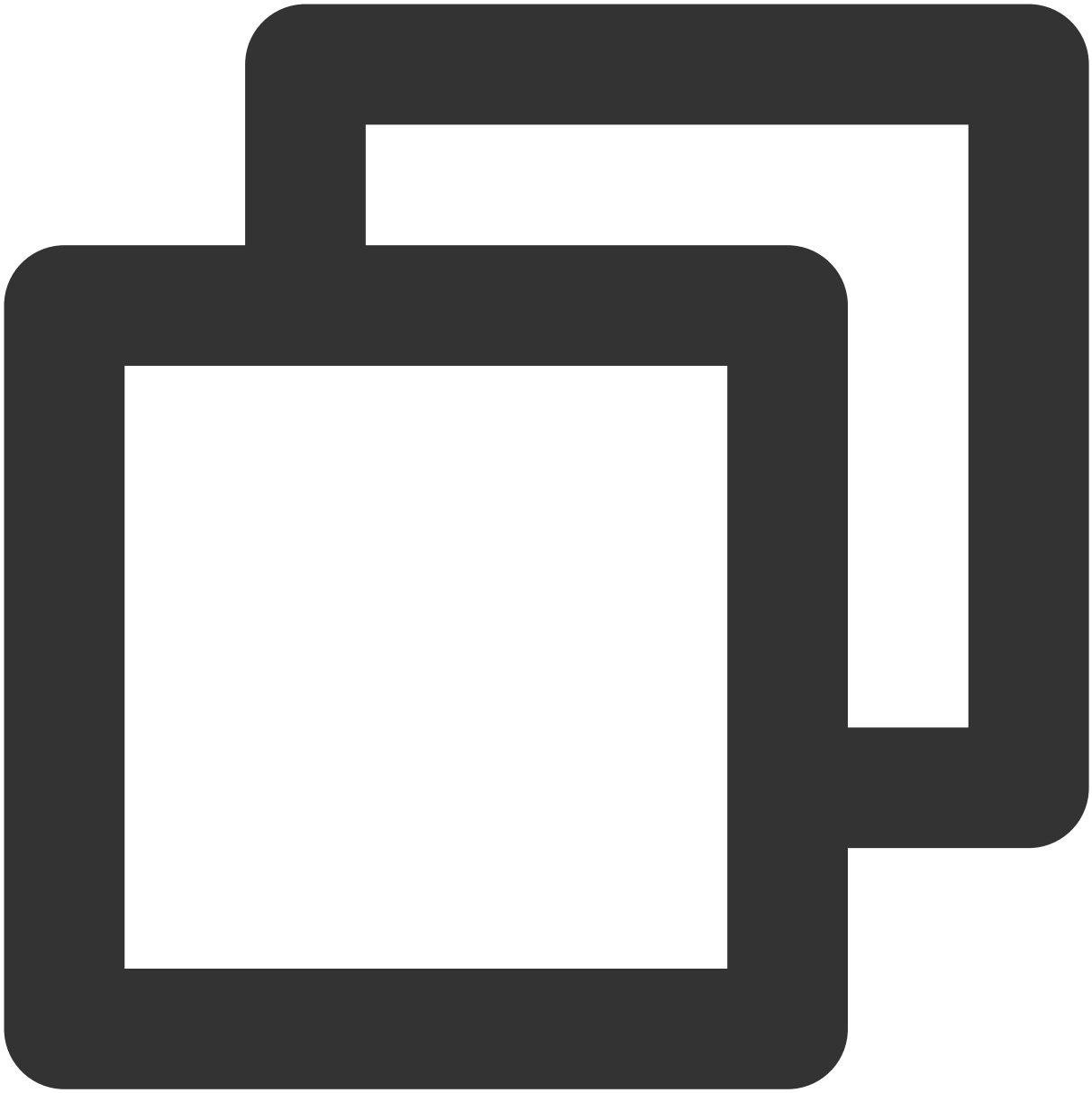
以 CDN 方式安装并初始化 SDK

1. 在 HTML 页面的 `<head></head>` 标签中引入下列代码。



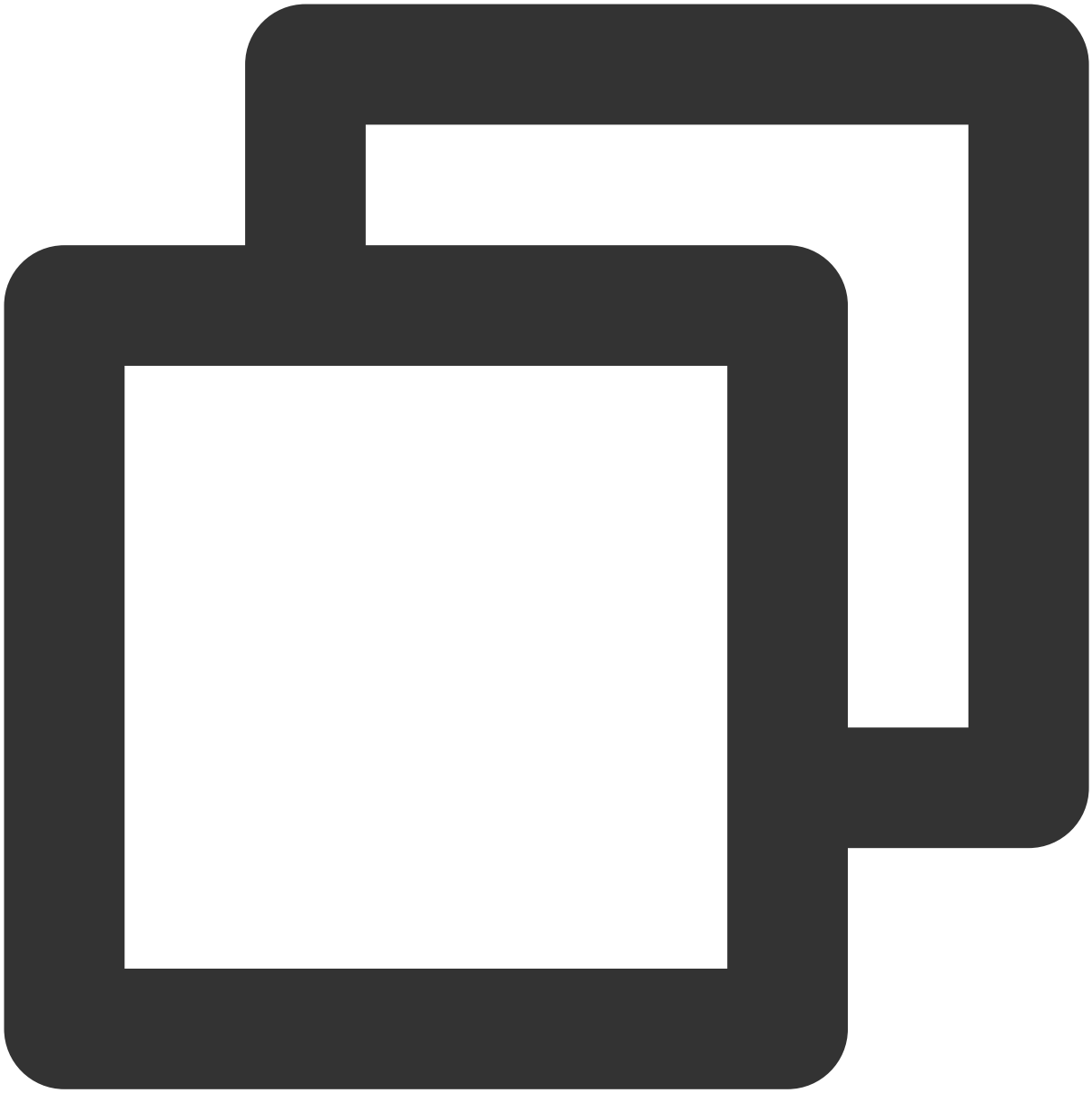
```
<script src="https://cdn-go.cn/aegis/aegis-sdk/latest/aegis.min.js"></script>
```

该 cdn 使用 “h3-Q050” 协议，默认 cache-control 为 max-age=666，如果需要修改 cache-control，可以添加参数 max_age，例如



```
<script src="https://cdn-go.cn/aegis/aegis-sdk/latest/aegis.min.js?max_age=3600"></
```

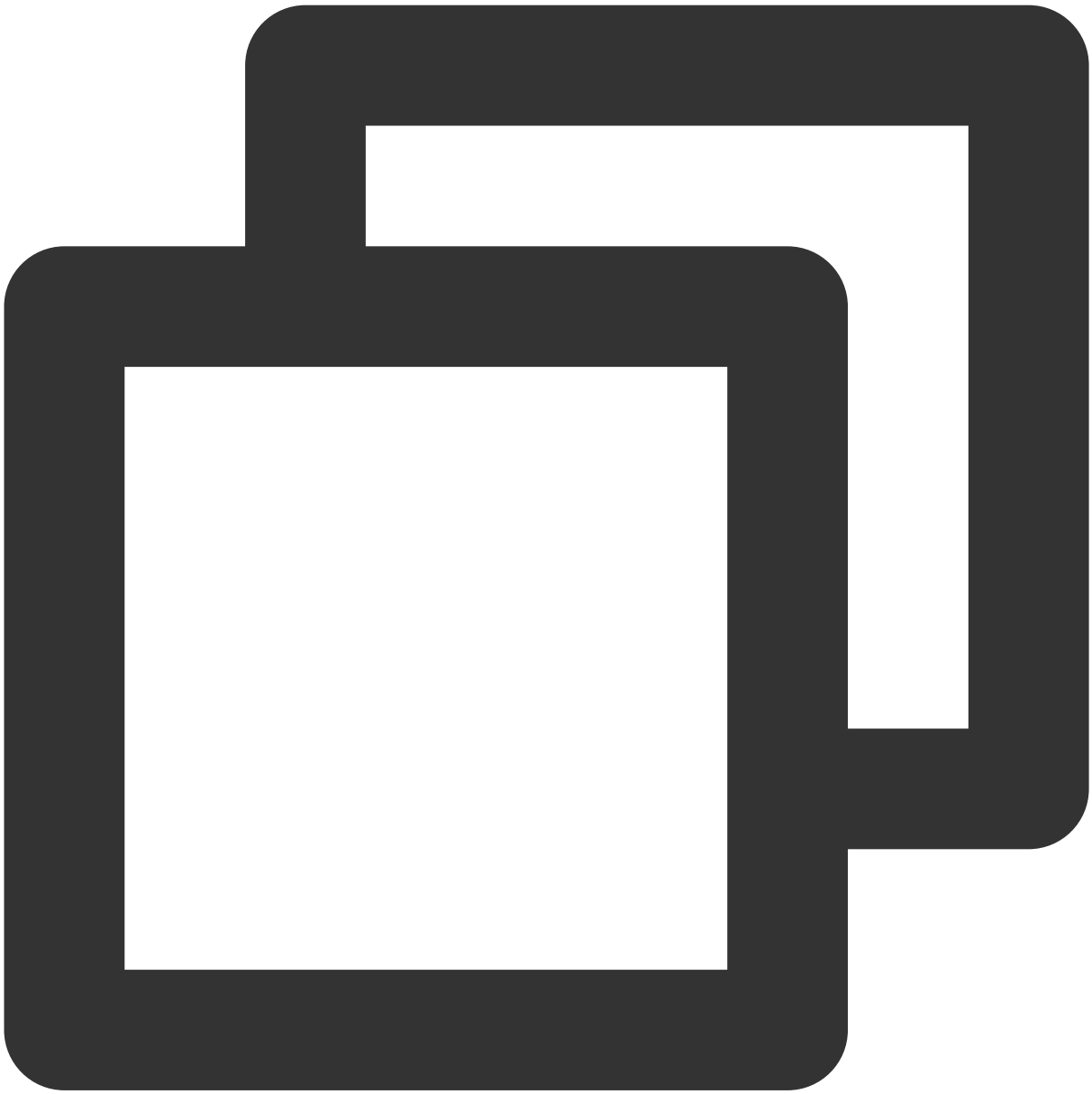
2. 参见下列步骤新建一个 Aegis 实例，传入相应的配置，初始化 SDK 。



```
const aegis = new Aegis({
    id: 'pGUVFTCZyewhxxxxxx', // 应用ID，即上报ID
    uin: 'xxx', // 用户唯一 ID（可选）
    reportApiSpeed: true, // 接口测速
    reportAssetSpeed: true, // 静态资源测速
    spa: true // spa 应用页面跳转的时候开启 pv 计算
});
```

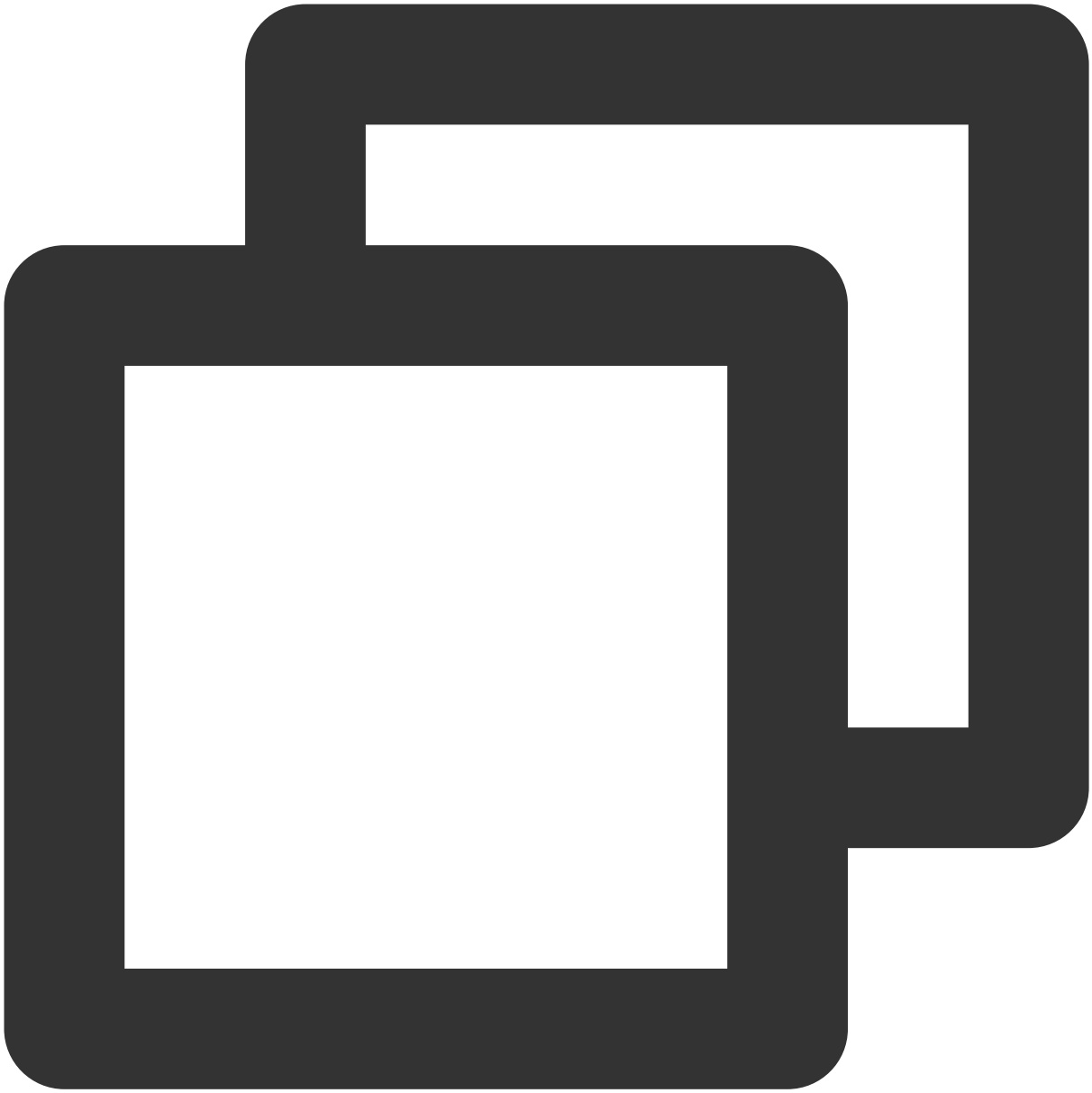
以 NPM 方式安装并初始化 SDK

1. 执行下列命令，在 npm 仓库安装 aegis-web-sdk。



```
$ npm install --save aegis-web-sdk
```

2. 参考下列步骤新建一个 Aegis 实例，传入相应的配置，初始化 SDK 。



```
import Aegis from 'aegis-web-sdk';

const aegis = new Aegis({
    id: 'pGUVFTCZyewhxxxxxx', // 应用ID，即上报ID
    uin: 'xxx', // 用户唯一 ID（可选）
    reportApiSpeed: true, // 接口测速
    reportAssetSpeed: true, // 静态资源测速
    spa: true // spa 应用页面跳转的时候开启 pv 计算
})
```

说明：

为了不遗漏数据，须尽早进行初始化。当您完成安装并初始化 SDK 之后，可以开始使用前端性能监控提供的以下功能：

错误监控：JS 执行错误、Promise 错误、Ajax 请求异常、资源加载失败、返回码异常、PV 上报、白名单检测等。

测速功能：页面性能测速、接口测速、静态资源测速。

数据统计和分析：可在 数据总览 上进行各个维度的数据分析。

说明：

aegis-sdk 默认使用 `https://aegis.qq.com` 作为上报域名，您可以通过 `hostUrl` 参数来控制上报域名。

国内可以选择选择使用 `https://tamaegis.com` 作为上报域名。

国外新加坡地区可以选择 `https://rumt-sg.com` 作为上报域名。

日志上报

最近更新时间：2024-01-22 19:39:30

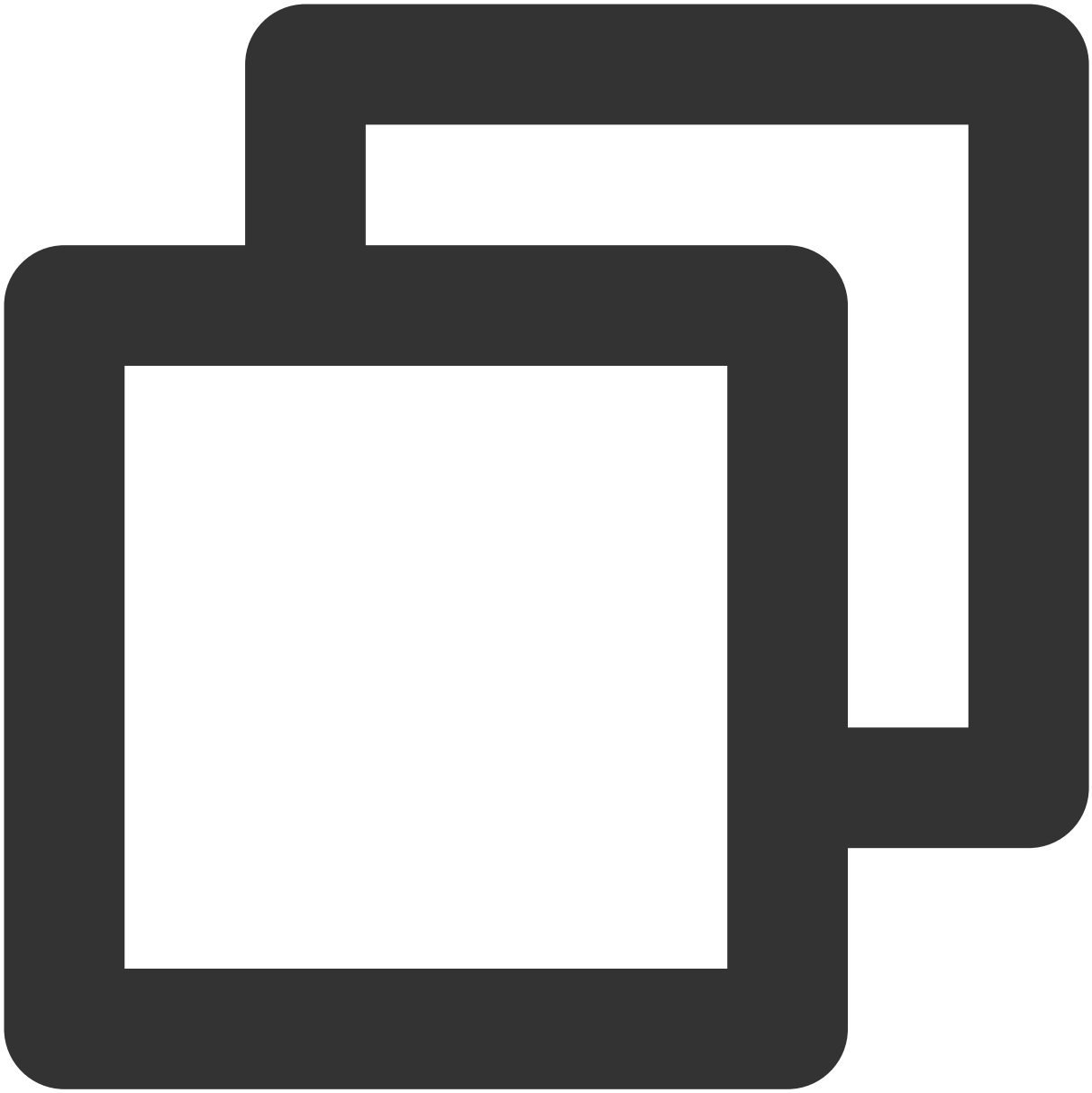
日志查询和离线日志功能需您上报日志后，才能正常使用。本文将为您介绍如何上报日志到前端性能监控。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

日志上报

引入下列参数，配置 SDK ，完成日志上报。



```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
```

```
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.LogType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

aid

最近更新时间：2024-01-22 19:39:30

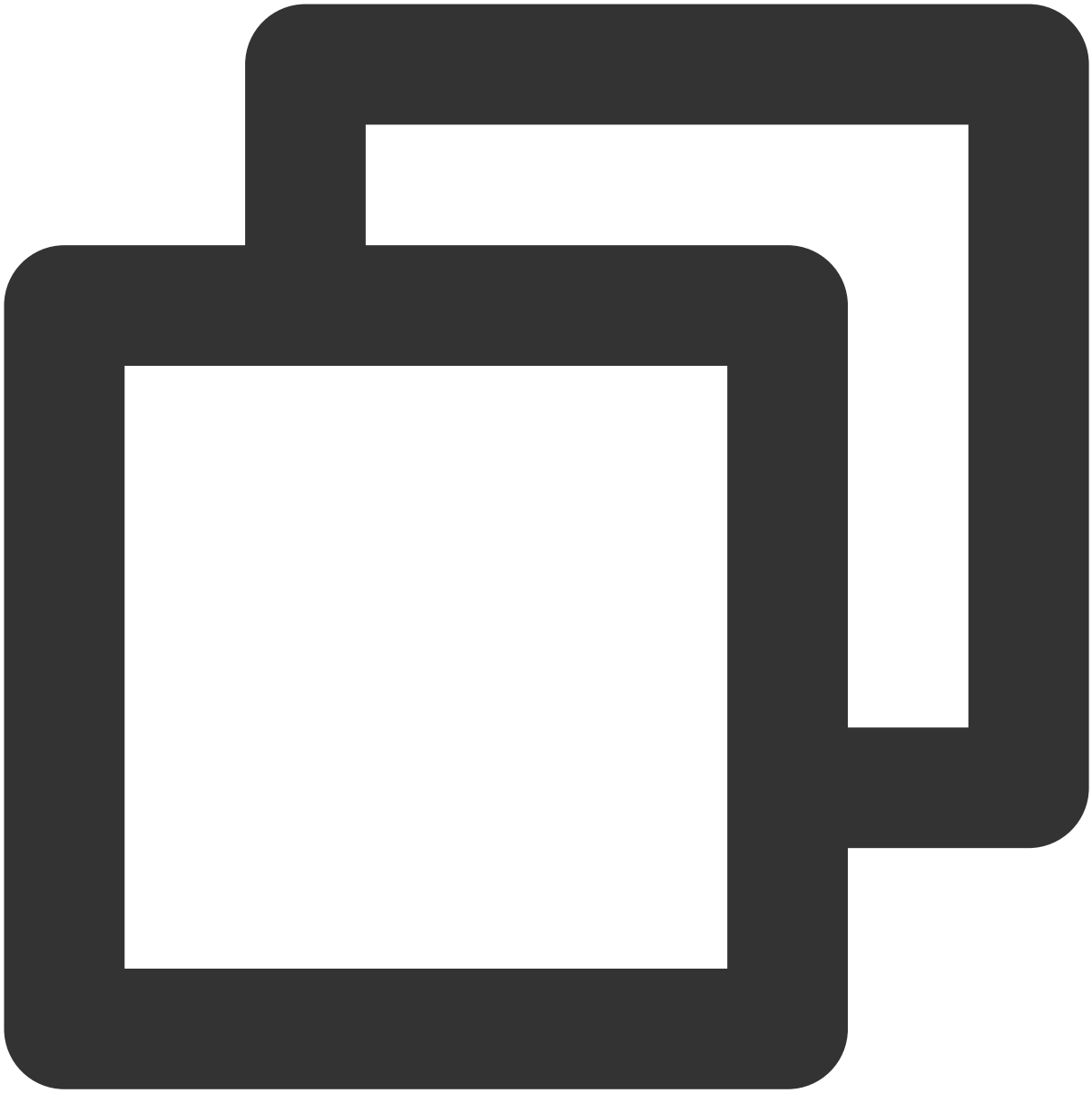
aid 是 Aegis SDK 为每个用户设备分配的唯一标识，存储在浏览器的 localStorage 。用于区分用户计算 UV 等。只有用户清理浏览器缓存 aid 才会更新。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

aid

您可以根据下列算法自定义 aid 上报规则：



```
async getAid(callback: Function) {
// 某些情况下操作 localStorage 会报错.
  try {
    let aid = await localStorage.getItem('AEGIS_ID');
    if (!aid) {
    aid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
      const r = (Math.random() * 16) | 0;
      const v = c === 'x' ? r : (r & 0x3) | 0x8;
      return v.toString(16);
    });
    localStorage.setItem('AEGIS_ID', aid);
```

```
      }
      callback?.(aid || '');
    } catch (e) {
      callback?.('');
    }
  }
```

说明：

若您需要使用自己构造的 aid 作为上报规则，需要后端对 aid 的校验规则，规则如下：`/^[@=.0-9a-zA-Z_-]{4,36}$/` 。

实例方法

最近更新时间：2024-01-22 19:39:30

前端性能监控为您提供多种实例方法用于上报数据，您可以通过实例方法修改实例配置、自定义上报事件、自定义上报测试资源等。
目前 RUM 提供的 Aegis 实例方法如下：

参数	用途
setConfig	传入配置对象，包括用户 ID 和 UIN 等信息。
info	主要上报字段，用于上报白名单日志。下列两种情况日志才会报到后台： 1. 打开页面的用户在名单中。 2. 对应的页面发生了错误。
infoAll	主要上报字段，用于上报白名单日志。该上报与 info 唯一的区别： info 指定用户上报。
error	主要上报字段，用于上报错误信息。
report	用来上报任意类型的日志信息。
reportEvent	上报自定义事件。
reportTime	上报自定义测速资源。
time	上报自定义测速资源，与 timeEnd 共同使用。适用于两个时间点之间时长的计算并上报。
timeEnd	上报自定义测速资源，与 time 共同使用。适用于两个时间点之间时长的计算并上报。
destroy	销毁 aegis 实例。

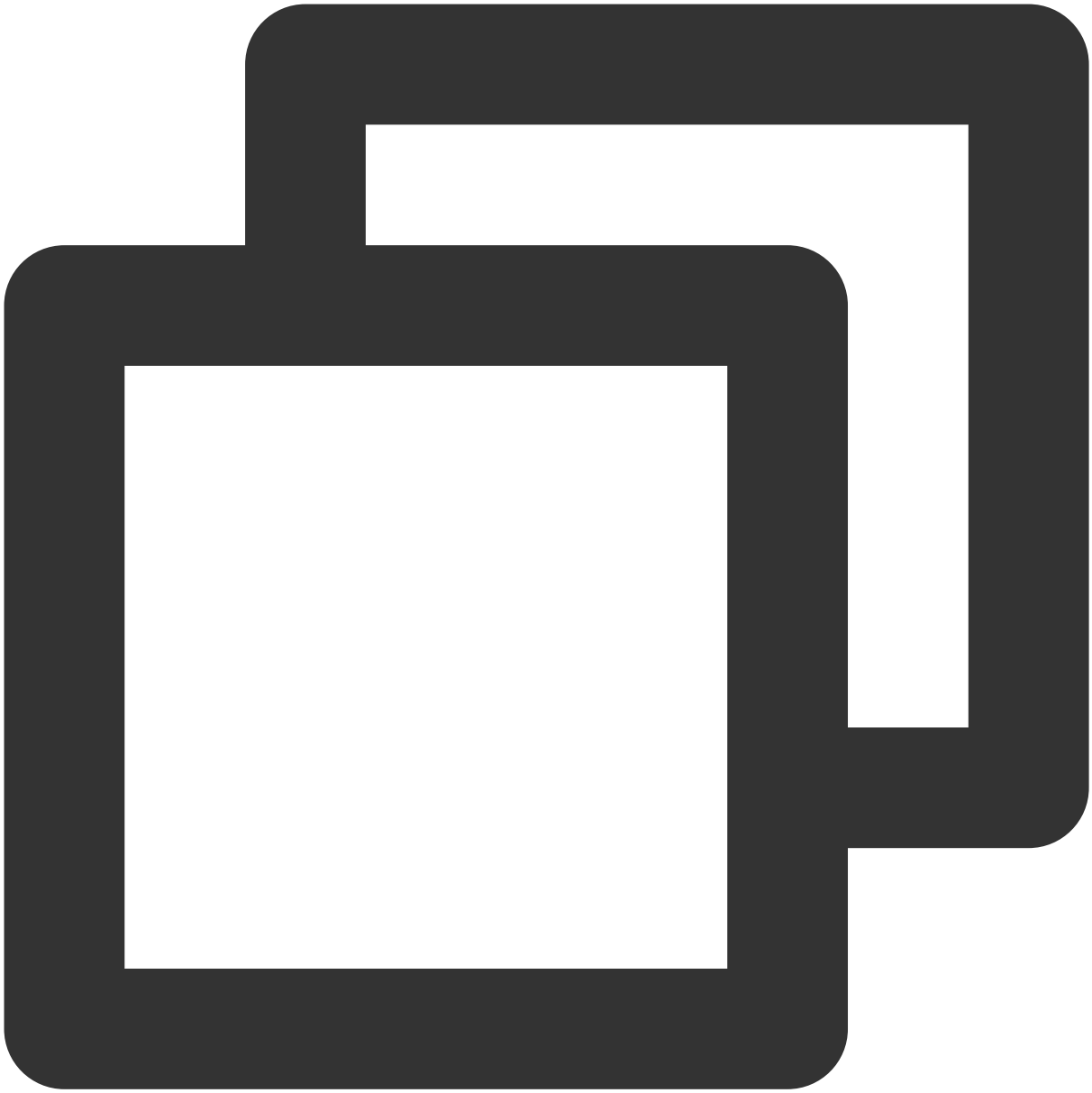
前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

实例方法

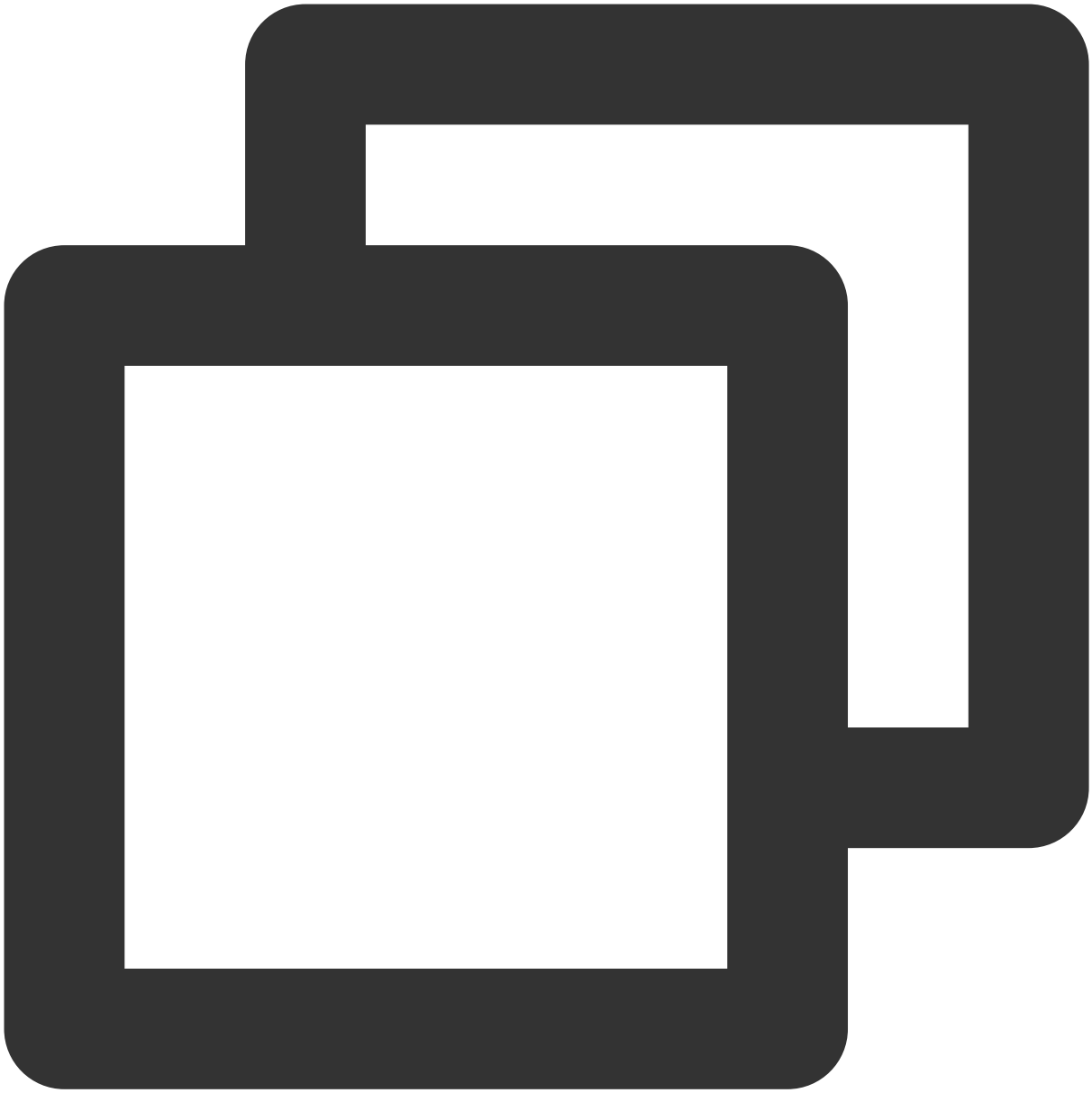
setConfig

该方法用于修改实例配置，使用场景如下：
1. 可获取到用户 UIN ，可同时传入配置用户 ID 和 UIN 两个实例对象，进行实例化：



```
const aegis = new Aegis({
    id: 'pGUVFTCZyewxxxxx',
    uin: '777'
})
```

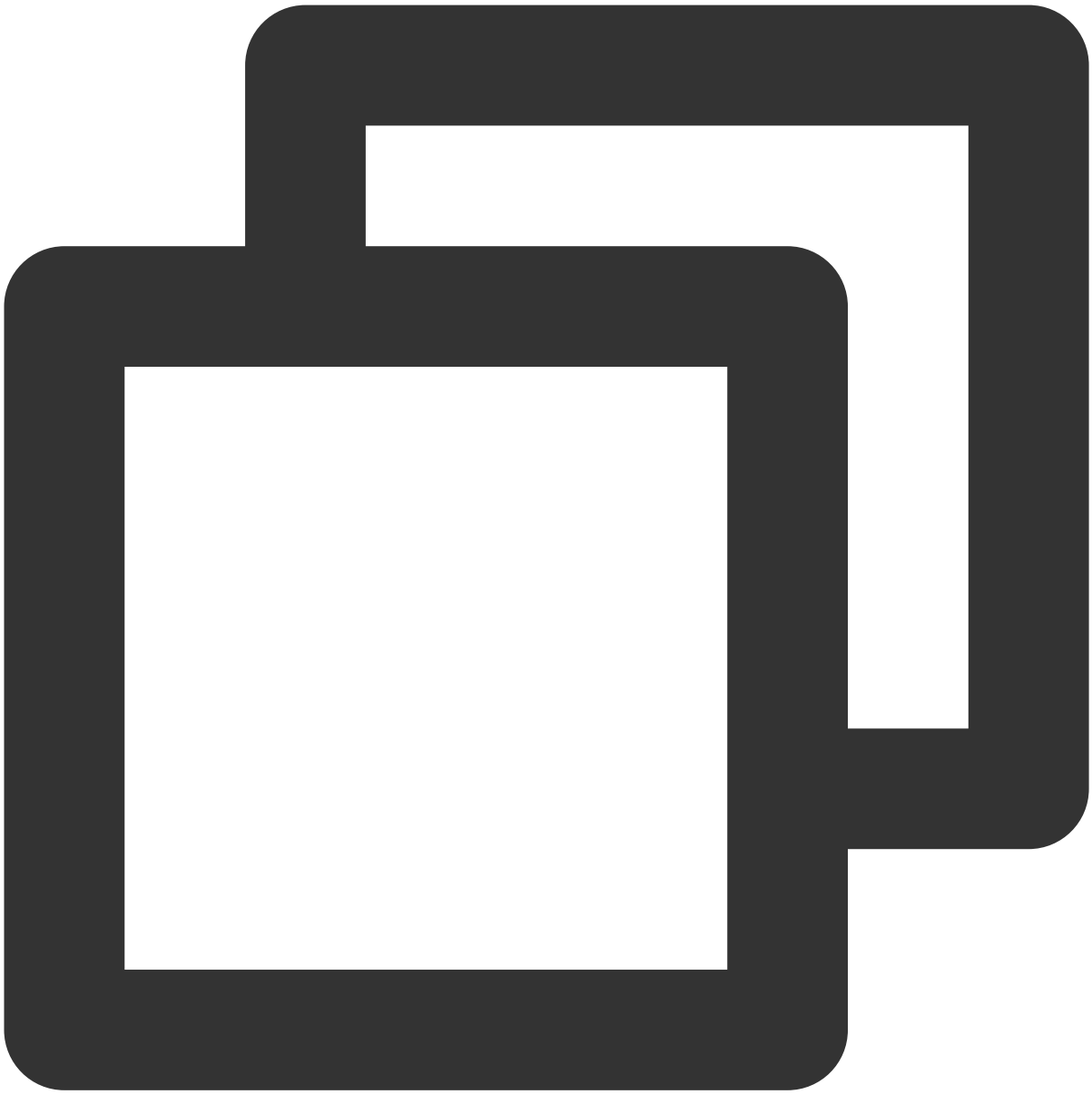
2. 通常情况下，我们并不能一开始就获取到用户的 `uin` 。若在获取 UIN 的这段时间不进行实例化，这期间发生的错误前端性能监控将无法监听。针对这种情况，我们可以先传入 ID 进行实例化，引用 setConfig 传入 UIN ，示例如下：



```
const aegis = new Aegis({
    id: 'pGUVFTCZyewxxxxx'
})

// 拿到uin之后...
aegis.setConfig({
  uin: '6666'
})
```

info、infoAll、error 和 report

这三个方法是前端性能监控提供的主要上报手段。



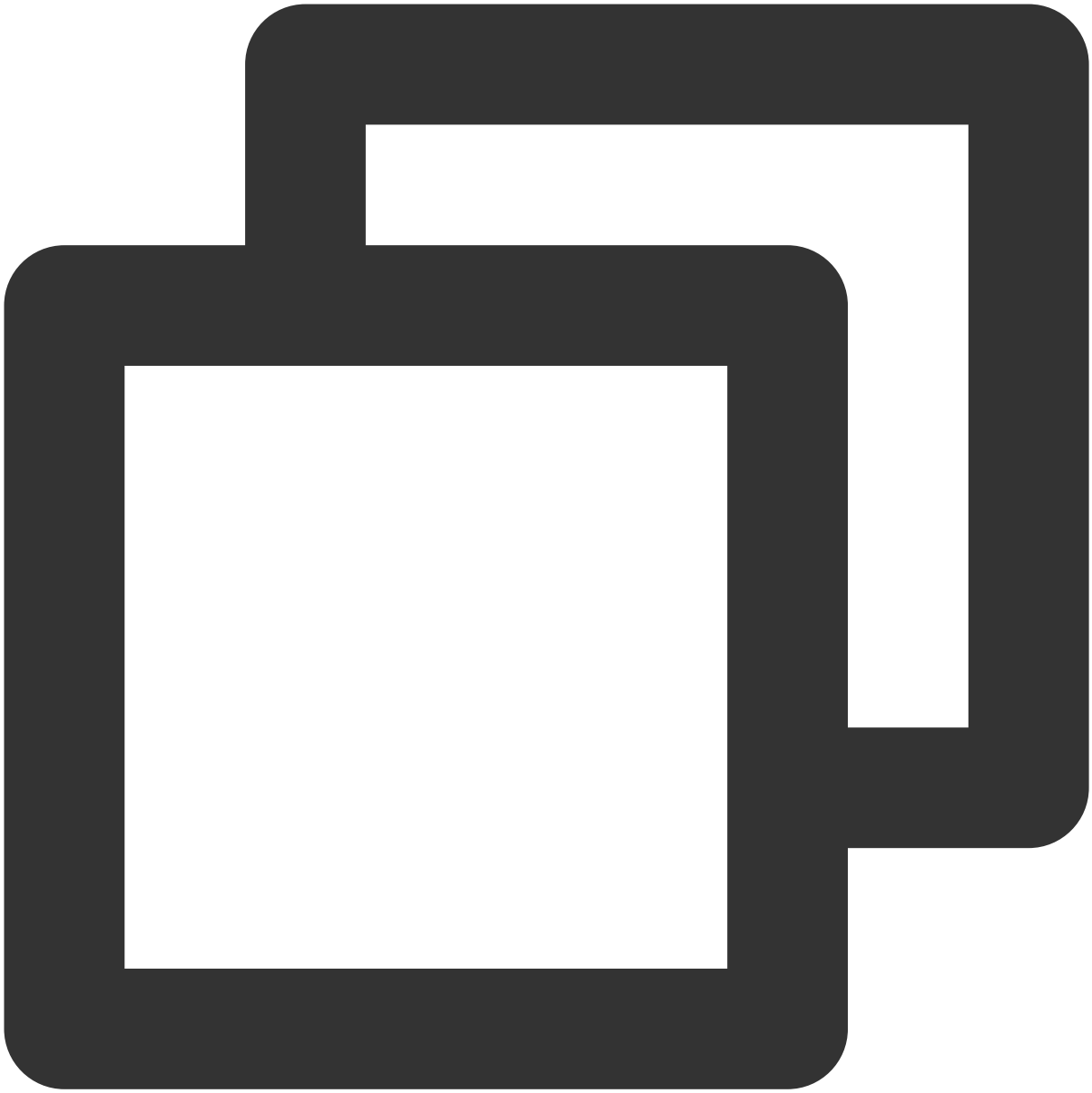
```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
```

```
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.LogType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

reportEvent

该方法可用来上报自定义事件，系统将会自动统计上报事件的各项指标，例如：PV、平台分布等。
reportEvent 可以支持字符串和对象两种类型上报参数。

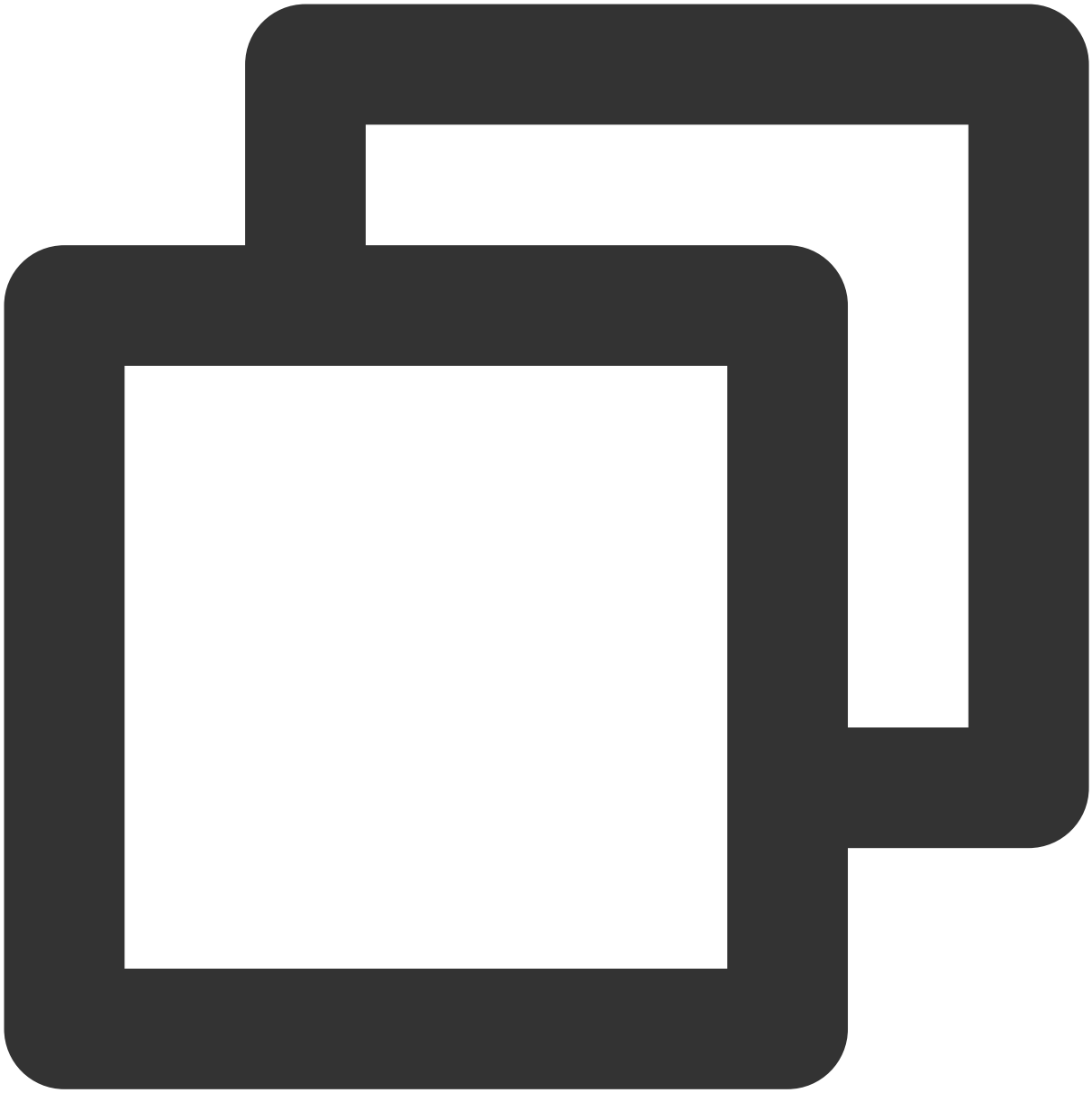
字符串类型



```
aegis.reportEvent('XXX请求成功');
```

对象类型

ext1、ext2 和 ext3 默认使用 new Aegis 的时候传入的参数，自定义事件上报的时候，可以覆盖默认值。



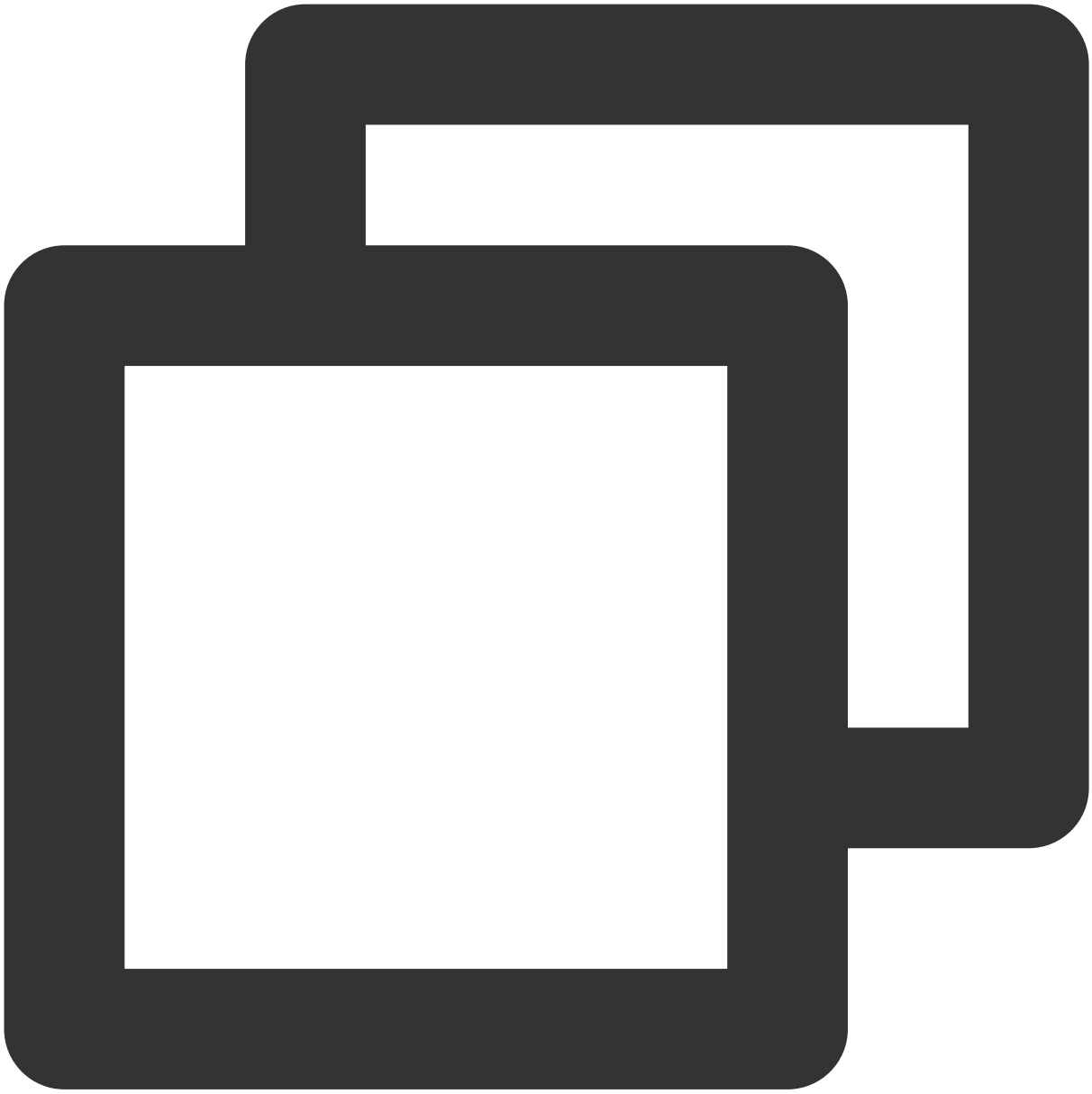
```
aegis.reportEvent({
  name: 'XXX请求成功', // 必填
  ext1: '额外参数1',
  ext2: '额外参数2',
  ext3: '额外参数3',
})
```

注意：

额外参数的三个 key 是固定的，目前只支持 ext1、ext2 和 ext3。

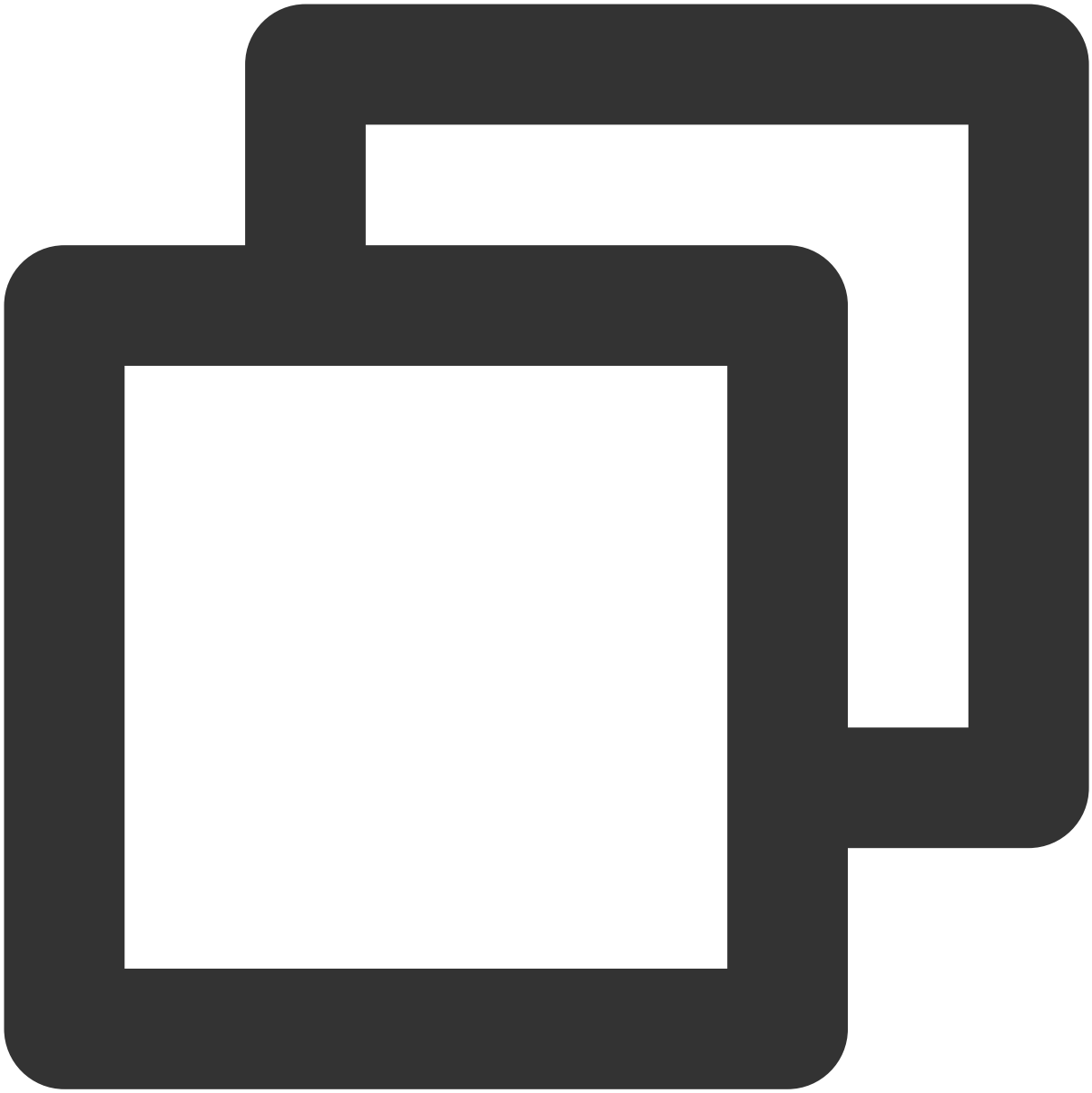
reportTime

该方法可用来上报自定义测速，例如：



```
// 假如‘onload’的时间是1s
aegis.reportTime('onload', 1000);
```

或者如果需要使用额外参数，可以传入对象类型参数，ext1，ext2，ext3 会覆盖默认值：



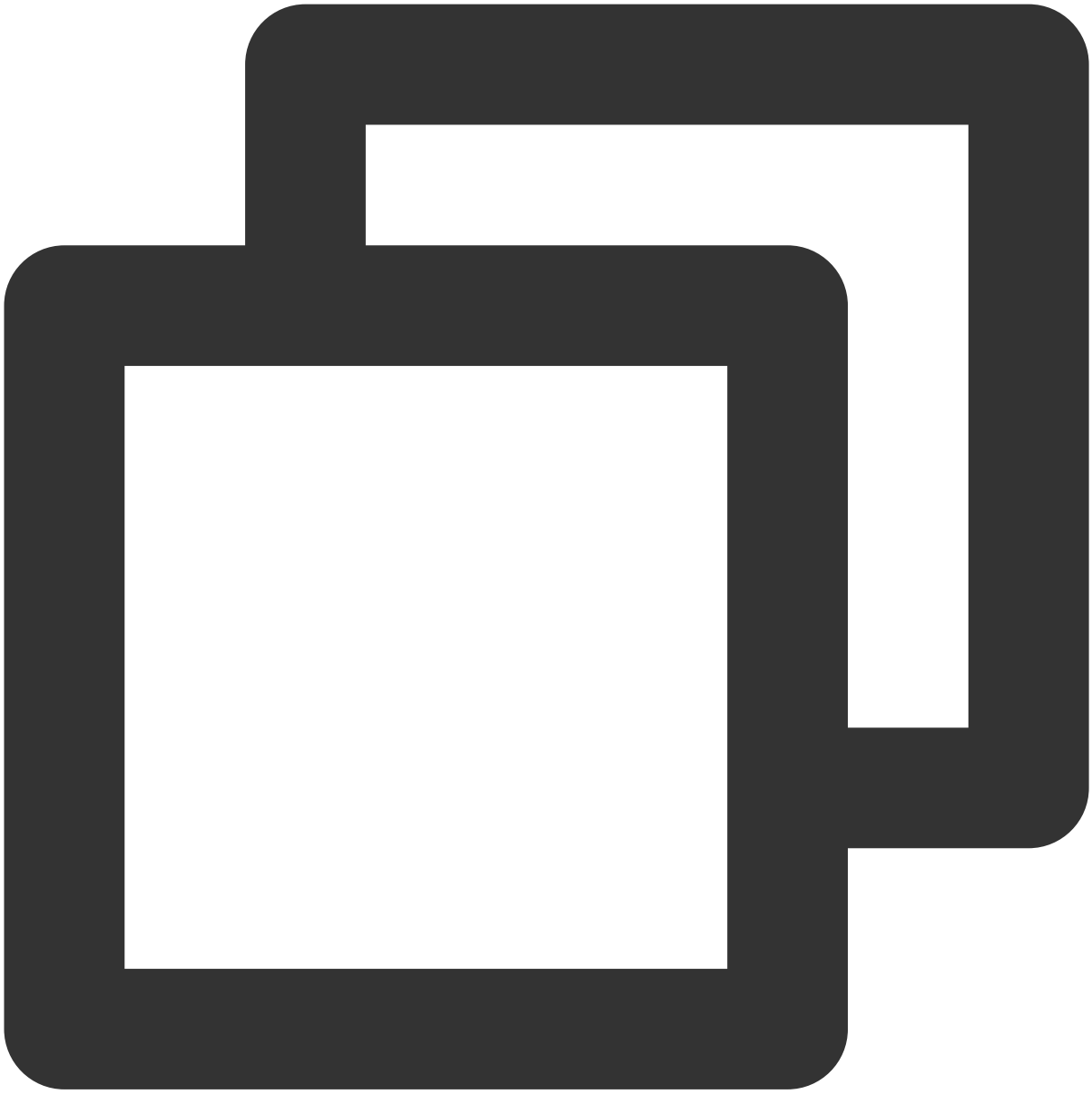
```
aegis.reportTime({
  name: 'onload', // 自定义测速 name
  duration: 1000, // 自定义测速耗时(0 - 60000)
  ext1: 'test1',
  ext2: 'test2',
  ext3: 'test3',
});
```

说明：

`onload` 可以修改为其他的命名。

time 和 timeEnd

该方法同样可用来上报自定义测速，适用于两个时间点之间时长的计算并上报，例如：



```
aegis.time('complexOperation');
/**
 * .
 * .
 * 做了很久的复杂操作之后。
 * .
 * .
 */
```

```
aegis.timeEnd('complexOperation'); /** 此时日志已经报上去了**/
```

说明：

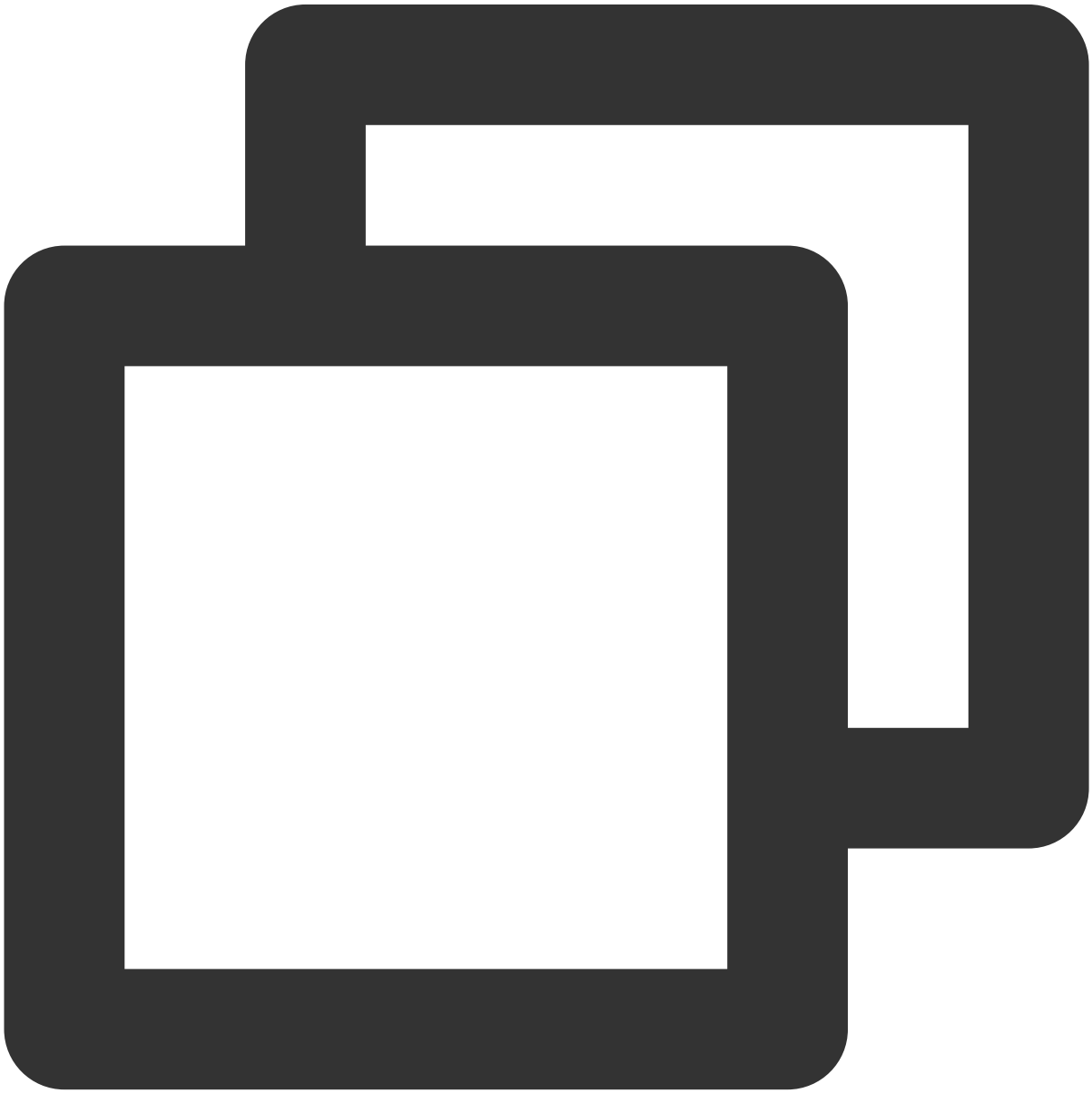
`complexOperation` 可以修改为其他的命名。

自定义测速是用户上报任意值，服务端对其进行统计和计算。由于服务端不能做脏数据处理，建议用户在上报端进行统计值限制，防止脏数据对整体产生影响。

目前 Aegis 只支持 0 - 60000 的数值计算，如果大于该值，建议进行合理改造。

destroy

销毁实例进程，销毁后数据不再上报，并且 Aegis 不再收集用户数据。



```
aegis.destroy();
```

白名单

最近更新时间：2024-01-22 19:39:30

白名单功能是适用于开发者针对特定的用户上报更多的信息。为了过滤部分用户，并减少用户的接口请求次数，因此 RUM 提供了白名单功能，并设定了白名单的逻辑。
白名单逻辑如下：
1. 白名单用户会上报全部的 API 请求信息，包括接口请求和请求结果。
2. 白名单用户可以使用 info 接口上报数据。
3. info 和 infoAll：在开发者实际体验过程中，白名单用户可以添加更多的日志，并且使用 info 进行上报。infoAll 会对所有用户无差别进行上报，因此可能导致日志量上报巨大。
4. 通过接口 whitelist 来判断当前用户是否是白名单用户，白名单用户的返回结果会绑定在 aegis 实例上 (aegis.isWhiteList) 用来给开发者使用。
5. 用了减少开发者使用负担，白名单用户是针对业务系统生效。您可以在**应用管理>白名单管理**内创建白名单，则业务系统下全部应用都对该白名单用户生效。

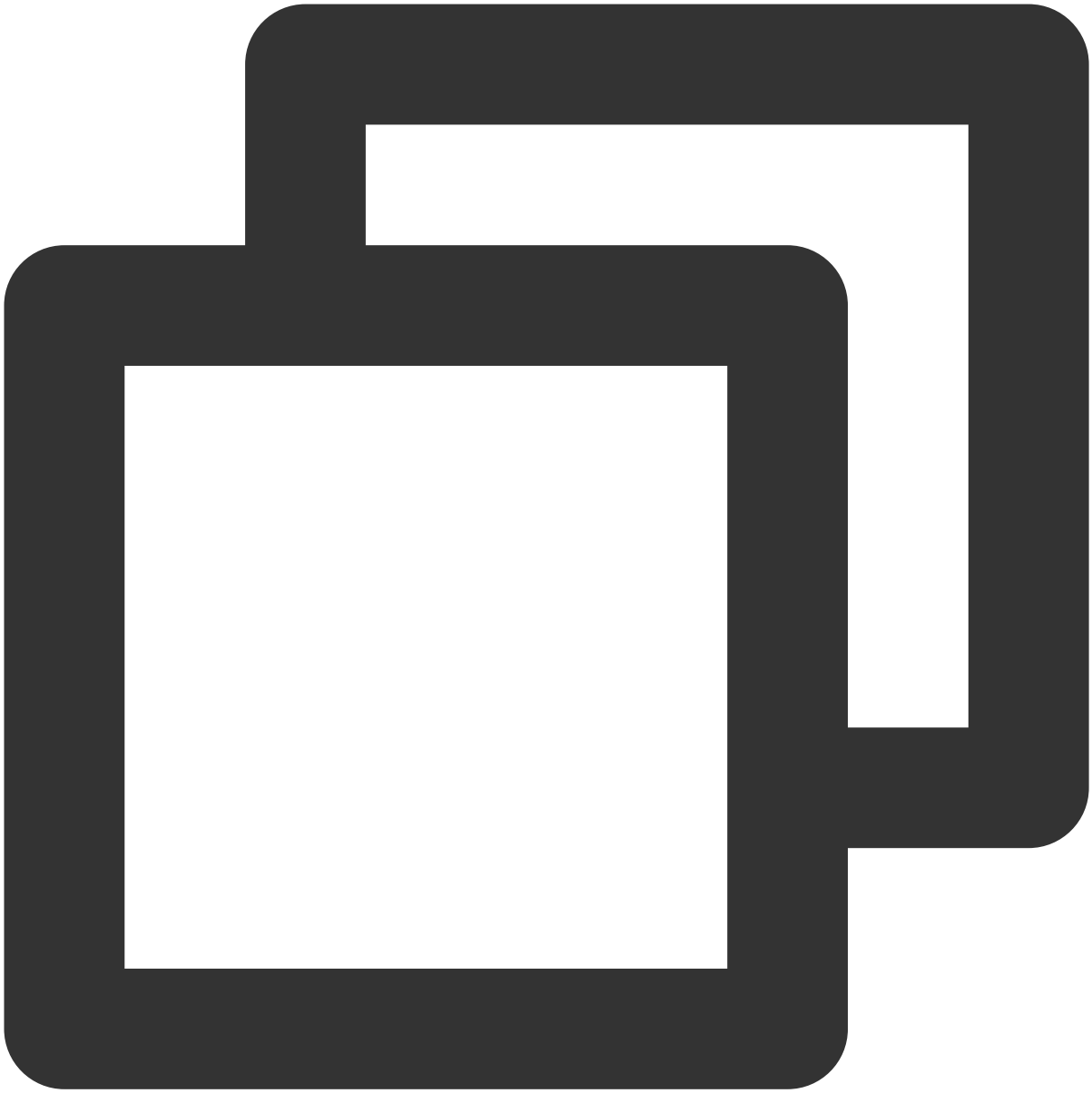
钩子函数

最近更新时间：2024-01-22 19:39:31

您可以使用钩子函数对某些资源的测速上报进行自定义配置，系统将会为您计算、统计函数执行时间等。您可以在自定义测速多维度分析函数执行耗时。

onBeforeRequest

该钩子函数会在所有请求发出前调用，参数中会传入请求的所有内容，必须返回待发送内容。

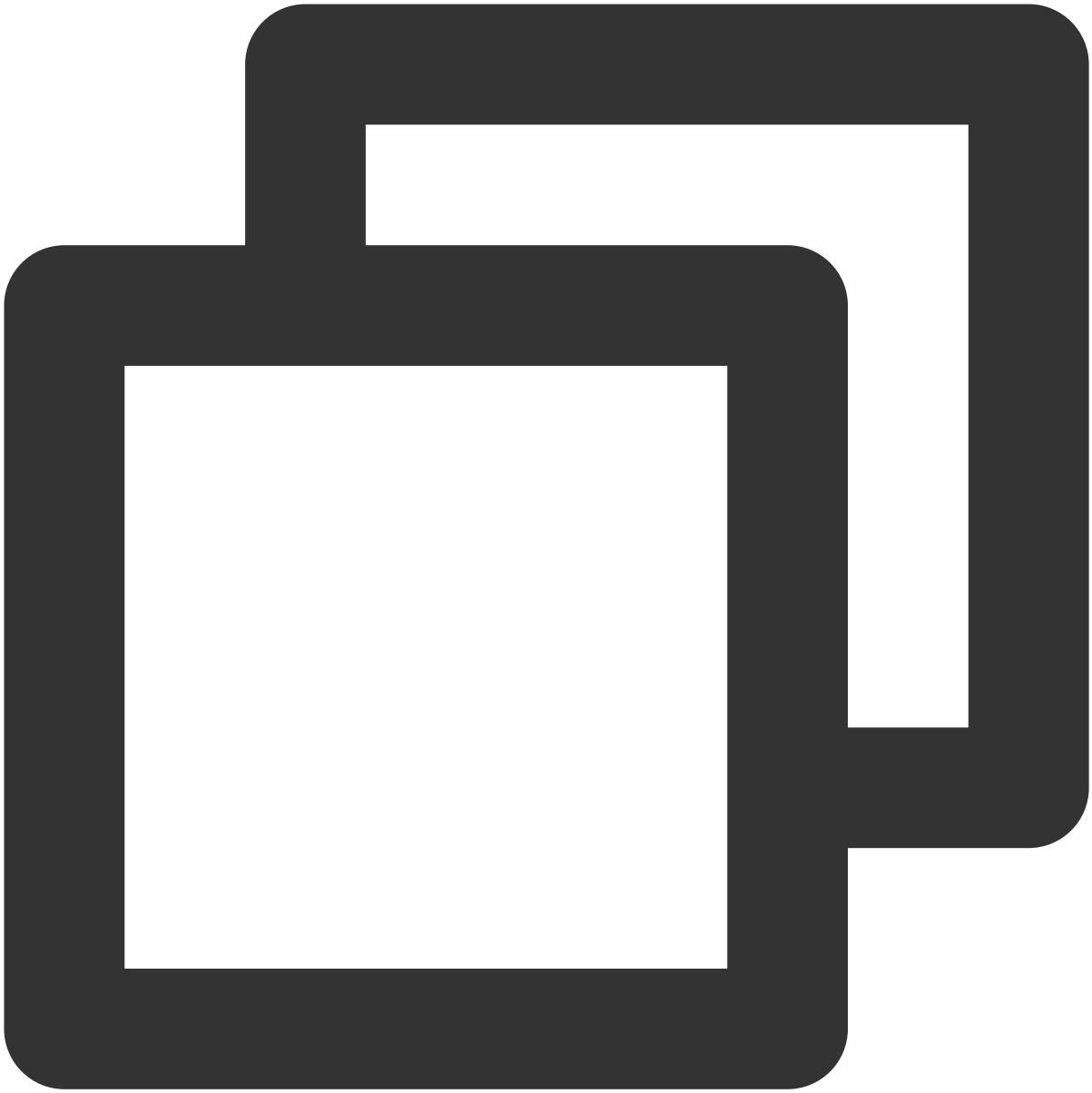


```
function changeURLArg(url,arg,arg_val) {
  var pattern=arg+' = ([^&]*)';
  var replaceText = arg+'='+arg_val;
  if (url.match(pattern)) {
    var tmp = '/('+ arg+'=)([^&]*)/gi';
    tmp = url.replace(eval(tmp),replaceText);
    return tmp;
  }
  return url;
}
const aegis = new Aegis({
```

```
  id: 'pGUVFTCZyewxxxxx',
  onBeforeRequest(log) {
    if (log.type === 'performance') {
      // 页面测速，此时可以修改log内容，如修改页面测速platform
      log.url = changeURLArg(log.url, 'platform', type)
    }
    return log
  }
});

// SEND_TYPE {
//   LOG = 'log',  // 日志
//   SPEED = 'speed', // 接口和静态资源测速
//   PERFORMANCE = 'performance', // 页面测速
//   OFFLINE = 'offline', // 离线日志上传
//   WHITE_LIST = 'whiteList', // 白名单
//   VITALS = 'vitals', // vitals
//   PV = 'pv', // 自定义pv
//   EVENT = 'event', // 自定义事件
//   CUSTOM = 'custom', // 自定义测速
//   SDK_ERROR = 'sdkError', // sdk报错
// }
```

beforeReport

1. 该钩子将会在日志上报（对应上报接口为 `/collect?` ）前执行，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    // 监听到下面抛出的错误
    console.log(log); // {level: "4", msg: "发生错误啦！！！！"}
    return log;
  }
});

throw new Error('发生错误啦！！！！');
```

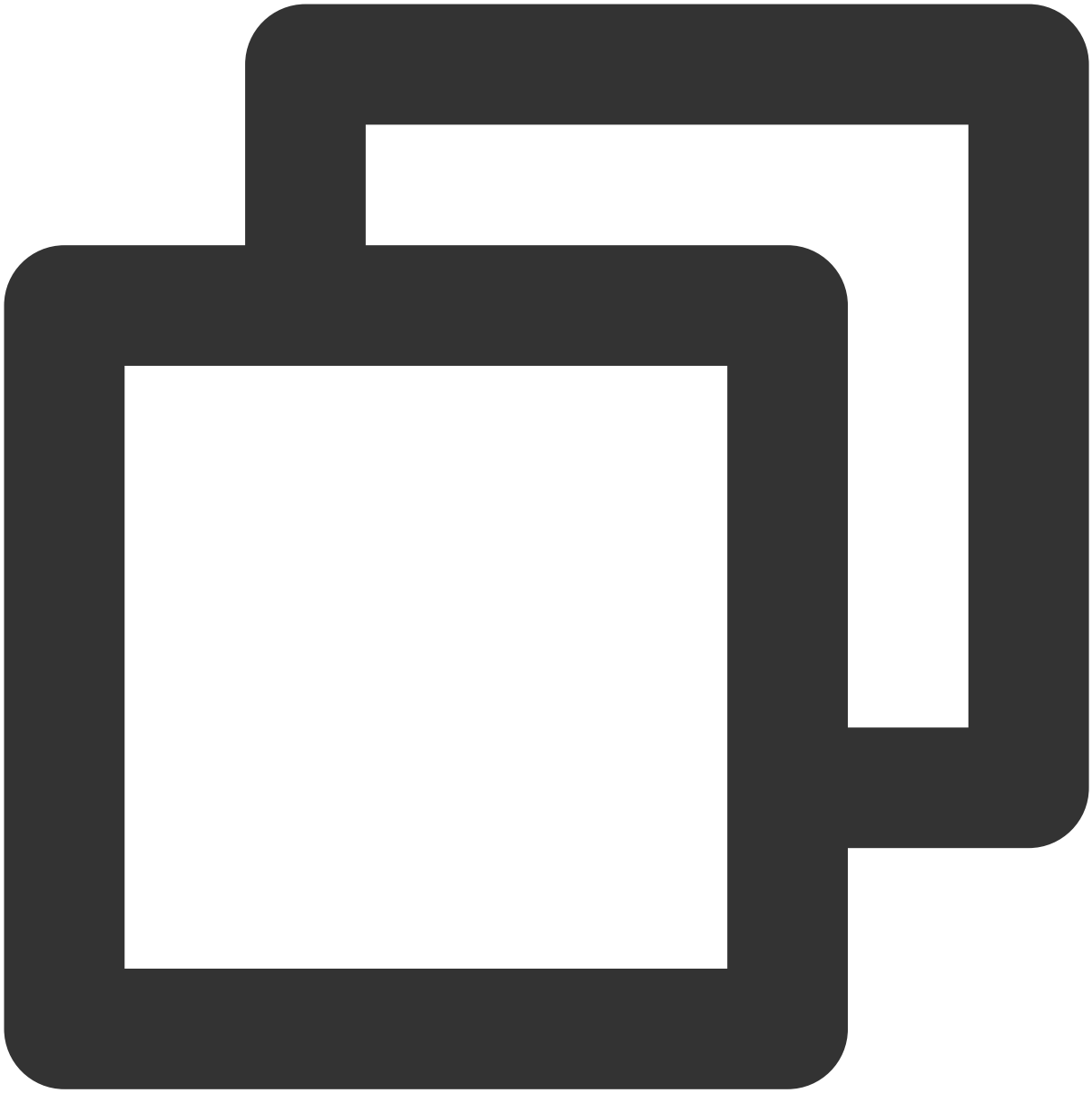
说明：

上述 `log` 将会有以下几个字段：

1. `level` ：日志等级，例如：当 level 为 `'4'` 时代表错误日志；
2. `msg` ：日志内容；
3. 全部日志等级如下：

```
{ level: '1', name: '接口请求日志（白名单日志）' }
{ level: '2', name: '一般日志(aegis.info 或者 aegis.infoAll)' }
{ level: '4', name: 'JS 执行错误' }
{ level: '8', name: 'Promise 错误' }
{ level: '16', name: 'Ajax 请求异常' }
{ level: '32', name: 'JS 加载异常' }
{ level: '64', name: '图片加载异常' }
{ level: '128', name: 'css 加载异常' }
{ level: '256', name: 'console.error （未启用)' }
{ level: '512', name: '音视频资源异常' }
{ level: '1024', name: 'retcode 异常' }
{ level: '2048', name: 'aegis report' }
{ level: 4096, name: 'PV' }
{ level: 8192, name: '自定义事件' }
```

2. 当该钩子返回 `false` 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    if (log.level === '4' && log.msg && log.msg.indexOf('碍眼的错误') !== -1) {
      return false
    }
    return log;
  }
});
throw new Error('碍眼的错误'); // 该错误将不会被上报
```

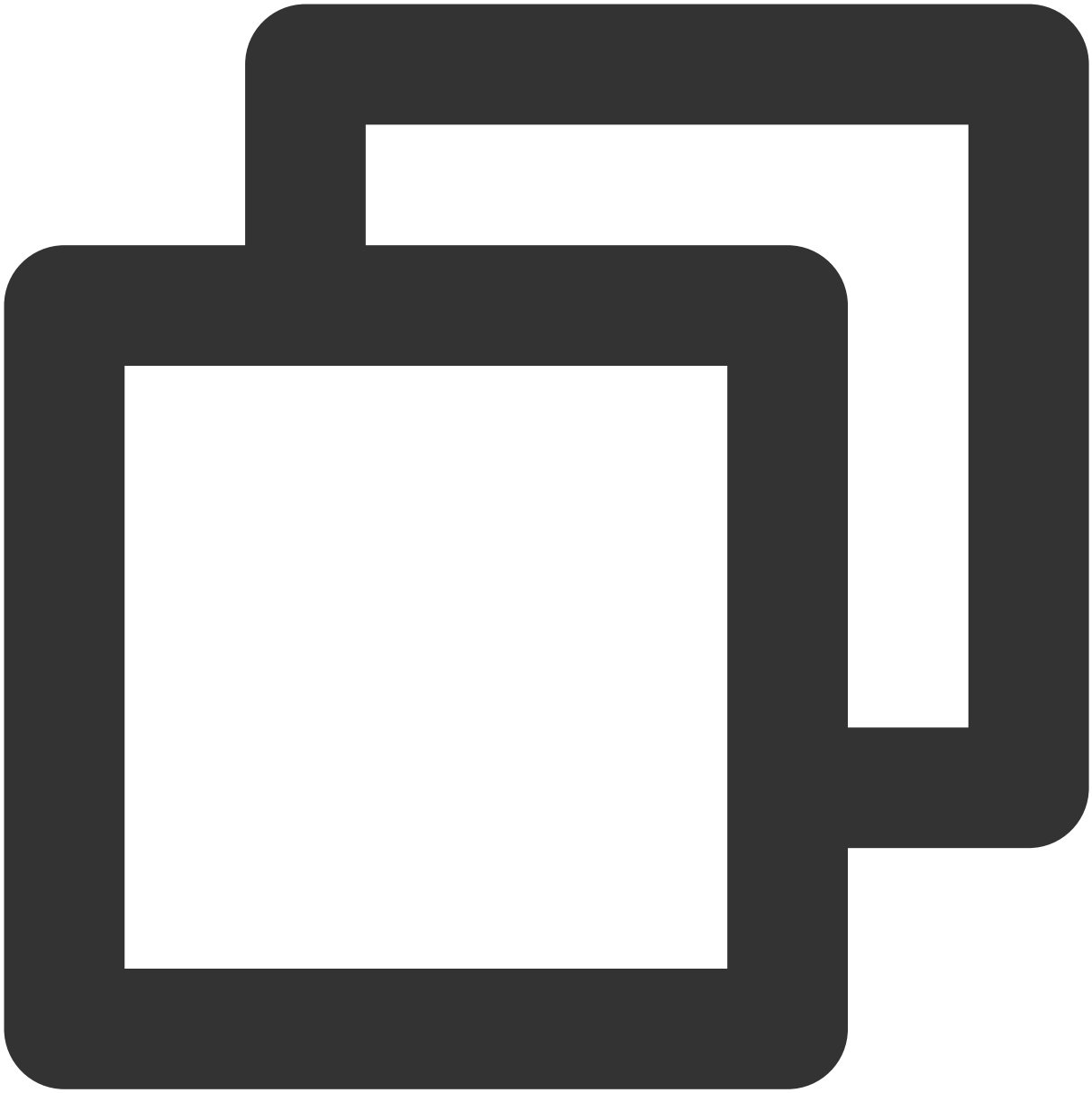
上面例子中，当上报的错误内容包含 `碍眼的错误` 几个关键字时，将不会上报至 RUM 后台中。

onReport

该钩子将在日志上报成功之后执行，用法类似 `beforeReport` 钩子，唯一不同点在于，该钩子接收到的所有参数都是已经上报完成的日志，而 `beforeReport` 钩子接收的参数是即将上报的日志。

beforeReportSpeed

1. 该钩子将会在测速数据上报前（ `/speed?` ）被执行，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    console.log(msg); // {url: "https://localhost:3001/example.e31bb0bc.js", method
    return msg
  }
});
```

说明：

上述 `msg` 将会有以下几个字段：

1. `url`：该资源的请求地址；
2. `type`：该资源的类型，目前有 `fetch` 、 `static` 两种，当为 `fetch` 时，Aegis 将会把该资源当成 API 请求进行上报， `static` 时则视为静态资源；
3. `duration`：该资源请求耗时；
4. `method`：请求该资源时使用的 `http method` ；
5. `status`：服务器返回状态码；
6. `payload`：提供给开发人员的完整资源请求信息（此数据不上报到 Aegis 后台，用户可自行操作）完整的数据结构如下：

`payload.type` - 表示该资源请求的类型，用于区分原始请求类型，可取值为： `'fetch'` , `'xhr'`

`payload.sourceURL` - 表示完整的 URL 请求连接

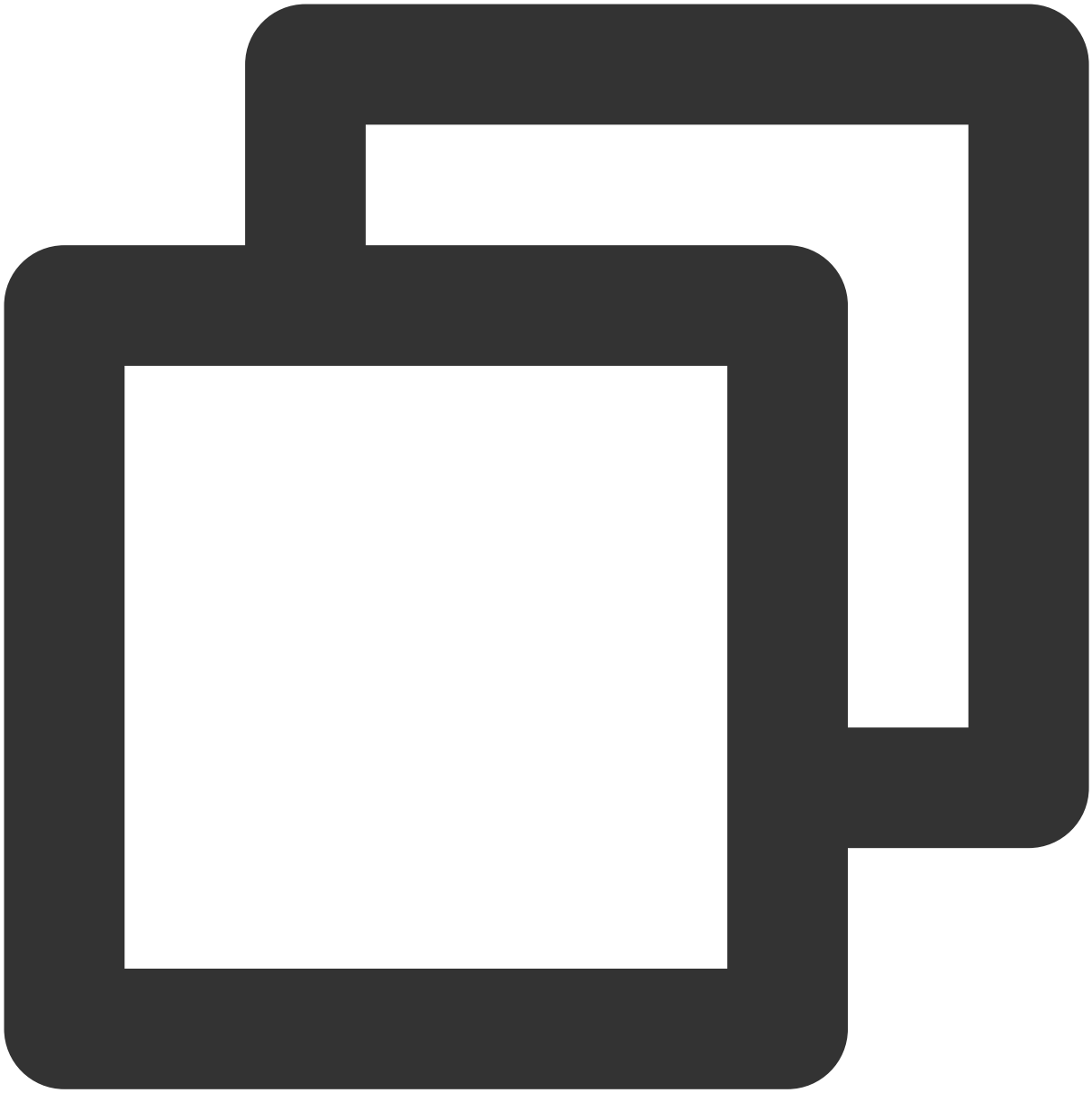
`payload.status` - 表示请求状态码

`payload.headers` - 包含所有的请求头，且 value 值都为字符串

`payload.data` - 表示完整的请求资源，用户可自定义操作（当请求类型为 fetch 时，表示 response 对象；当请求类型为 XHR 时，表示 XMLHttpRequest 对象）

上面的例子中，每当 Aegis 收集到一个资源的加载详情时，将会以该资源的加载情况（上面返回的 `msg` ）作为参数调用 `beforeReportSpeed` 钩子。

2. 如果您配置了该钩子，Aegis 最终的上报内容将以钩子的执行结果为准。例如：



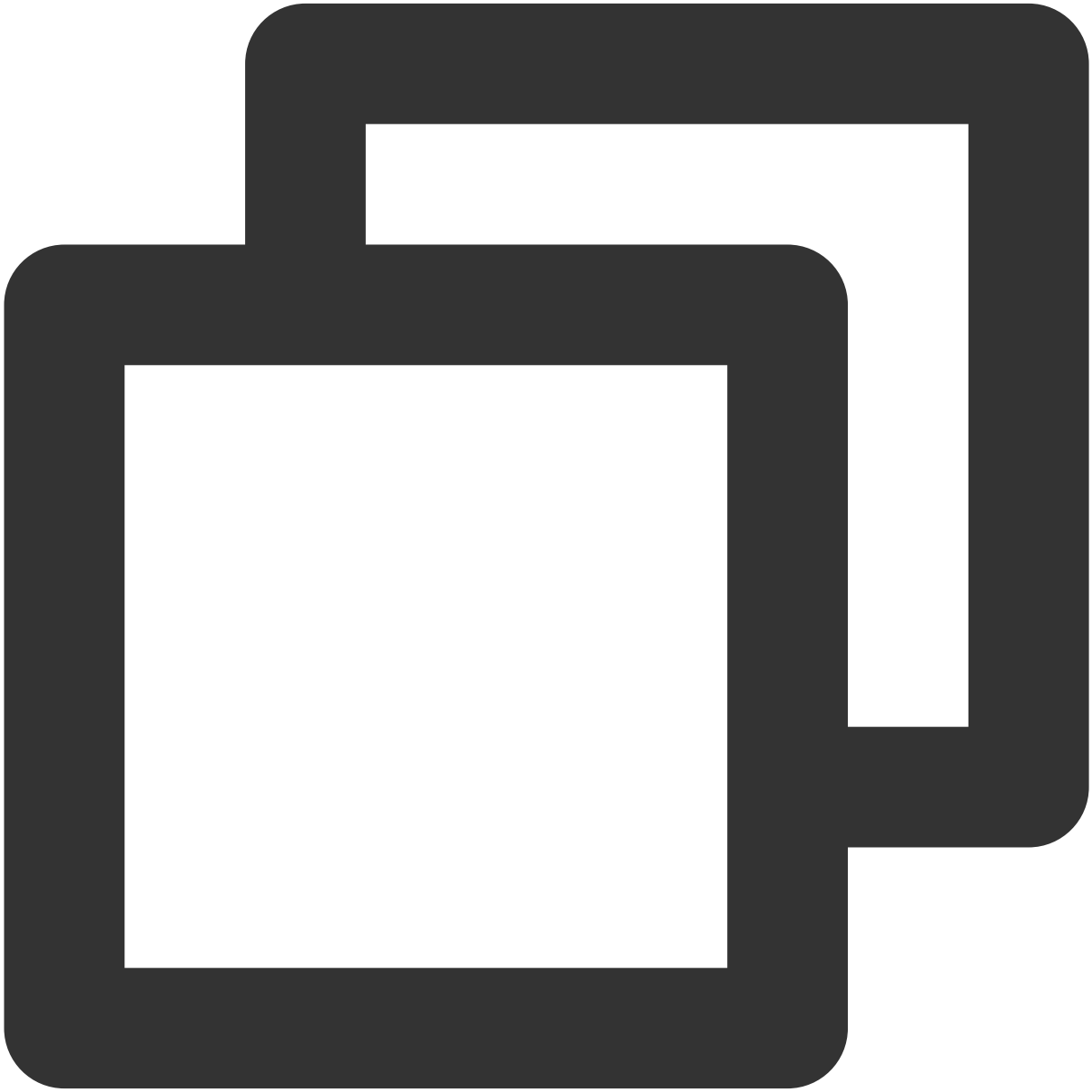
```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    msg.type = 'static';
    return msg;
  }
});
```

上面的代码中，将所有的 `msg.type` 设置为 `static`，这意味着所有的资源都将被当成静态资源进行上报，API 请求也将被报至静态资源中。

3. 使用该钩子，您可以校准 Aegis 类型判断错误的请求。

示例：

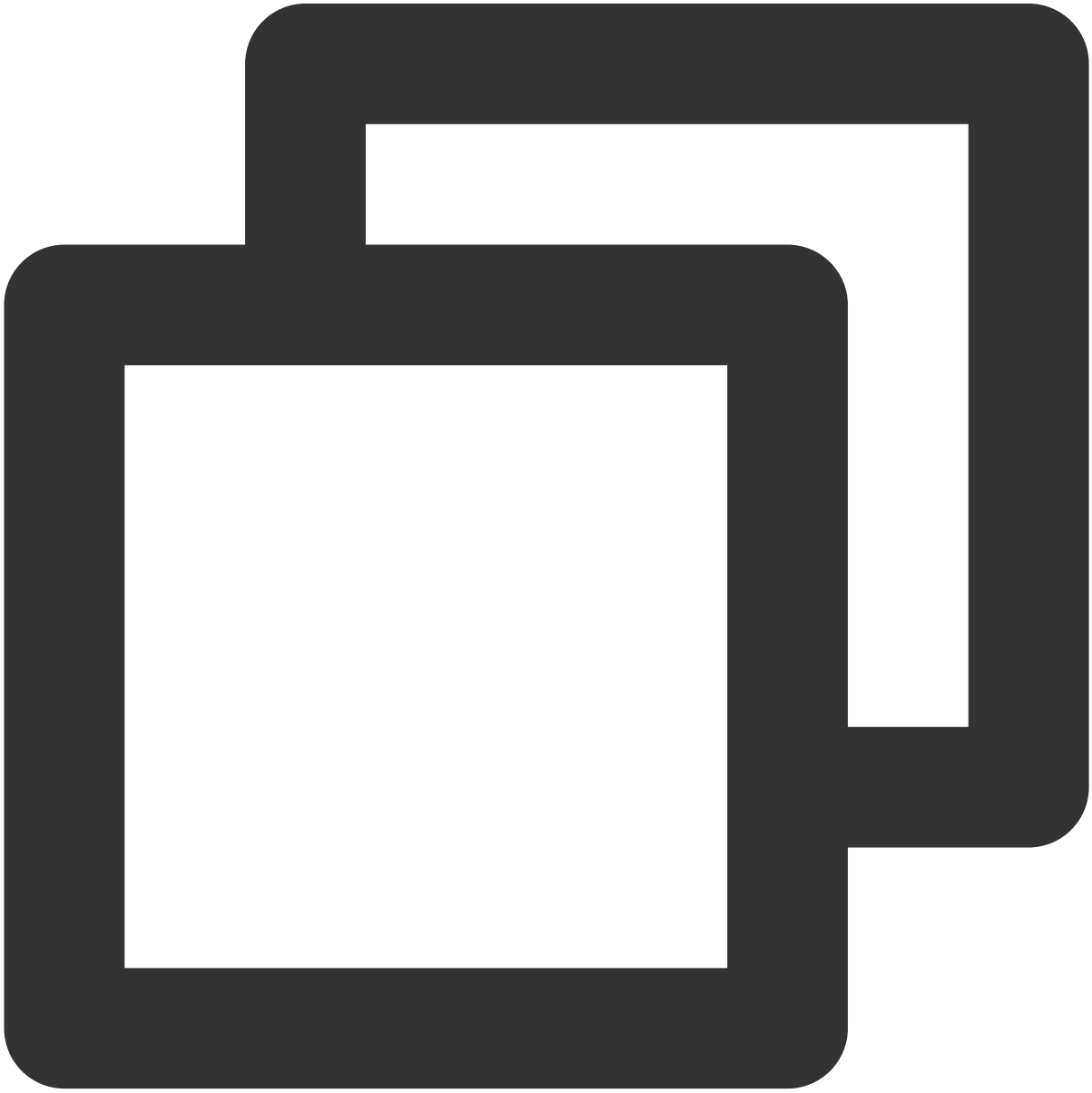
假如您有一条接口 `https://example.com/api`，该接口的响应头 `Content-Type` 为 `text/html`。正常情况下，RUM 会将该资源当成静态资源进行上报。但在您的业务中，该接口就必须视为 API 请求进行上报，您可以给 Aegis 配置如下钩子进行校正：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
```

```
    reportApiSpeed: true,
    reportAssetSpeed: true,
    beforeReportSpeed(msg) {
      if (msg.url === 'https://example.com/api') {
        msg.type = 'fetch';
      }
    }
});
```

4. 您还可以屏蔽某些资源的测速上报，例如：



```
const aegis = new Aegis({
```

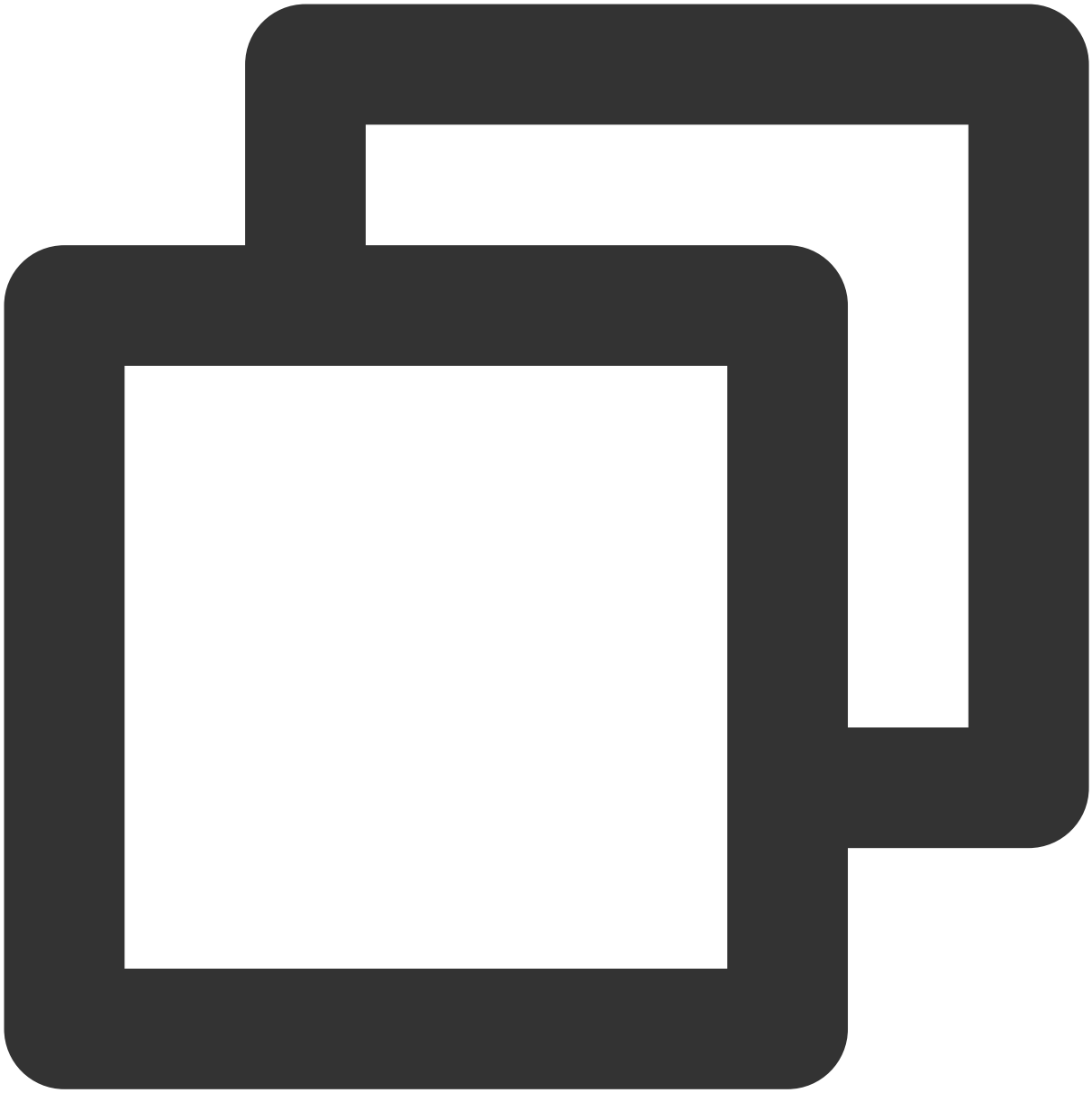
```
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    // 地址中包含‘https://example.com/api’的都不上报
    if (msg.url.indexOf('https://example.com/api') !== -1) {
      // 返回 ‘false’ 将阻止本条测速日志的上报
      return false
    }
  }
});
```

beforeRequest

该钩子将会在日志上报前执行，例如：

注意：

SDK 版本应大于等于 1.24.44。



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeRequest: function(msg) {
    if (msg.logs && msg.logs.level === '4' && msg.logs.msg && msg.logs.msg.indexOf(
      return false
    }
    return msg;
  }
});
```

其中，msg 将会有以下几个字段：

1. logType：日志类型，有以下值：

custom：自定义测速

event：自定义事件

log：日志

performance：页面测速

pv：页面 PV

speed：接口和静态资源测速

vitals：web vitals

2. logs：上报的日志内容：

当 logType 为 'custom' 时，logs 数据类型为 `{name: "白屏时间", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}` 。

当 logType 为 'event' 时，logs 数据类型为 `{name: "ios", ext1: "", ext2: "", ext3: ""}` 。

当 logType 为 'performance' 时，logs 数据类型为 `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 2660, ssl: 4, tcp: 4, ttfb: 5}` 。

当 logType 为 'speed' 时，logs 数据类型为 `{connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "static", url: "https://xxxxxx", urlQuery: "max_age=1296000"}` 。

当 logType 为 'vitals' 时，logs 数据类型为 `{delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "LCP", value: 1100}` 。

当 logType 为 'log' 时，logs 数据类型为 `{msg: "日志详情", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}` 。

说明：

其中 level 枚举值如下：

```
{ level: '1', name: '接口请求日志（白名单日志）' }
{ level: '2', name: '一般日志(aegis.info 或者 aegis.infoAll)' }
{ level: '4', name: 'JS 执行错误' }
{ level: '8', name: 'Promise 错误' }
{ level: '16', name: 'Ajax 请求异常' }
{ level: '32', name: 'JS 加载异常' }
{ level: '64', name: '图片加载异常' }
{ level: '128', name: 'css 加载异常' }
{ level: '256', name: 'console.error (未启用)' }
{ level: '512', name: '音视频资源异常' }
{ level: '1024', name: 'retcode 异常' }
{ level: '2048', name: 'aegis report' }
{ level: 4096, name: 'PV' }
{ level: 8192, name: '自定义事件' }
```

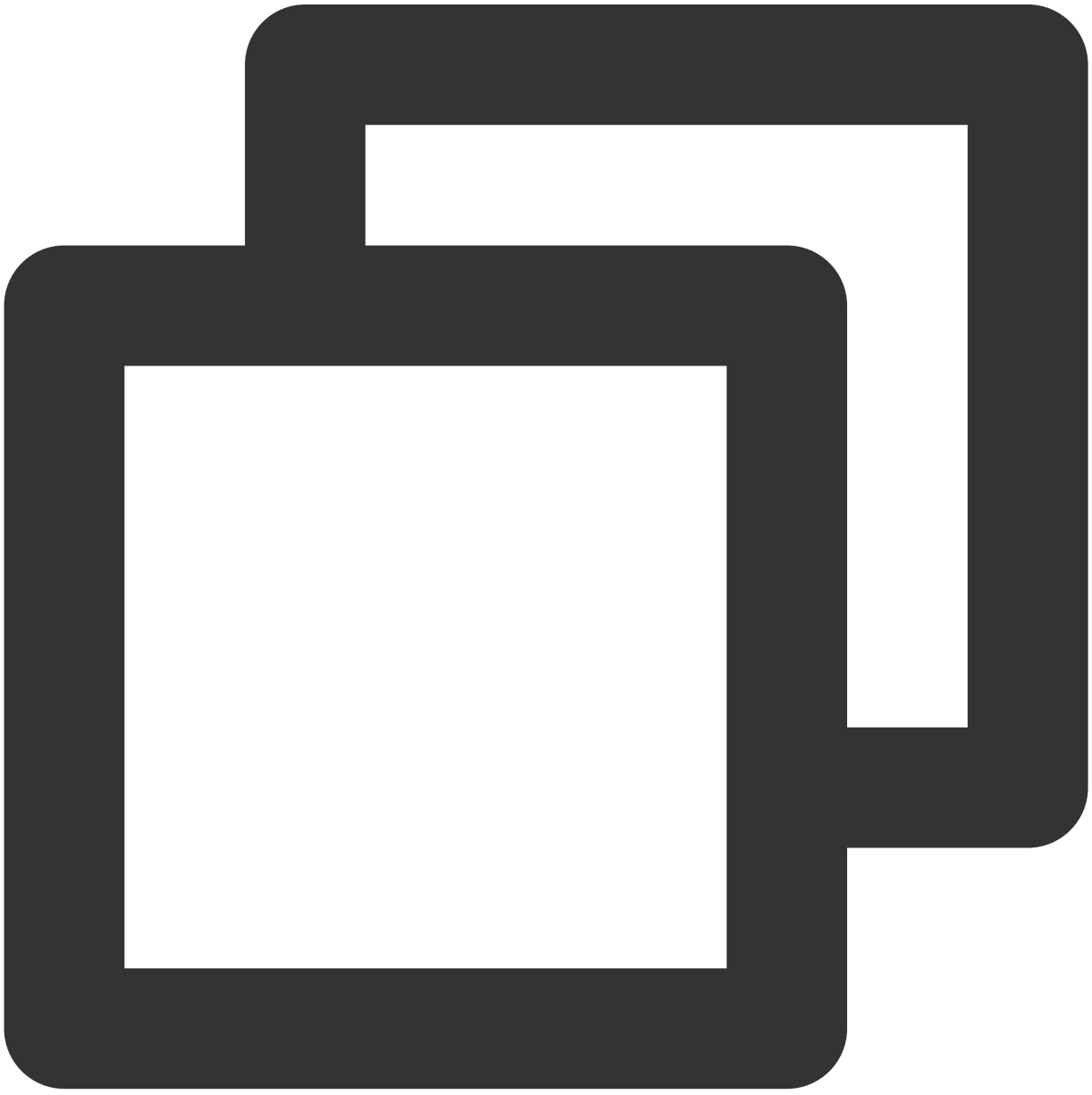
该钩子返回 false 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，可以用来过滤不希望上报的日志。

afterRequest

该勾子将会在测速数据上报后被执行，例如：

注意：

SDK 版本应大于等于 1.24.44。



```
const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx",
  afterRequest: function(msg) {
    // {isErr: false, result: Array(1), logType: "log", logs: Array(4)}
    console.log(msg);
  }
});
```

其中，msg 将会有以下几个字段：

1. isErr：请求上报接口是否错误。
2. result：上报接口的返回结果。
3. logs：上报的日志内容。
4. logType：日志类型，同 beforeRequest 中的 logType。

错误监控

最近更新时间：2024-01-22 19:39:30

前端性能监控的 Aegis 的实例会自动进行 JS 执行错误、Promise 执行错误、Ajax（Fetch）请求异常等监控。本文将为您介绍各错误监控逻辑及处理方式。

注意：

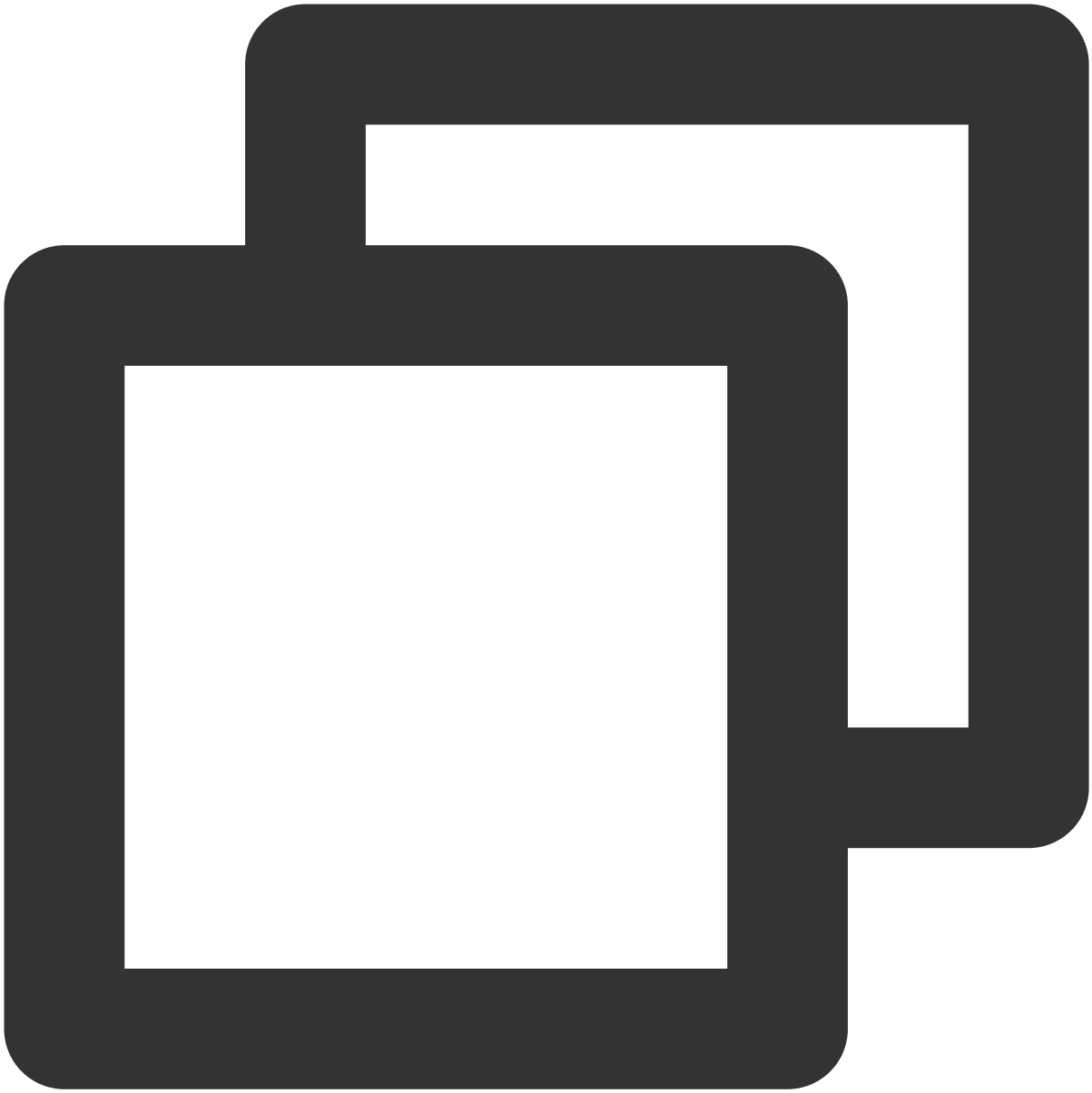
Aegis 实例会对这些异常进行监控，当您只是引入了 SDK 而没有将其实例化时，Aegis 将不会上报数据。

JS执行错误

Aegis 通过监听 `window` 对象上的 `onerror` 事件来获取项目中的报错，并且通过解析错误和分析堆栈，将错误信息自动上报到后台服务中。该上报的上报等级为 error，所以当自动上报的错误达到阈值时，Aegis 将会自动告警，帮助您尽早发现异常。由于上报等级为 error，自动上报也将影响项目的评分。

如果页面上引入了跨域的 JS 脚本，需要给对应的 `script` 标签添加 `crossorigin` 属性，否则 Aegis 将无法获取详细的错误信息。

如果用户使用的是 VUE 框架，请引入下列代码，获取错误并且主动上报。



```
Vue.config.errorHandler = function(err, vm, info) {
  console.log(`Error: ${err.toString()}\\nStack: ${err.stack}\\nInfo: ${info}`);
  aegis.error(`Error: ${err.toString()}\\nStack: ${err.stack}\\nInfo: ${info}`);
};
```

Promise 执行错误

通过监听 `unhandledrejection` 事件，捕获到未被 `catch` 的 Promise 错误，为了页面的稳定性，建议您 `catch` 住所有的 Promise 错误。

Ajax（Fetch）请求异常

Aegis 将会改写 `XMLHttpRequest` 对象，监听每次接口请求，Aegis 认为以下情况是异常情况：

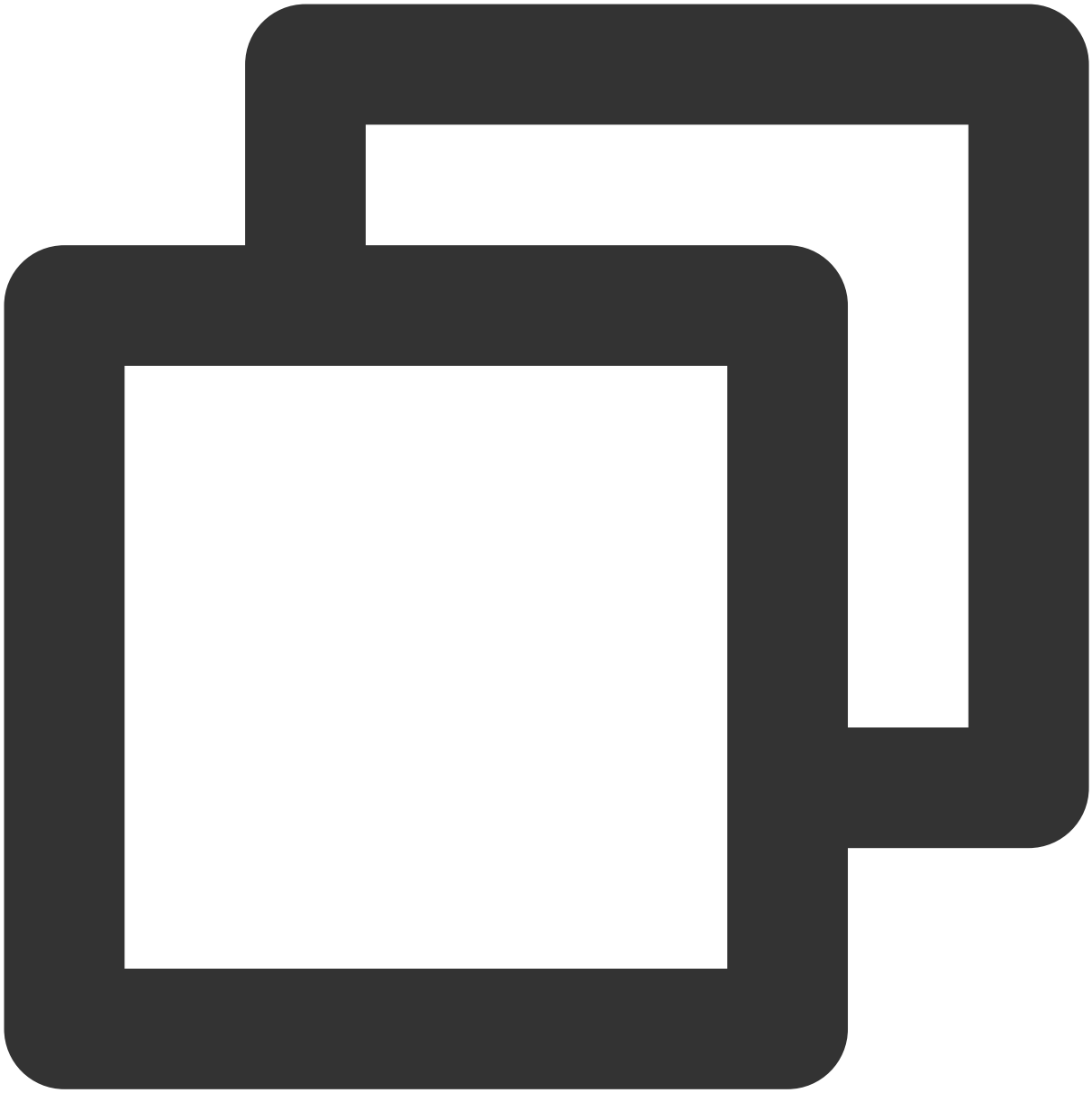
`http status` 大于等于 400

请求超时，abort，跨域，cancel

请求结束时 `http status` 仍然是 0，通常发生于请求失败

注意：

Aegis SDK 在错误发生的时候，不会主动收集接口请求参数和返回信息，如果需要对进口信息进行上报，可以使用 API 参数里面的 apiDetail 进行开启。



```
new Aegis({
  api: {
    apiDetail: true,
  },
});
```

retcode异常

Aegis 改写 `XMLHttpRequest` 对象之后，将获得 API 返回的内容，并尝试在内容中获取到本次请求的 `retcode`，当 `retcode` 不符合预期的时候，会认为本次请求出现了异常，并进行上报。

说明：

如何获取 `retcode` 以及哪些 `retcode` 是正常的，详情请参见 配置文档 。

资源加载失败

页面元素发出的请求如果失败，将会被 `window.onerror` 事件捕获到（捕获阶段），Aegis 正是通过这个特性监听的资源加载失败。Aegis 监听了以下资源：

`<link>` 标签请求的 css、font 等。

`<script>` 标签请求的脚本。

`<audio>` 、 `<video>` 标签请求的多媒体资源。

性能监控

最近更新时间：2024-01-22 19:39:30

您可以通过本文了解页面测速、接口测试、资源测速的统计方式和传入配置等信息。

页面测速

说明：
RUM 默认为您开启页面测速功能。
当您成功安装和初始化 SDK 后，Aegis 实例默认会上报以下指标：
1. **DNS 查询**：domainLookupEnd - domainLookupStart；
2. **TCP 连接**：connectEnd - connectStart；
3. **SSL 建连**：requestStart - secureConnectionStart；
4. **请求响应**：responseStart - requestStart；
5. **内容传输**：responseEnd - responseStart；
6. **DOM 解析**：domInteractive - domLoading；
7. **资源加载**：loadEventStart - domInteractive；
8. **首屏耗时**：监听页面打开3s内的 首屏 DOM 变化，并认为 DOM 变化数量最多的那一刻为首屏框架渲染完成时间（SDK 初始化后 setTimeout 3s 收集首屏元素，由于 JS 是在单线程环境下执行，收集时间点可能大于 3s）；
9. **页面完全加载时间**：为1-7项（DNS 查询、 TCP 连接、SSL 建连、请求响应、内容传输、DOM 解析、资源加载）时间总和；

说明：
1-7 项页面打开性能指标计算说明可从 PerformanceTiming 获取。首屏耗时对应的 DOM 元素，可以通过打印 aegis.firstScreenInfo 查看。如果 DOM 元素不能代表首屏，可以添加属性 `<div AEGIS-FIRST-SCREEN-TIMING></div>`，把某个元素识别为首屏关键元素，SDK 认为只要用户首屏出现此元素就是首屏完成。也可以添加属性 `<div AEGIS-IGNORE-FIRST-SCREEN-TIMING></div>`，把该 DOM 列入黑名单。
根据以上数据，前端性能监控为用户绘制了页面加载瀑布图
说明：
在服务端场景，瀑布图会出现首屏时间大于 DOM 解析的情况，这是由于移动端设备兼容性问题，有些设备无法获取到 DNS 查询、TCP 连接、SSL 建连时间，这三个指标汇总后的平均值偏小，导致除了首屏时间外的其他指标都往左偏移。

接口测速

说明：

打开方式：初始化时传入配置 `reportApiSpeed: true`

Aegis 通过劫持 `XHR` 及 `fetch` 进行接口测速。

资源测速

说明：

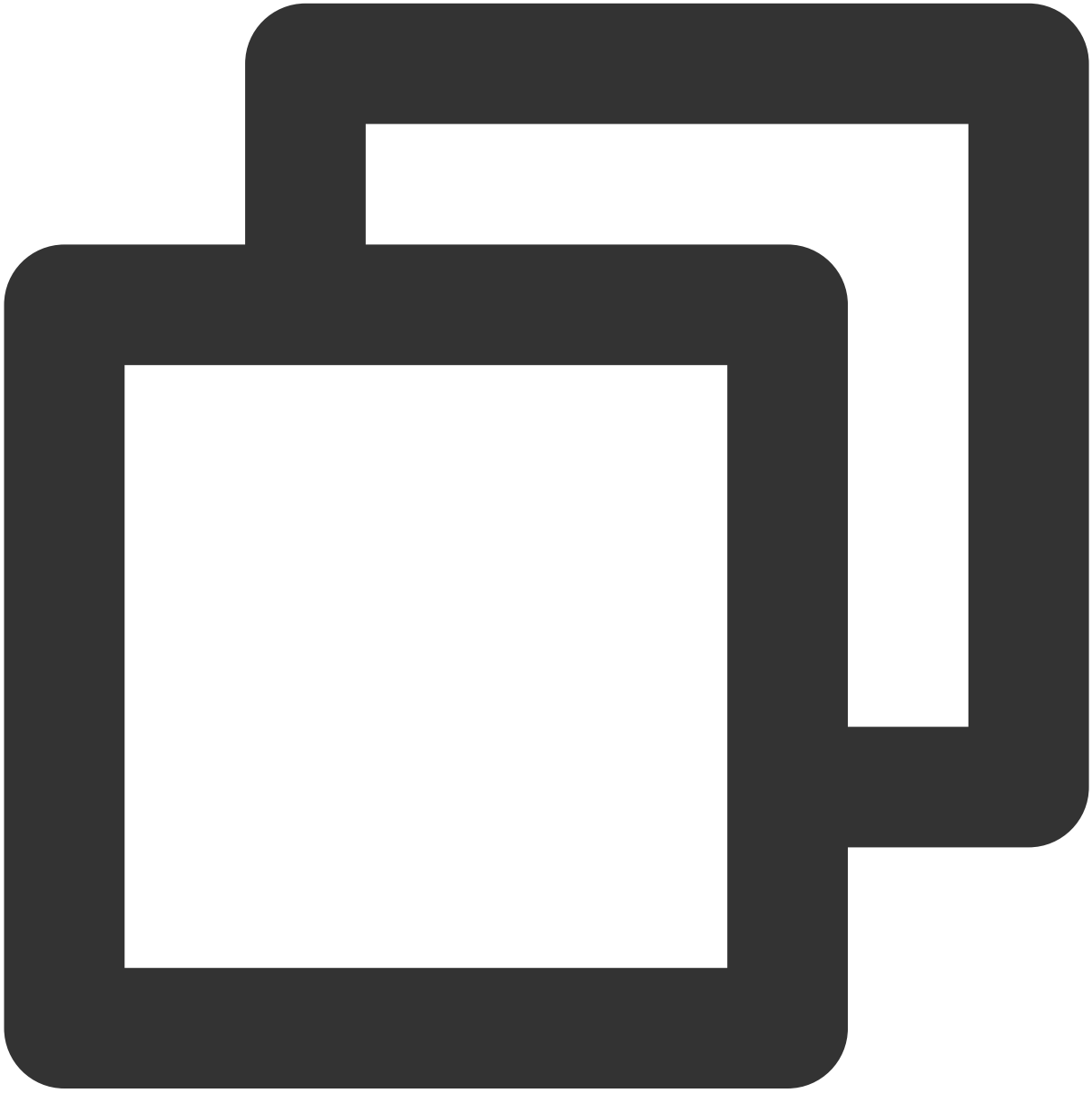
打开方式：初始化时传入配置 `reportAssetSpeed: true`

Aegis 通过浏览器提供的 `PerformanceResourceTiming` 进行资源测速。

环境控制

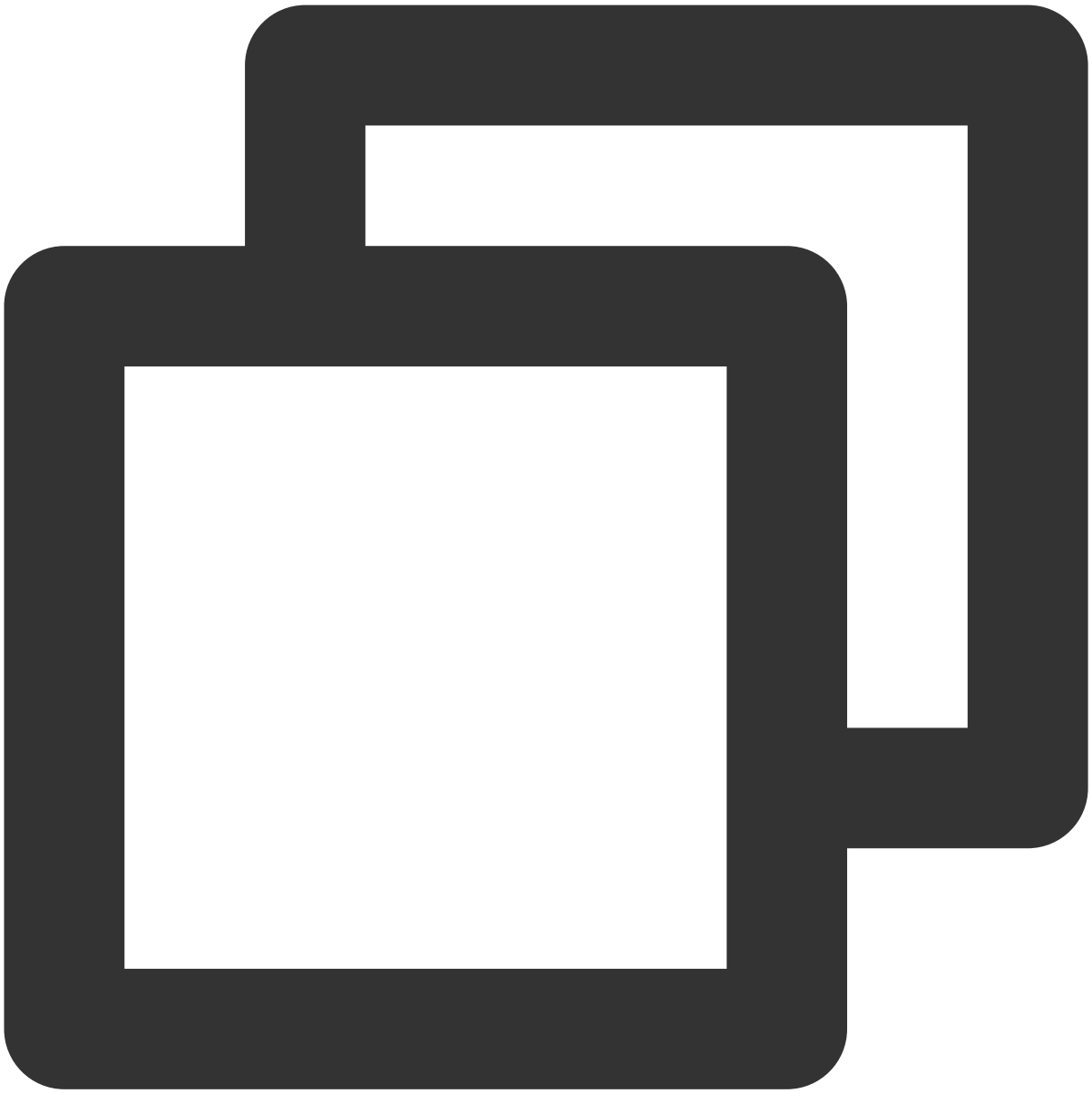
最近更新时间：2024-01-22 19:39:30

aegis 默认把数据所在环境当作 `production` 进行上报，如果需自行修改，可以使用 env 参数进行修改。



```
new Aegis({
  id: '',
  env: Aegis.environment.gray
})
```

Aegis.environment 枚举值如下：



```
export enum Environment {
  production = 'production', // 生产环境
  gray = 'gray', // 灰度环境
  pre = 'pre', // 预发布环境
  daily = 'daily', // 日发布环境
  local = 'local', // 本地环境
  test = 'test', // 测试环境
  others = 'others' // 其他环境
}
```

修改 env 参数后，aegis 上报的数据都会带上该参数，方便开发者区分不同环境的数据，但是只有 production 环境的数据会参与项目得分的计算。

配置文档

最近更新时间：2024-01-22 19:39:30

配置说明

配置文档各配置项说明如下：

配置	描述
id	必须，string，默认无。 前端性能监控分配的项目 ID。
uin	建议，string，默认取 cookie 中的 UIN 字段。 当前用户的唯一标识符，白名单上报时将根据该字段判定用户是否在白名单中，字段仅支持 `字母数字@=._-` ，正则表达式: `/^[@=.0-9a-zA-Z_-]{1,60}$/` 。
reportApiSpeed	可选，boolean 或者 object>，默认 false。 是否开启接口测速。
reportAssetSpeed	可选，boolean，默认 false 是否开启静态资源测速。
pagePerformance	可选，boolean 或者 object，默认 true。 是否开启页面测速。
webVitals	可选，boolean，默认 true。 是否开启 web vitals 测速。
onError	可选，boolean，默认 true。 当前实例是否需要进行错误监听，获取错误日志。
aid	可选，boolean，默认 true。 当前实例是否生成 aid。
random	可选，number，默认 1。 0 - 1 抽样率。
spa	可选，boolean，默认 false。 当前页面是否为单页应用？若为 true，则将监听 hashchange 及 history api，在页面跳转时进行 PV 上报。
version	可选，string，默认 SDK 版本号。 当前上报版本，当页面使用了 pwa 或者存在离线包时，可用来判断当前的上报是来自哪一个版本的代码，仅支持 `字母数字.,:_-` ，长度在 60 位以内 `/^[0-9a-zA-Z.,:_-]{1,60}$/` 。

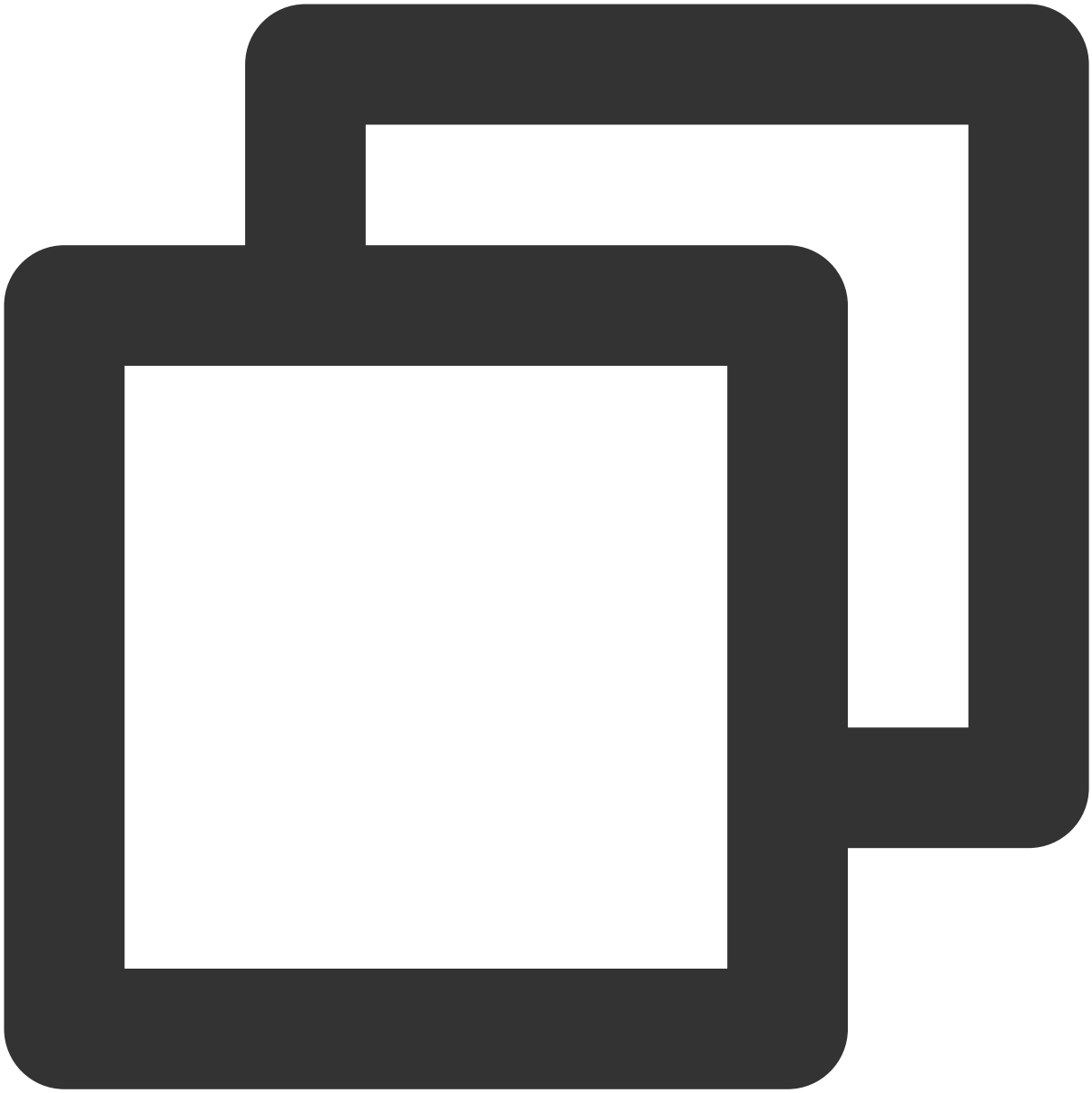
delay	可选，number，默认 1000ms。 上报节流时间，在该时间段内的上报将会合并到一个上报请求中。
repeat	可选，number，默认 5。 重复上报次数，对于同一个错误超过多少次不上报。
env	可选 enum，默认 Aegis.environment.production。当前项目运行所处的环境。 |
offlineLog	可选，boolean，默认 false。 是否使用离线日志。
offlineLogExp	可选，number，默认3。 离线日志过期天数。
api	可选，object，默认为{}。相关的配置： apiDetail：可选，boolean，默认：false。api 失败时，是否上报 api 的请求参数和返回值。 retCodeHandler：`Function(data: String, url: String, xhr: Object): {isErr: boolean, code: string}`，返回码上报钩子函数。会传入接口返回数据，请求 url 和 xhr 对象 `resourceTypeHandler: Function`，请求资源类型修正钩子函数会传入接口 url，返回值为 `static` 或 `fetch`。详情请参见示例 api.retCodeHandler。 reportRequest：boolean，默认：false。开启后，aegis.info 会变成全量上报，不需要白名单配置，并且会上报所有接口的信息（上报接口需开启 reportApiSpeed）。
speedSample	可选，boolean，默认 true。 测速日志是否抽样（限制每条 url 只上报一次测速日志）。
hostUrl	可选，默认是 `//aegis.qq.com`。 影响全部上报数据的 host 地址，下面几个 URL 地址设置后会覆盖对应的上报地址。
url	可选，string，默认 `//aegis.qq.com/collect`。 日志上报地址。 设置为空字符串可以不进行日志上报。
pvUrl	可选，string，默认 `//aegis.qq.com/collect/pv`。 pv 上报地址。 设置为空字符串可以不进行 pv 上报。
whiteListUrl	可选，string，默认 `//aegis.qq.com/collect/whitelist`。 白名单确认接口。 设置为空字符串可以关闭白名单接口请求
offlineUrl	可选，string，默认 `//aegis.qq.com/collect/offline`。 离线日志上报地址。 设置为空字符串可以不进行离线日志上报。

eventUrl	可选，string，默认 `//aegis.qq.com/collect/events` 。 自定义事件上报地址。 设置为空字符串可以不进行自定义事件上报。
speedUrl	可选，string，默认 `//aegis.qq.com/speed` 。 测速日志上报地址。 设置为空字符串可以不进行测速数据上报。
customTimeUrl	可选，string，默认 `//aegis.qq.com/speed/custom` 。 自定义测速上报地址。 设置为空字符串可以不进行自定义测速上报。
performanceUrl	可选，string，默认 `//aegis.qq.com/speed/performance` 。 页面性能日志上报地址。 设置为空字符串可以不进行页面性能上报。
webVitalsUrl	可选，string，默认 `//aegis.qq.com/speed/webvitals` 。 webvitals 上报地址。 设置为空字符串可以不进行web-vitals上报。
dbConfig	可选，Object。
ext1	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext2	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext3	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。

示例

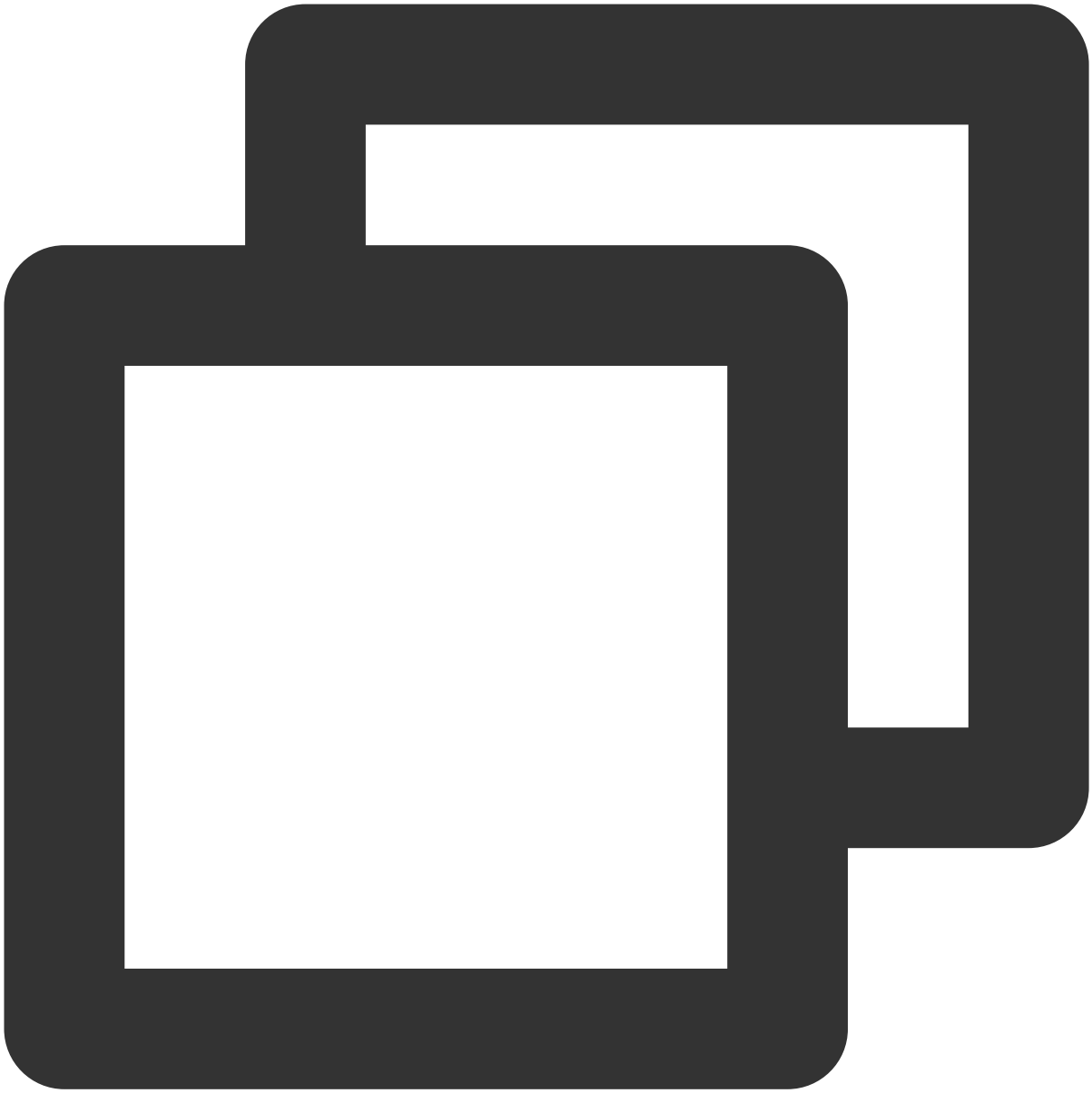
api.retCodeHandler

假如后台返回数据为：



```
{
    body: {
        code: 200,
        retCode: 0,
        data: {
            // xxx
        }
    }
}
```

业务需要：code 不为200，或者 retCode 不为0，此次请求就是错误的。此时只需进行以下配置：

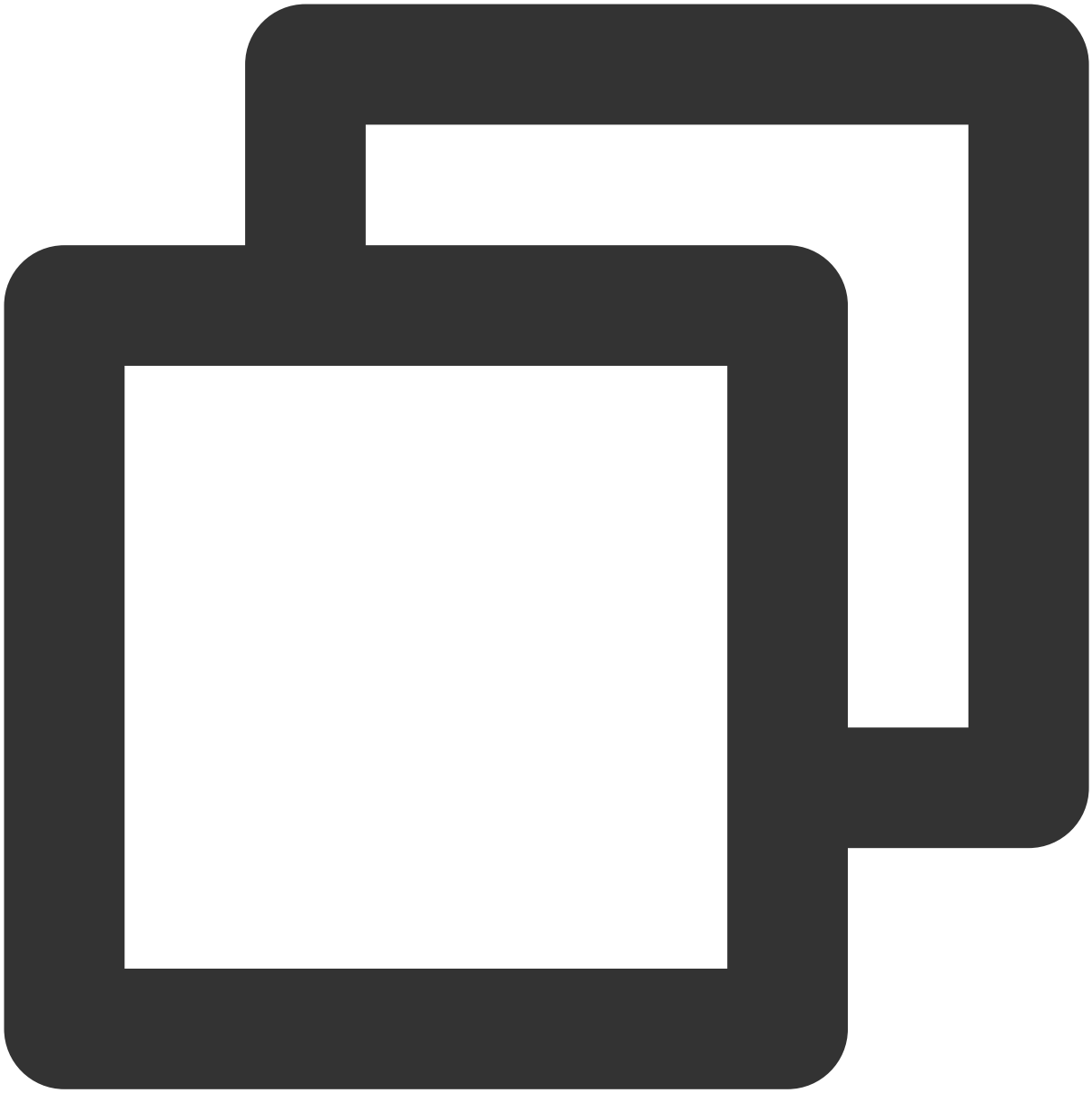


```
new Aegis({
    // xxx
    reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
    reportAssetSpeed: true,
    api: {
        retCodeHandler(data, url, xhr) {
            // data 是string类型，如果需要对象需要手动parse下
            // url 为请求url
            // xhr 响应,可以通过xhr.response拿到完整的后台响应
            try {
                data = JSON.parse(data)
```

```
            } catch(e) {}
            return {
                isErr: data.body.code !== 200 || data.body.retCode !== 0,
                code:  data.body.code
            }
        }
    }
})
```

api.resourceTypeHandler

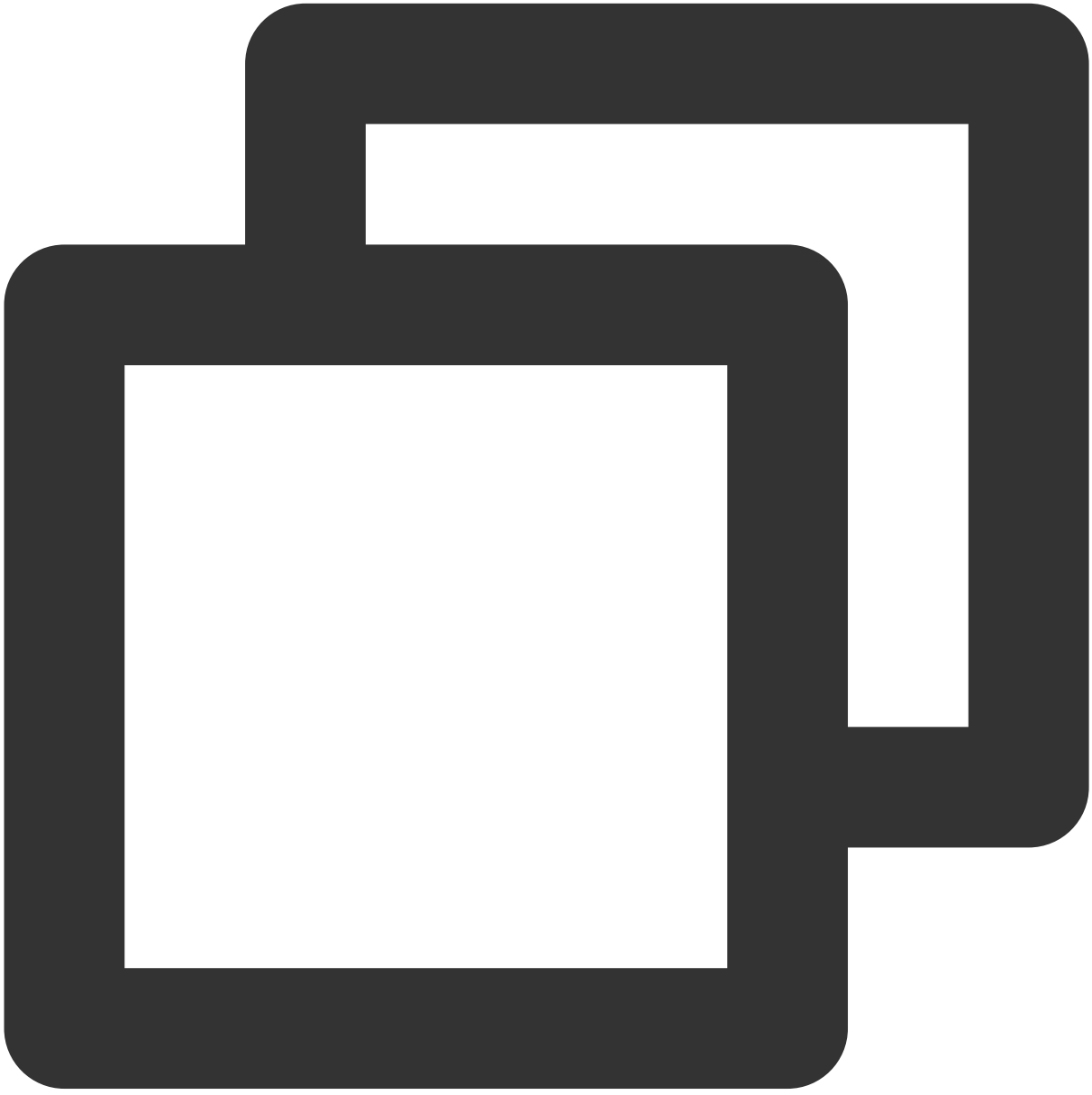
假如接口为：`http://example.com/test-api` 。返回的 `Content-Type` 为 `text/html`，这将导致 Aegis 认为该接口返回的是静态资源，可以通过以下方法修正：



```
new Aegis({
    reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
    reportAssetSpeed: true,
    api: {
        resourceTypeHandler(url) {
            if (url?.indexOf('http://example.com/test-api') != -1) {
                return 'fetch';
            }
        }
    }
})
```

reportApiSpeed.urlHandler

假如您页面中有 restful 风格的接口，例
如：`www.example.com/user/1000` `www.example.com/user/1001`
在上报测速时需要将这些接口聚合：

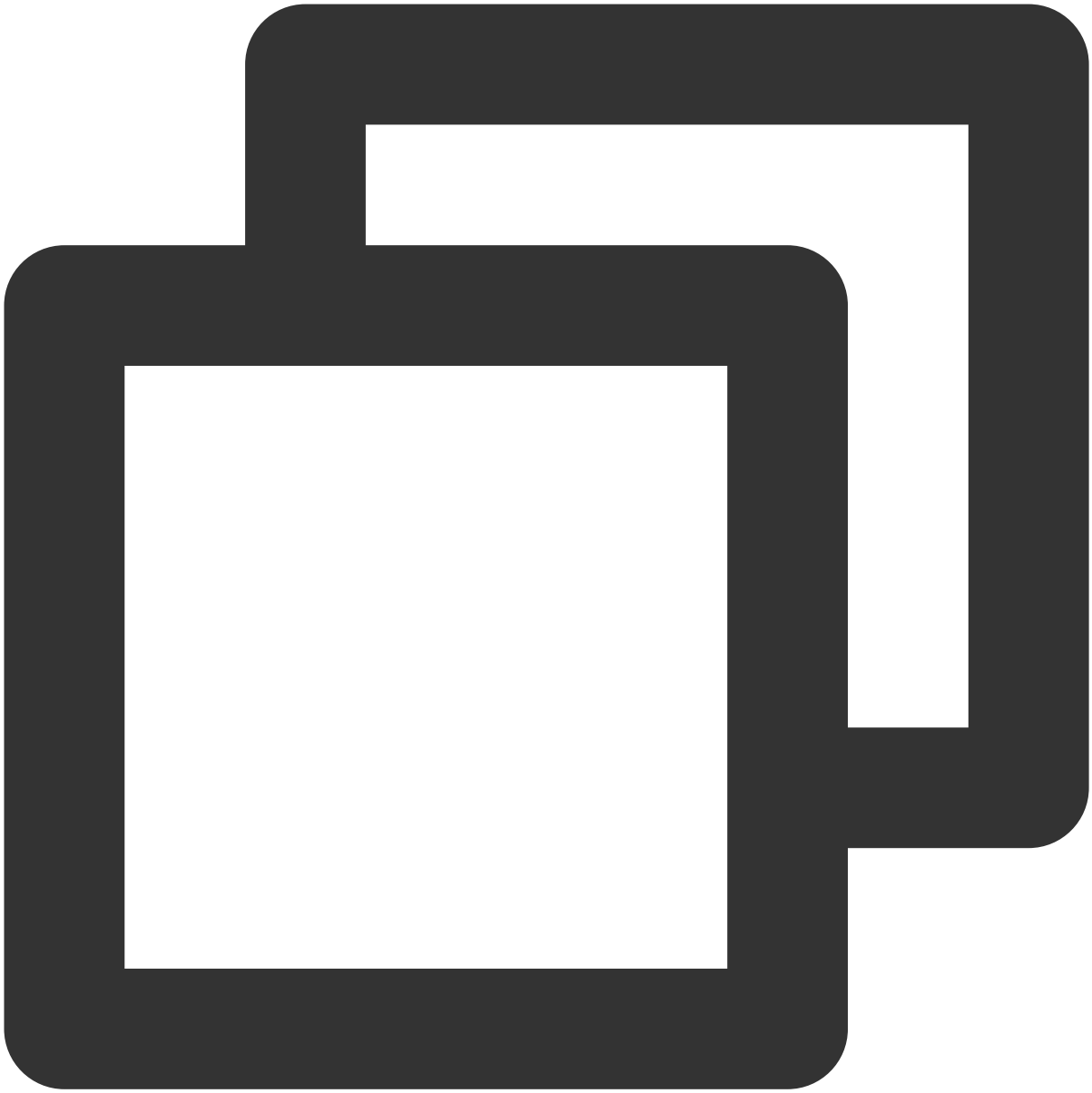


```
new Aegis({
    // xxx
    reportApiSpeed: {
        urlHandler(url, payload) {
```

```
            if ((/www\\.example\\.com\\/user\\/\\d*/).test(url)) {
                return 'www.example.com/user/:id';
            }
            return url;
        }
    }
    // xxx
})
```

pagePerformance.urlHandler

假如您的页面 URL 是 restful 风格的，例如：`www.example.com/user/1000` `www.example.com/user/1001`
在上报页面测速时需要将这些页面地址聚合：



```
new Aegis({
    // xxx
    pagePerformance: {
        urlHandler() {
            if ((/www\\.example\\.com\\/user\\/\\d*/).test(window.location.href)) {
                return 'www.example.com/user/:id';
            }
        }
    }
    // xxx
})
```

小程序场景
安装和初始化

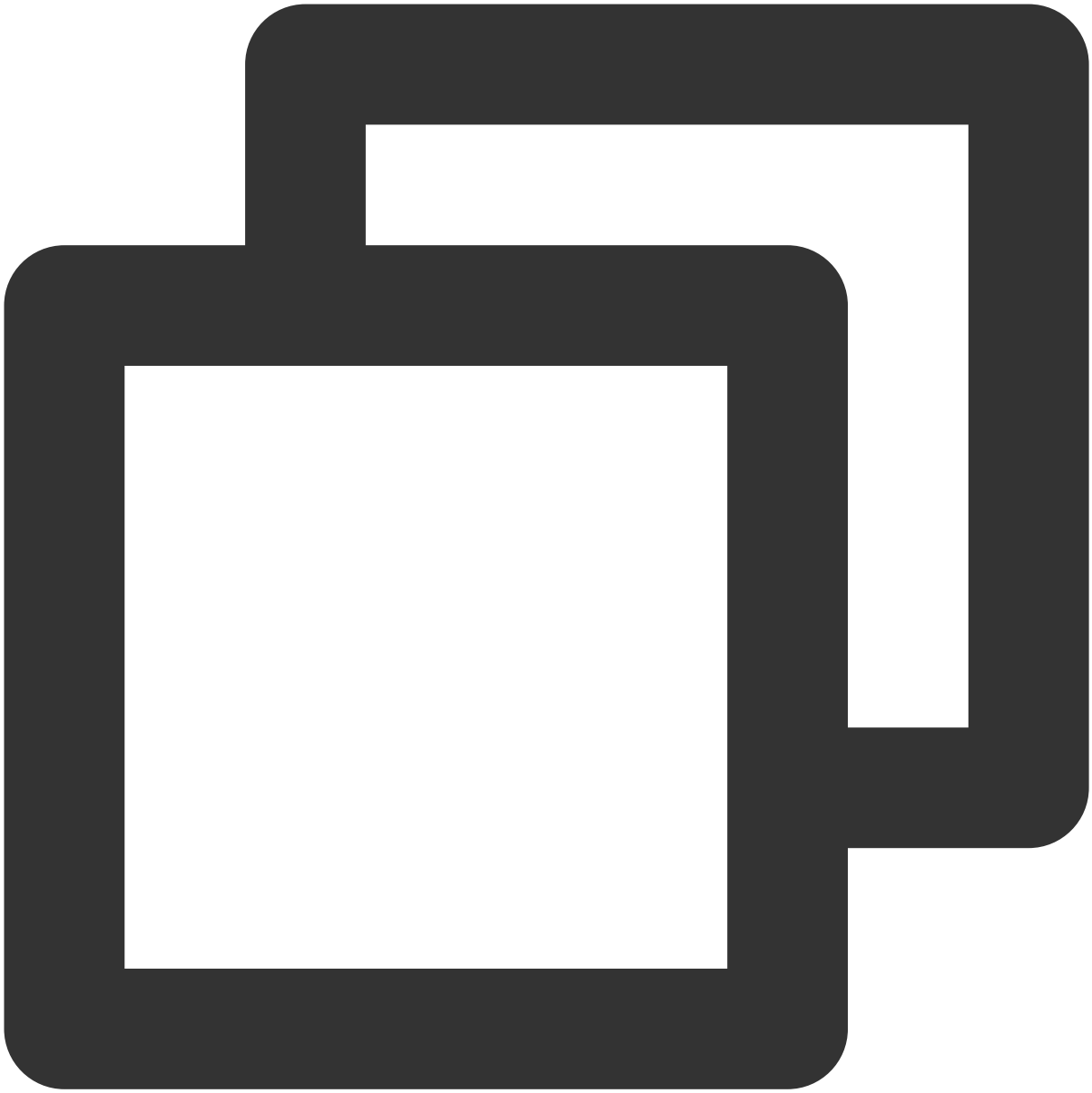
最近更新时间：2024-01-22 19:39:30

注意事项

小程序仅支持 NPM 方式安装 SDK 。
本 SDK 支持微信小程序和 QQ 小程序。
正式环境接入小程序需要将上报域名添加到安全域名中。
说明：
aegis-sdk 默认使用 `https://aegis.qq.com` 作为上报域名，您可以通过 `hostUrl` 参数来控制上报域名。
国内可以选择选择使用 `https://tamaegis.com` 作为上报域名。
国外新加坡地区可以选择 `https://rumt-sg.com` 作为上报域名。

安装 SDK

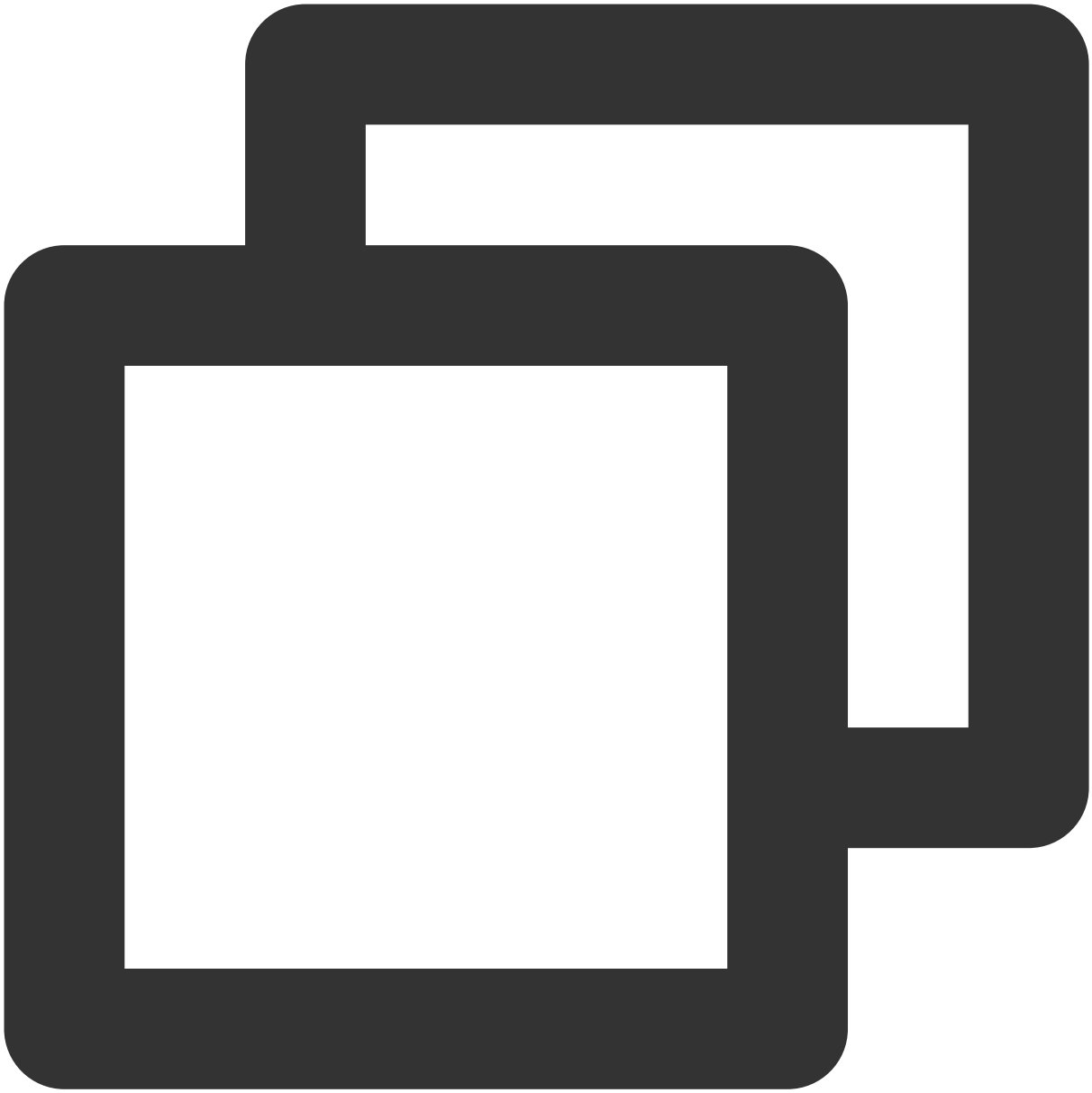
执行下列命令，在 npm 仓库安装 aegis-mp-sdk。



```
$ npm install --save aegis-mp-sdk
```

初始化

参考下列步骤新建一个 Aegis 实例，传入相应的配置，初始化 SDK



```
import Aegis from 'aegis-mp-sdk';

const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx", // 项目key
  uin: 'xxx', // 用户唯一 ID（可选）
  reportApiSpeed: true, // 接口测速
  spa: true, // 页面切换的时候上报 pv
});
```

说明：

为了不遗漏数据，须尽早进行初始化。如果您的小程序项目中使用了 `miniprogram-api-promise` 来封装 `wx.request` 请求，需要注意：由于我们是通过重写 `wx.request` 来进行接口监控的，所以需要您将初始化 Aegis 放在引入这个包之前执行，否则可能会导致接口信息无法完整收集。

当您完成安装并初始化 SDK 之后，可以开始使用前端性能监控提供的以下功能：

1. 错误监控：JS执行错误。
2. 测速功能：接口测速。
3. 数据统计和分析：可在前端性能监控控制台 **数据总览** 页面上进行各个维度的数据分析。

日志上报

最近更新时间：2024-01-22 19:39:30

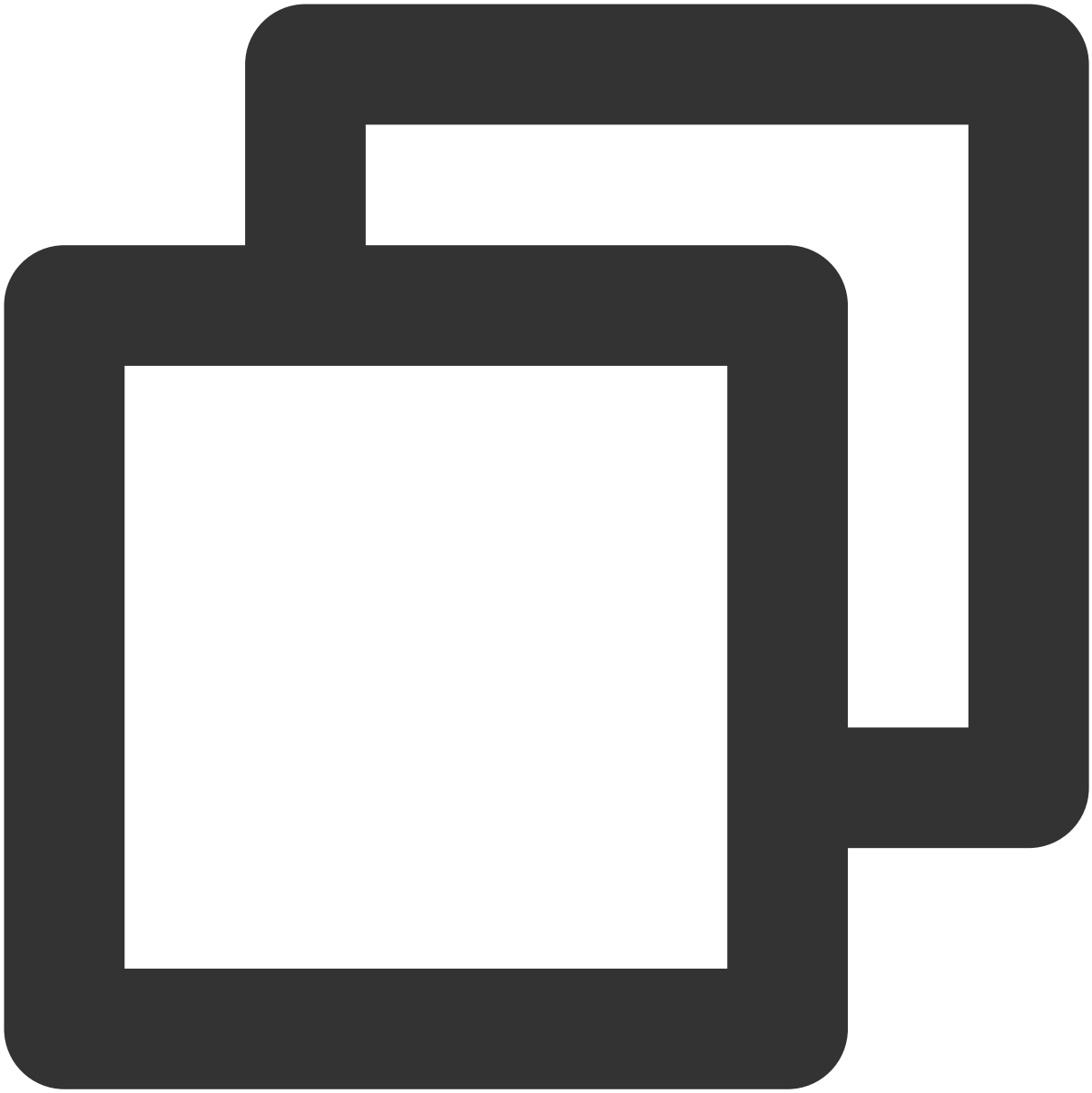
日志查询功能需您上报日志后，才能正常使用。本文将为您介绍如何上报日志到前端性能监控。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

日志上报

引入下列参数，配置 SDK ，完成日志上报。



```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
```

```
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.LogType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

aid

最近更新时间：2024-01-22 19:39:31

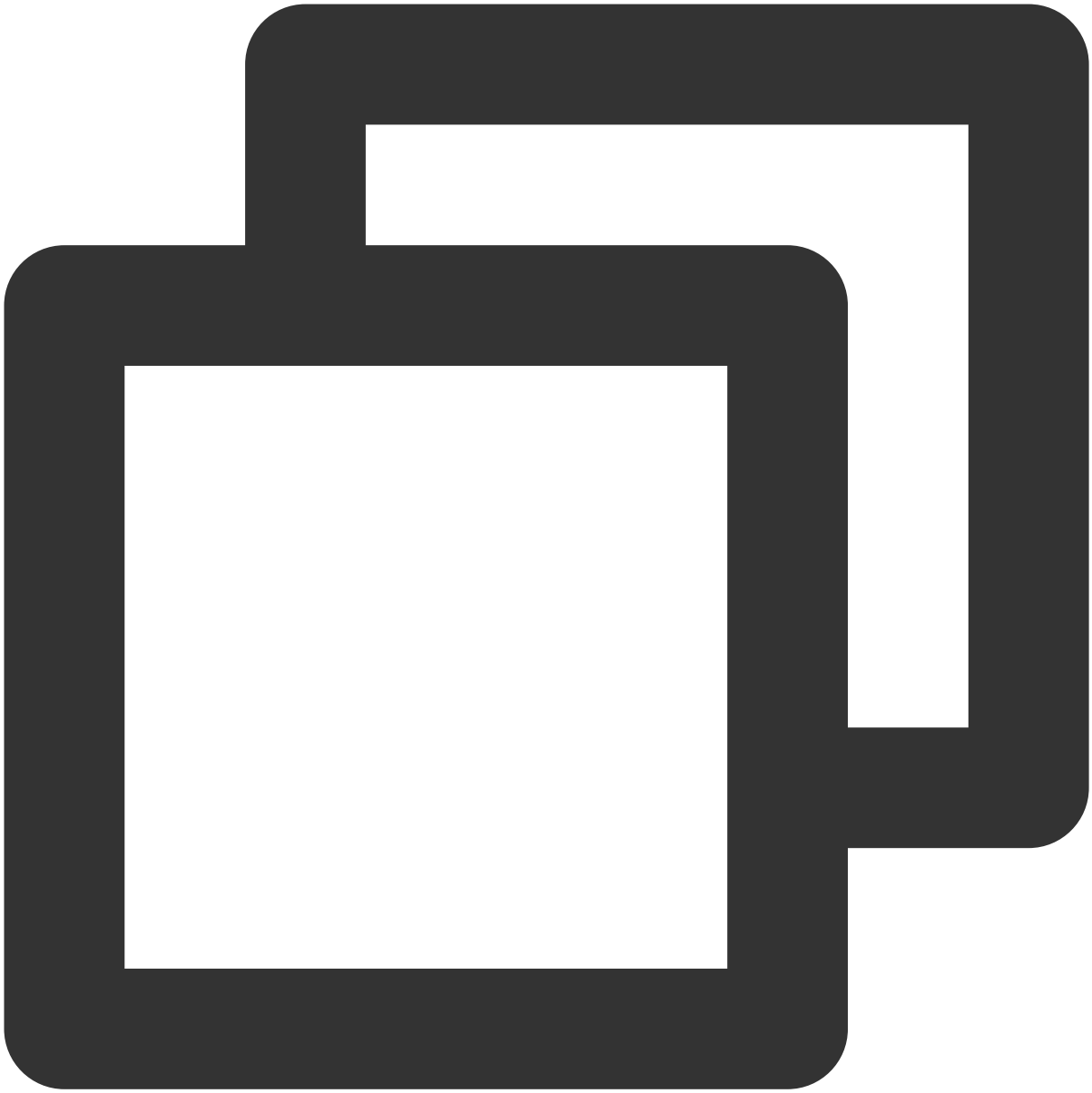
aid 是 Aegis SDK 为每个用户设备分配的唯一标识，存储在小程序的 storage 。用于区分用户计算 UV 等。只有用户清理小程序缓存 aid 才会更新。

前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

aid

您可以根据下列算法自定义 aid 上报规则：



```
aid = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
const r = (Math.random() * 16) | 0;
const v = c === 'x' ? r : (r & 0x3) | 0x8;
return v.toString(16);
```

实例方法

最近更新时间：2024-01-22 19:39:30

前端性能监控为您提供多种实例方法用于上报数据，您可以通过实例方法修改实例配置、自定义上报事件、自定义上报测试资源等。
目前 RUM 提供的 Aegis 实例方法如下：

参数	用途
setConfig	传入配置对象，包括用户 ID 和 UIN 等信息
info	主要上报字段，用于上报白名单日志。下列两种情况日志才会报到后台： 1. 打开页面的用户在名单中。 2. 对应的页面发生了错误。
infoAll	主要上报字段，用于上报白名单日志。该上报与 info 唯一的区别： info 指定用户上报。
error	主要上报字段，用于上报错误信息。
report	用来上报任意类型的日志信息。
reportEvent	上报自定义事件。
reportTime	上报自定义测速资源。
time	上报自定义测速资源，与 timeEnd 共同使用。适用于两个时间点之间时长的计算并上报。
timeEnd	上报自定义测速资源，与 time 共同使用。适用于两个时间点之间时长的计算并上报。
destroy	销毁aegis实例。

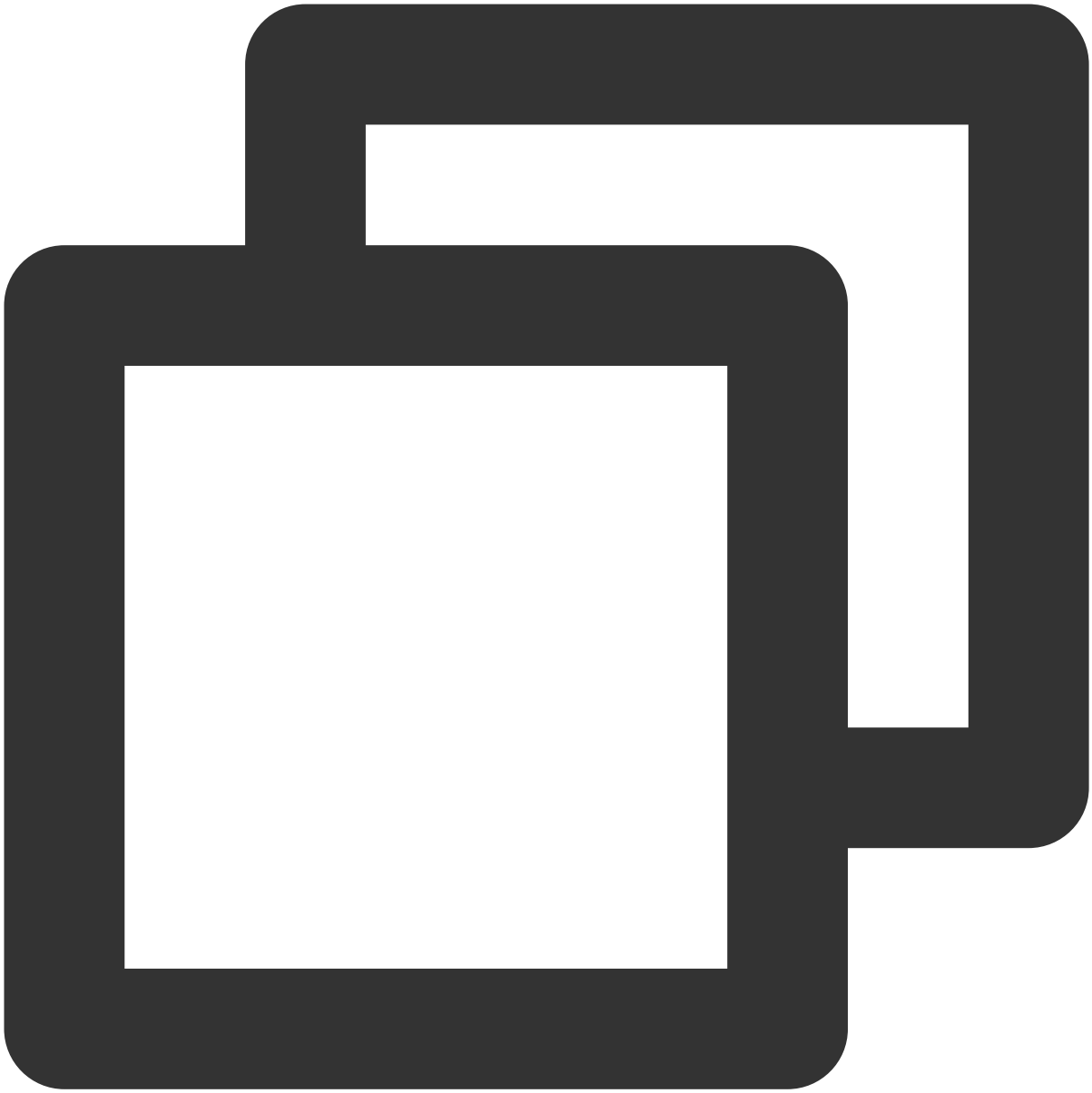
前提条件

参见 安装和初始化 文档，选择任意一种方式完成前端性能监控 SDK 的安装和初始化。

实例方法

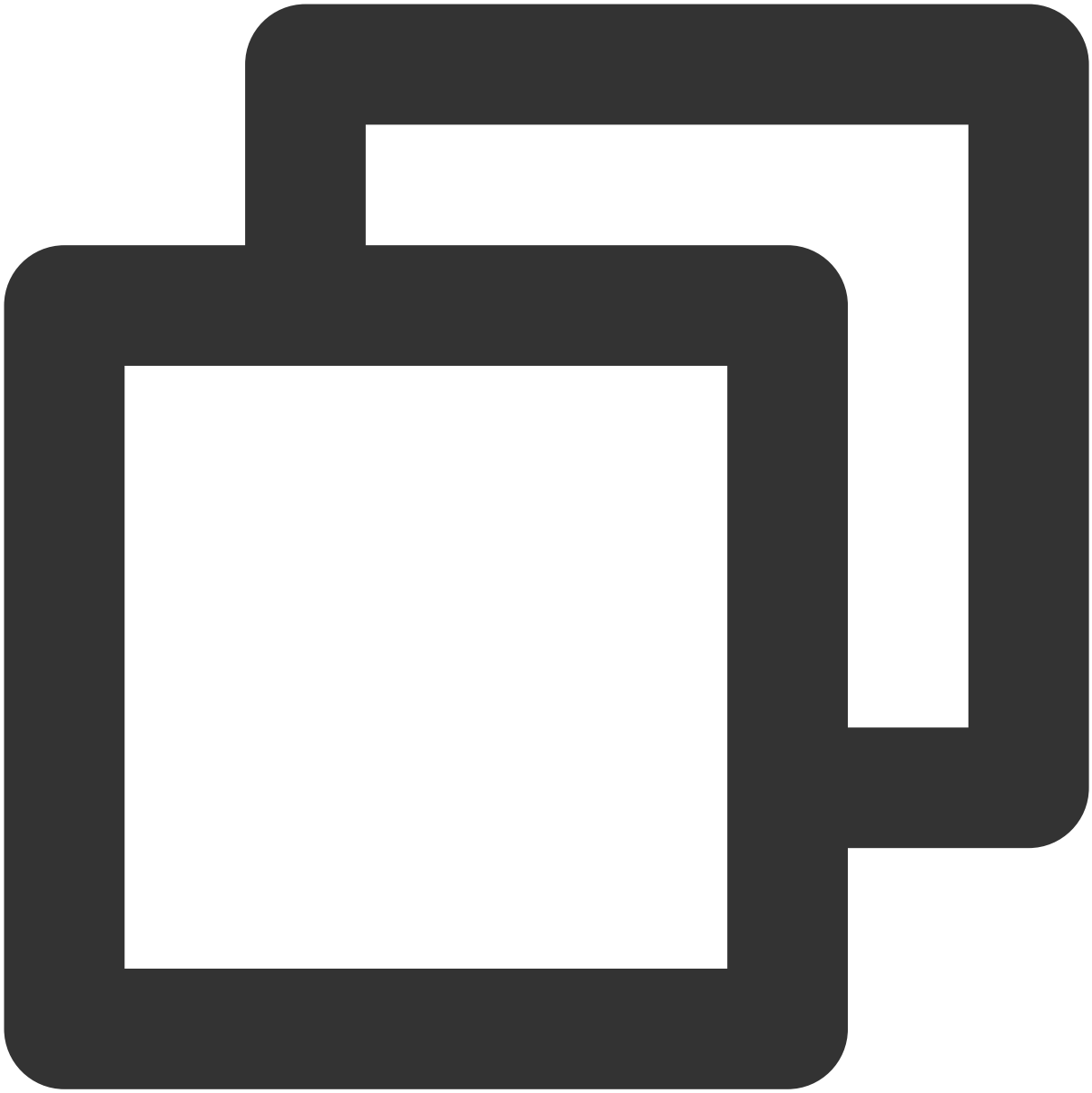
setConfig

该方法用于修改实例配置，使用场景如下：
1. 可获取到用户 UIN ，可同时传入配置用户 ID 和 UIN 两个实例对象，进行实例化：



```
const aegis = new Aegis({
    id: 'pGUVFTCZyewxxxxx',
    uin: '777'
})
```

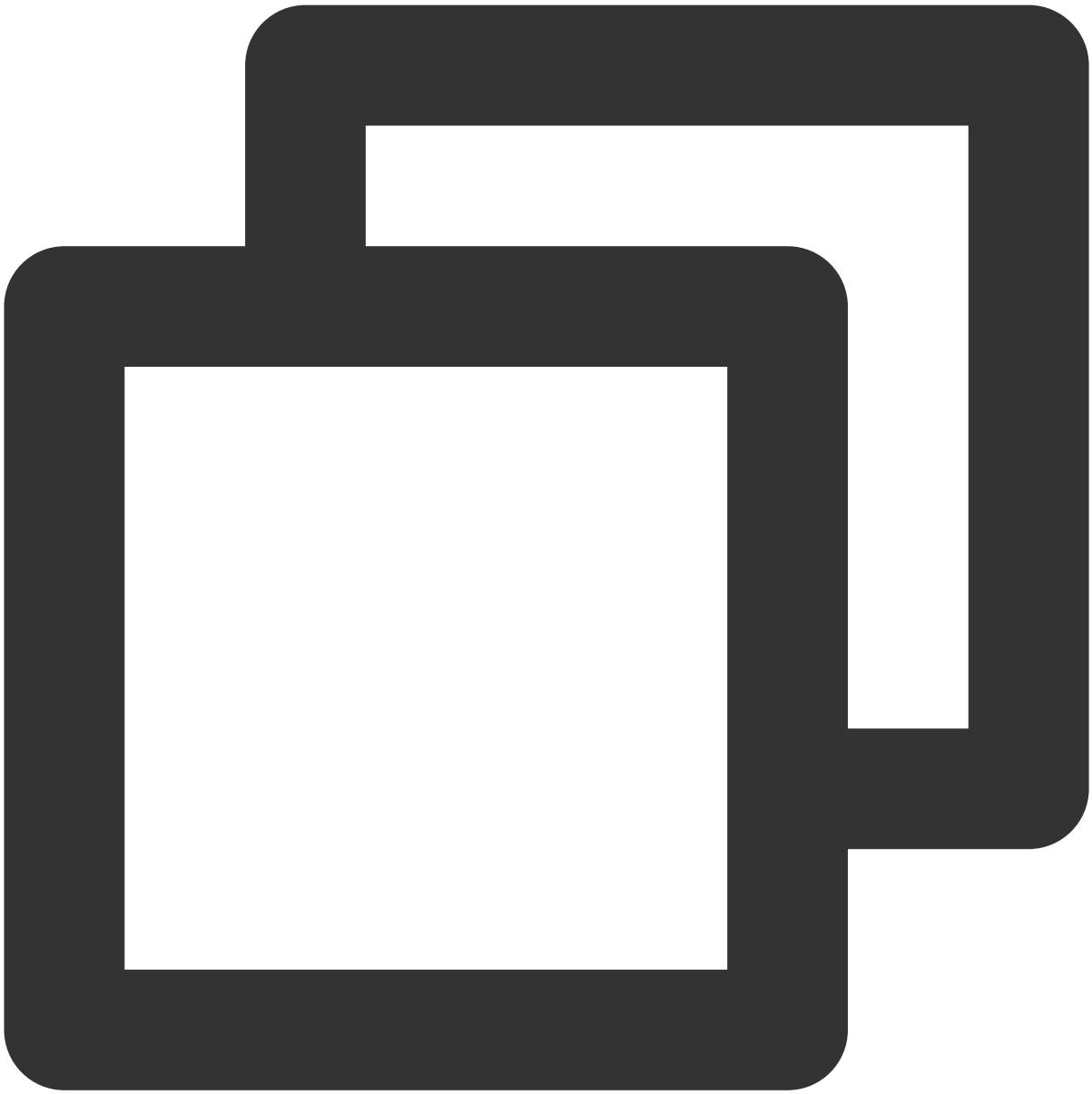
2. 通常情况下，我们并不能一开始就获取到用户的 `uin` 。若在获取 UIN 的这段时间不进行实例化，这期间发生的错误前端性能监控将无法监听。针对这种情况，我们可以先传入 ID 进行实例化，引用 setConfig 传入 UIN ，示例如下：



```
const aegis = new Aegis({
    id: 'pGUVFTCZyewxxxxx'
})

// 拿到uin之后...
aegis.setConfig({
  uin: '6666'
})
```

info、infoAll、error 和 report

这三个方法是前端性能监控提供的主要上报手段。



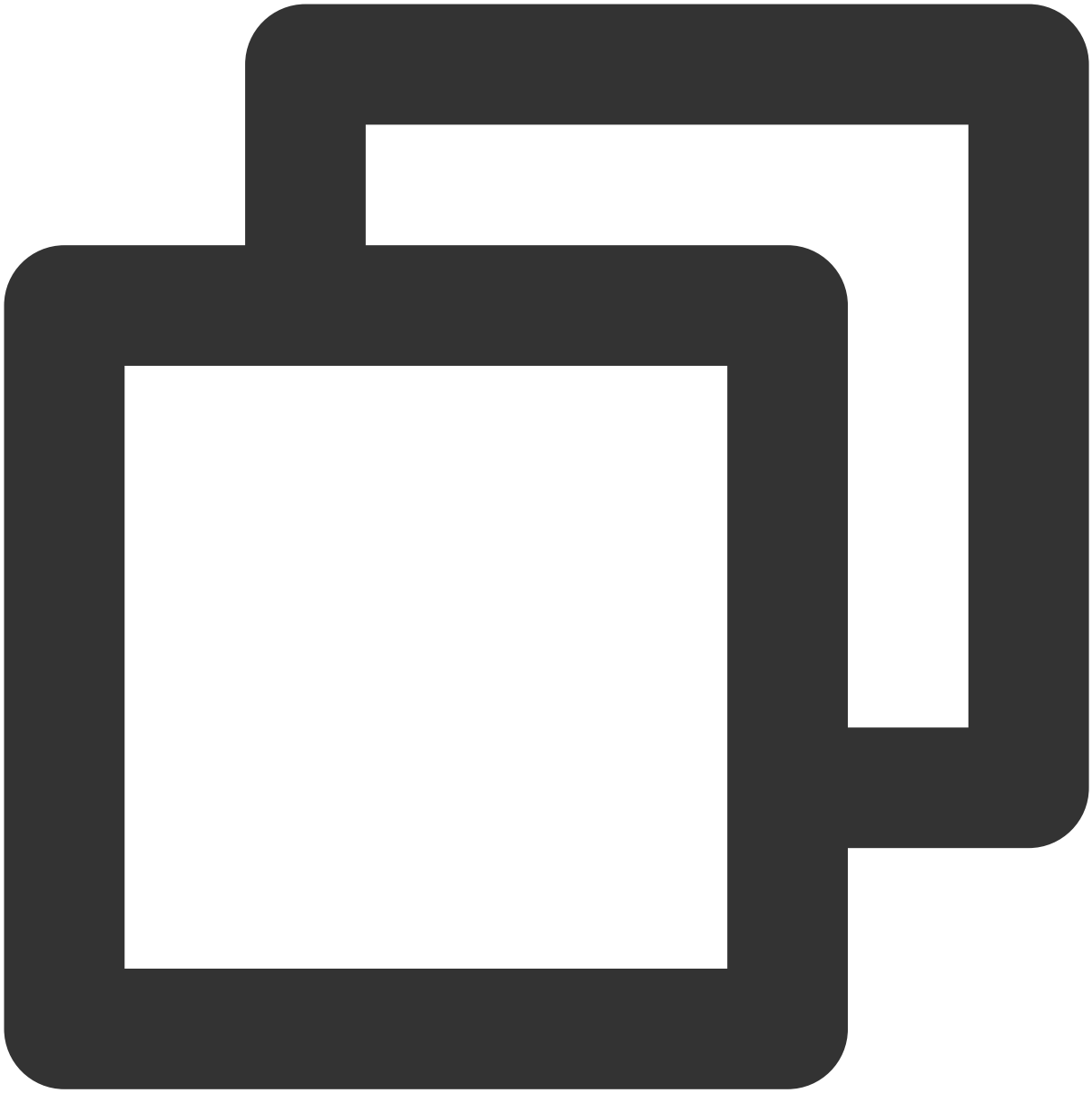
```
// info 可以上报任意字符串，数字，数组，对象，但是只有打开页面的用户在名单中才会上报
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// 也可以上报特定的对象，支持用户传ext参数和trace参数
// 注意这种 case 一定要传 msg 字段
aegis.info({
 msg: 'test',
 ext1: 'ext1',
```

```
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// 不同于 info，infoAll 表示全量上报
aegis.infoAll({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});

// error 用来表示 JS 错误日志，也是全量上报，一般用于开发者主动获取JS异常，然后进行上报
aegis.error({
 msg: 'test',
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
aegis.error(new Error('主动上报一个错误'));

// report 默认是 aegis.report 的日志类型，但是现在您可以传入任何日志类型了
aegis.report({
 msg: '这是一个ajax错误日志',
 level: Aegis.LogType.AJAX_ERROR,
 ext1: 'ext1',
 ext2: 'ext2',
 ext3: 'ext3',
 trace: 'trace',
});
```

reportEvent

该方法可用来上报自定义事件，系统将会自动统计上报事件的各项指标，例如：PV、平台分布等。
reportEvent 可以支持字符串和对象两种类型上报参数。

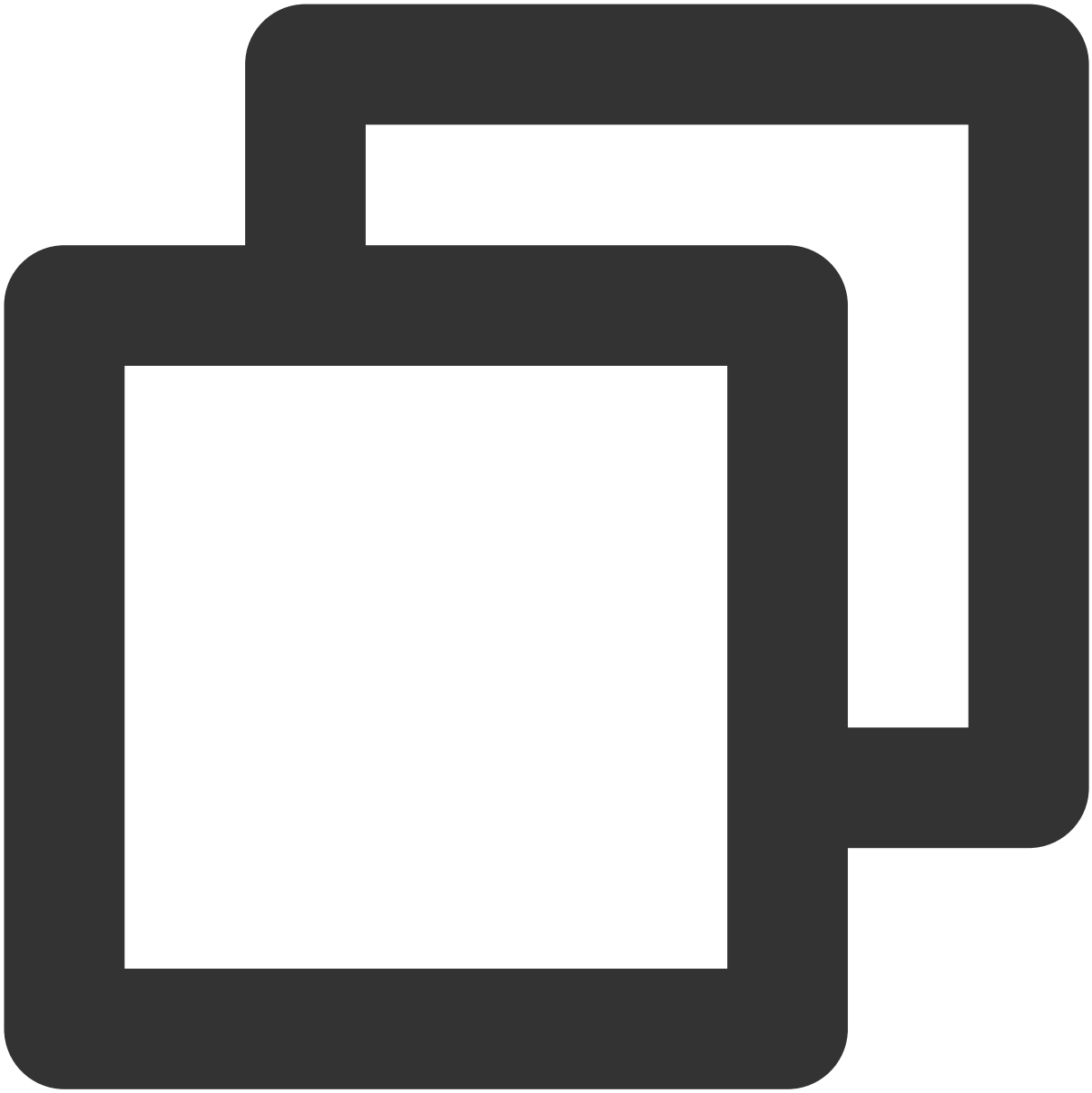
字符串类型



```
aegis.reportEvent('XXX请求成功');
```

对象类型

ext1、ext2 和 ext3 默认使用 new Aegis 的时候传入的参数，自定义事件上报的时候，可以覆盖默认值。



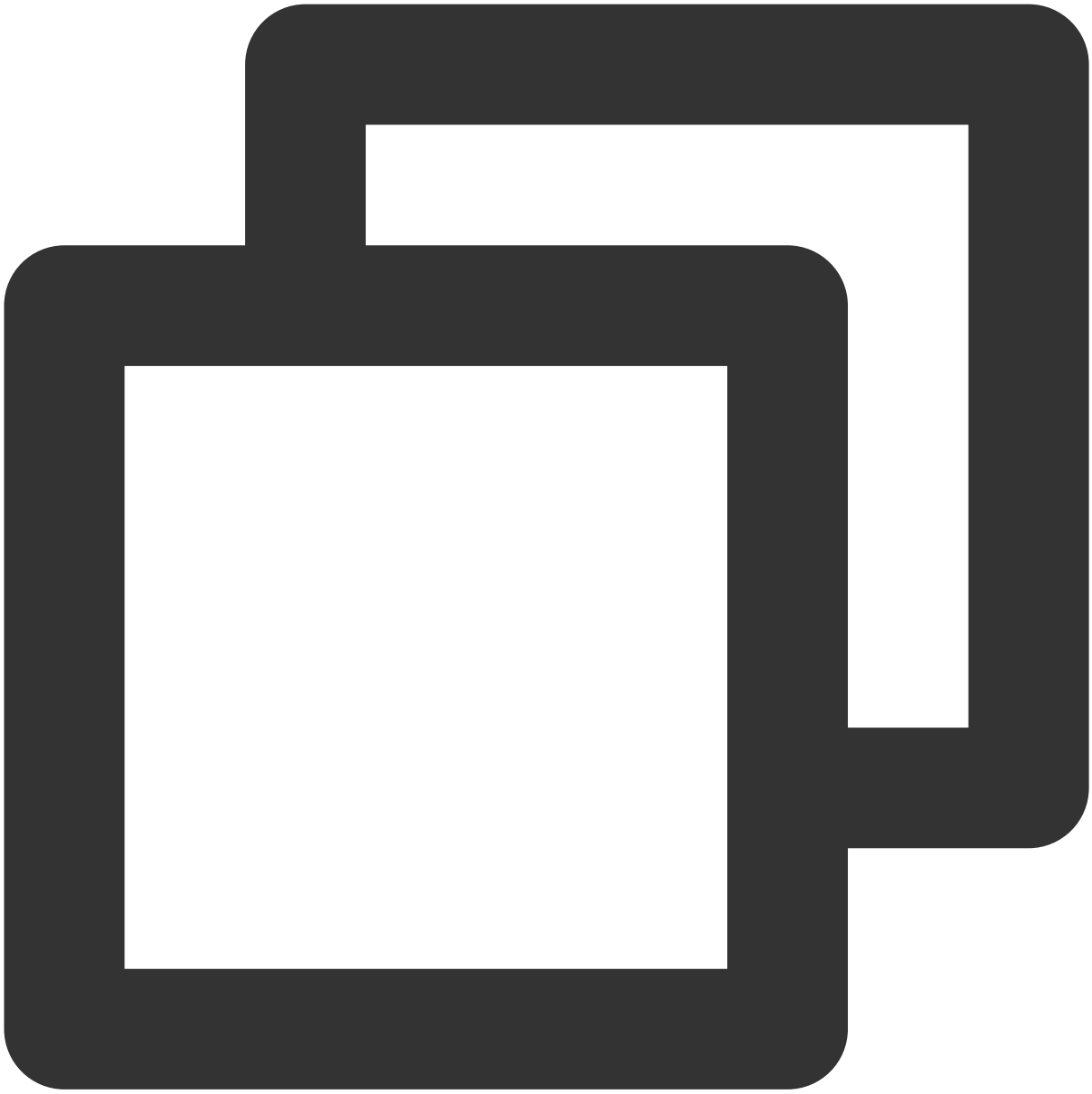
```
aegis.reportEvent({
  name: 'XXX请求成功', // 必填
  ext1: '额外参数1',
  ext2: '额外参数2',
  ext3: '额外参数3',
})
```

注意：

额外参数的三个 key 是固定的，目前只支持 ext1、ext2 和 ext3 。

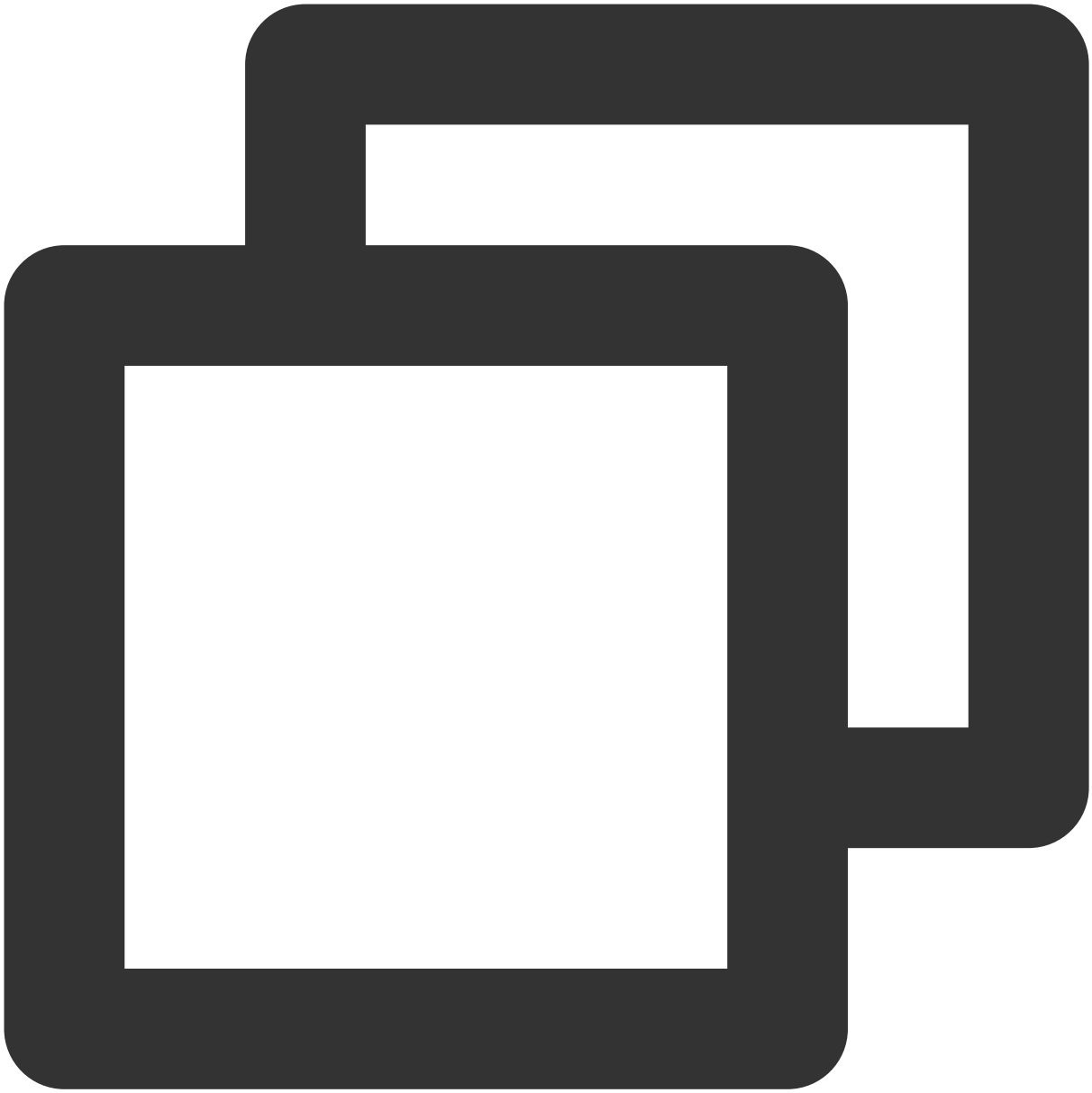
reportTime

该方法可用来上报自定义测速，例如：



```
// 假如‘onload’的时间是1s
aegis.reportTime('onload', 1000);
```

或者如果需要使用额外参数，可以传入对象类型参数，ext1，ext2，ext3 会覆盖默认值：



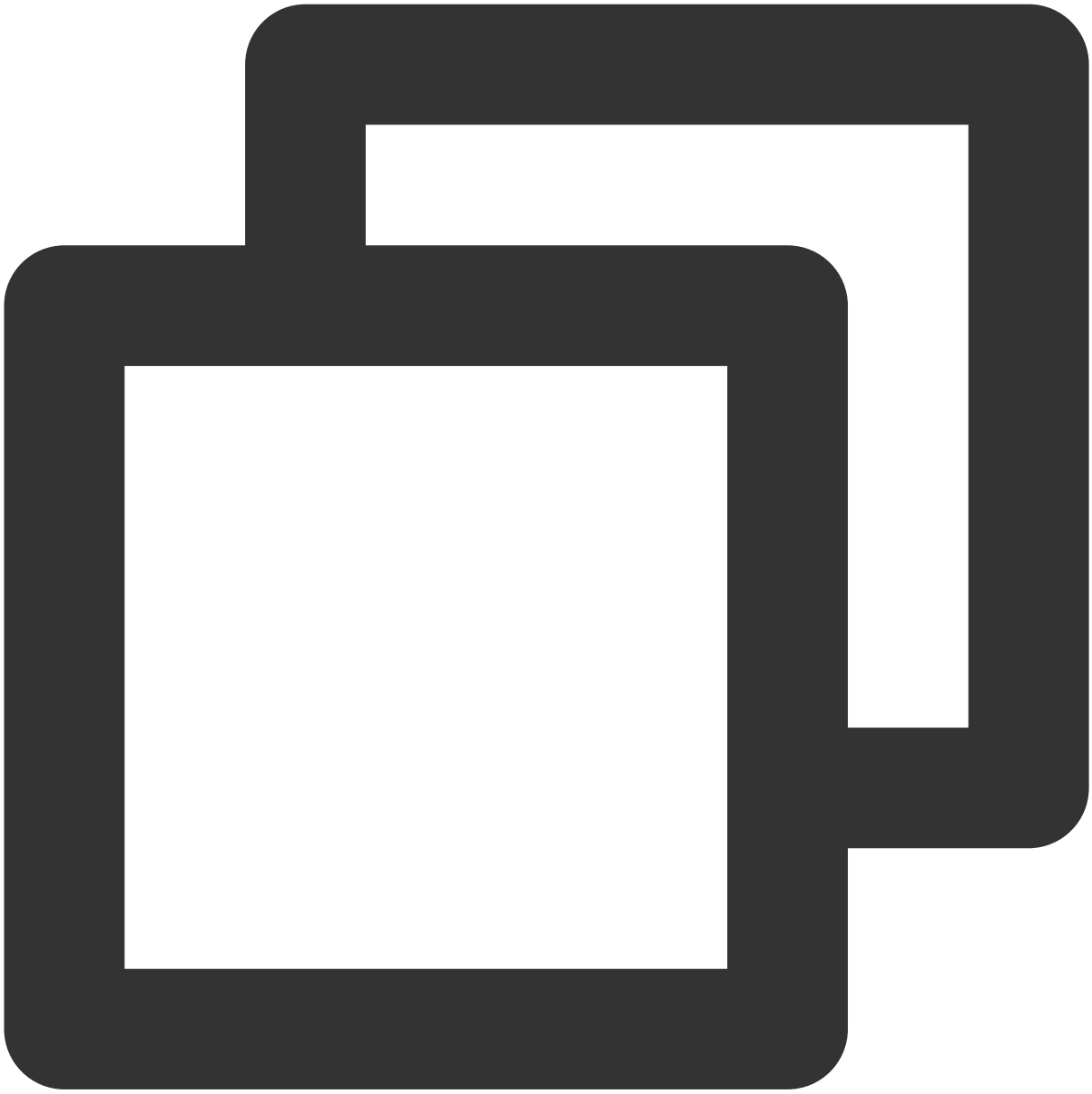
```
aegis.reportTime({
  name: 'onload', // 自定义测速 name
  duration: 1000, // 自定义测速耗时(0 - 60000)
  ext1: 'test1',
  ext2: 'test2',
  ext3: 'test3',
});
```

说明：

`onload` 可以修改为其他的命名。

time 和 timeEnd

该方法同样可用来上报自定义测速，适用于两个时间点之间时长的计算并上报，例如：



```
aegis.time('complexOperation');
/**
 * .
 * .
 * 做了很久的复杂操作之后。
 * .
 * .
 */
```

```
aegis.timeEnd('complexOperation'); /** 此时日志已经报上去了**/
```

说明：

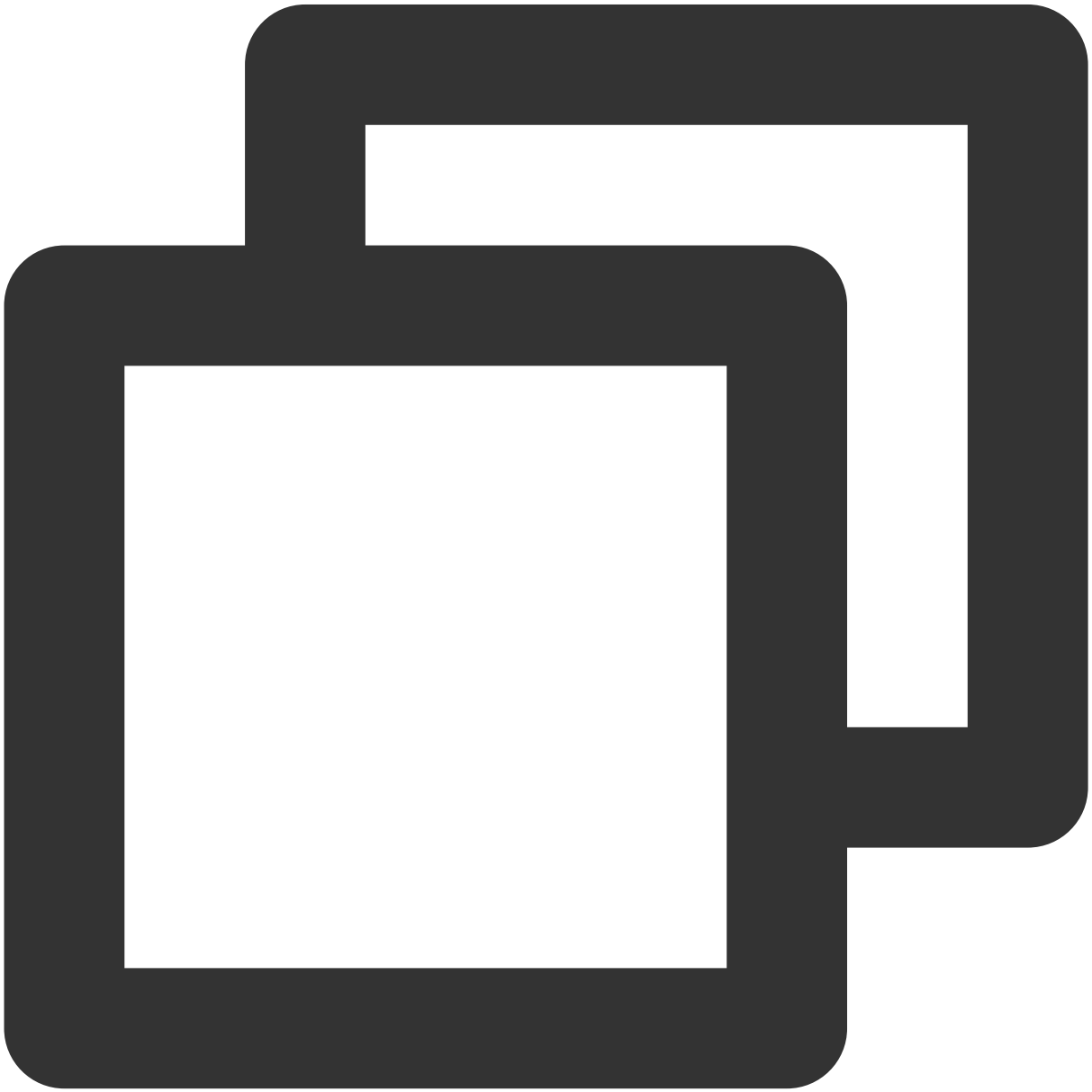
`complexOperation` 可以修改为其他的命名。

自定义测速是用户上报任意值，服务端对其进行统计和计算。由于服务端不能做脏数据处理，建议用户在上报端进行统计值限制，防止脏数据对整体产生影响。

目前 Aegis 只支持 0 - 60000 的数值计算，如果大于该值，建议进行合理改造。

destroy

销毁实例进程，销毁后数据不再上报，并且 Aegis 不再收集用户数据。



```
aegis.destroy();
```

白名单

最近更新时间：2024-01-22 19:39:30

白名单功能是适用于开发者针对特定的用户上报更多的信息。为了过滤部分用户，并减少用户的接口请求次数，因此 RUM 提供了白名单功能，并设定了白名单的逻辑。
白名单逻辑如下：
1. 白名单用户会上报全部的 API 请求信息，包括接口请求和请求结果。
2. 白名单用户可以使用 info 接口上报数据。
3. info 和 infoAll：在开发者实际体验过程中，白名单用户可以添加更多的日志，并且使用 info 进行上报。infoAll 会对所有用户无差别进行上报，因此可能导致日志量上报巨大。
4. 通过接口 whitelist 来判断当前用户是否是白名单用户，白名单用户的返回结果会绑定在 Aegis 实例上 (aegis.isWhiteList) 用来给开发者使用。
5. 用了减少开发者使用负担，白名单用户是针对业务系统生效。您可以在**应用管理 > 白名单管理**内创建白名单，则业务系统下全部应用都对该白名单用户生效。

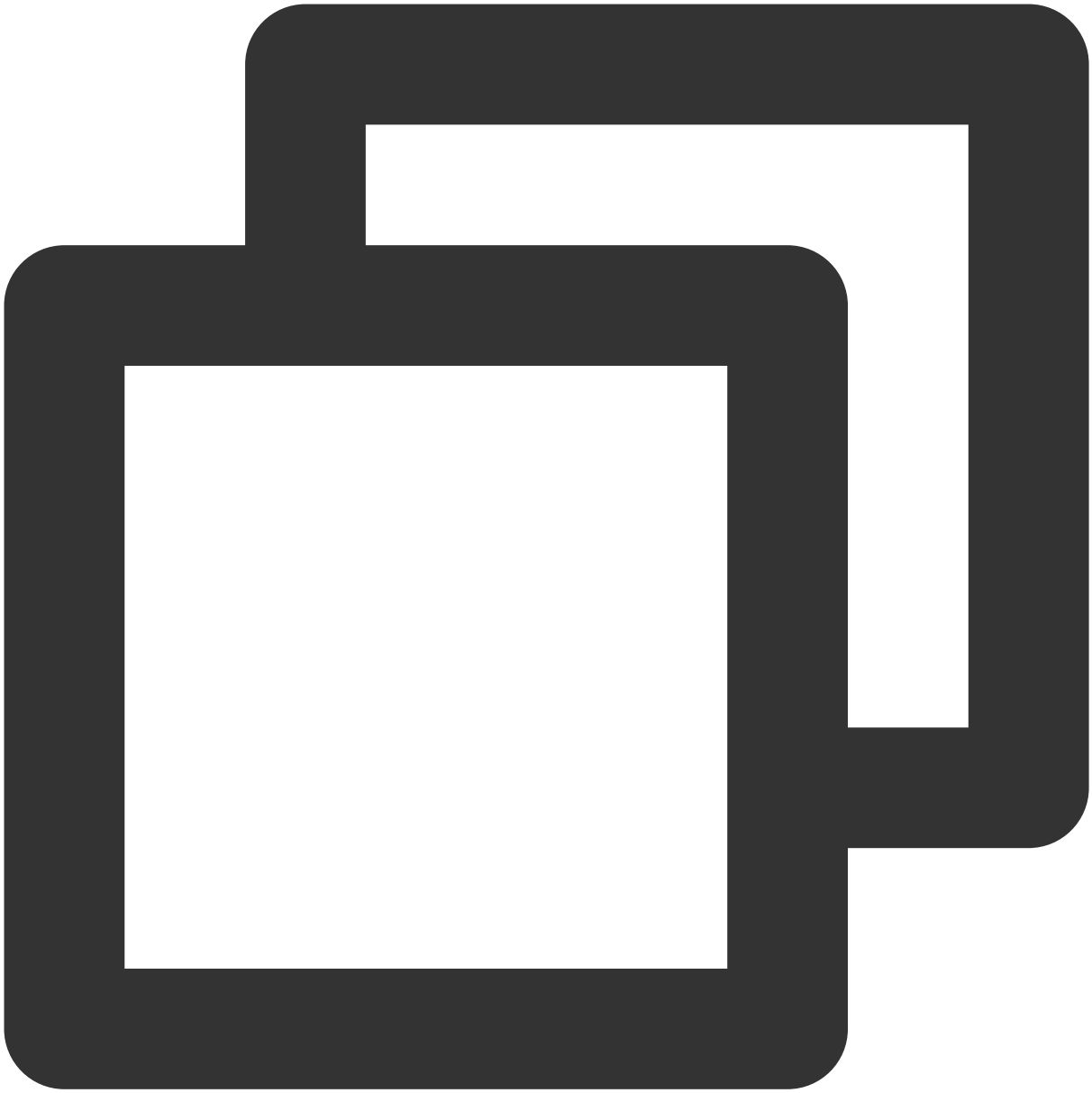
钩子函数

最近更新时间：2024-01-22 19:39:30

您可以使用钩子函数对某些资源的测速上报进行自定义配置，系统将会为您计算、统计函数执行时间等。您可以在自定义测速多维度分析函数执行耗时。

beforeReport

1. 该钩子将会在日志上报（对应上报接口为 `/collect?` ）前执行，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
  // 监听到下面抛出的错误
    console.log(log); // {level: "4", msg: "发生错误啦！！！！"}
    return log;
  }
});

throw new Error('发生错误啦！！！！');
```

说明：

上述 `log` 将会有以下几个字段：

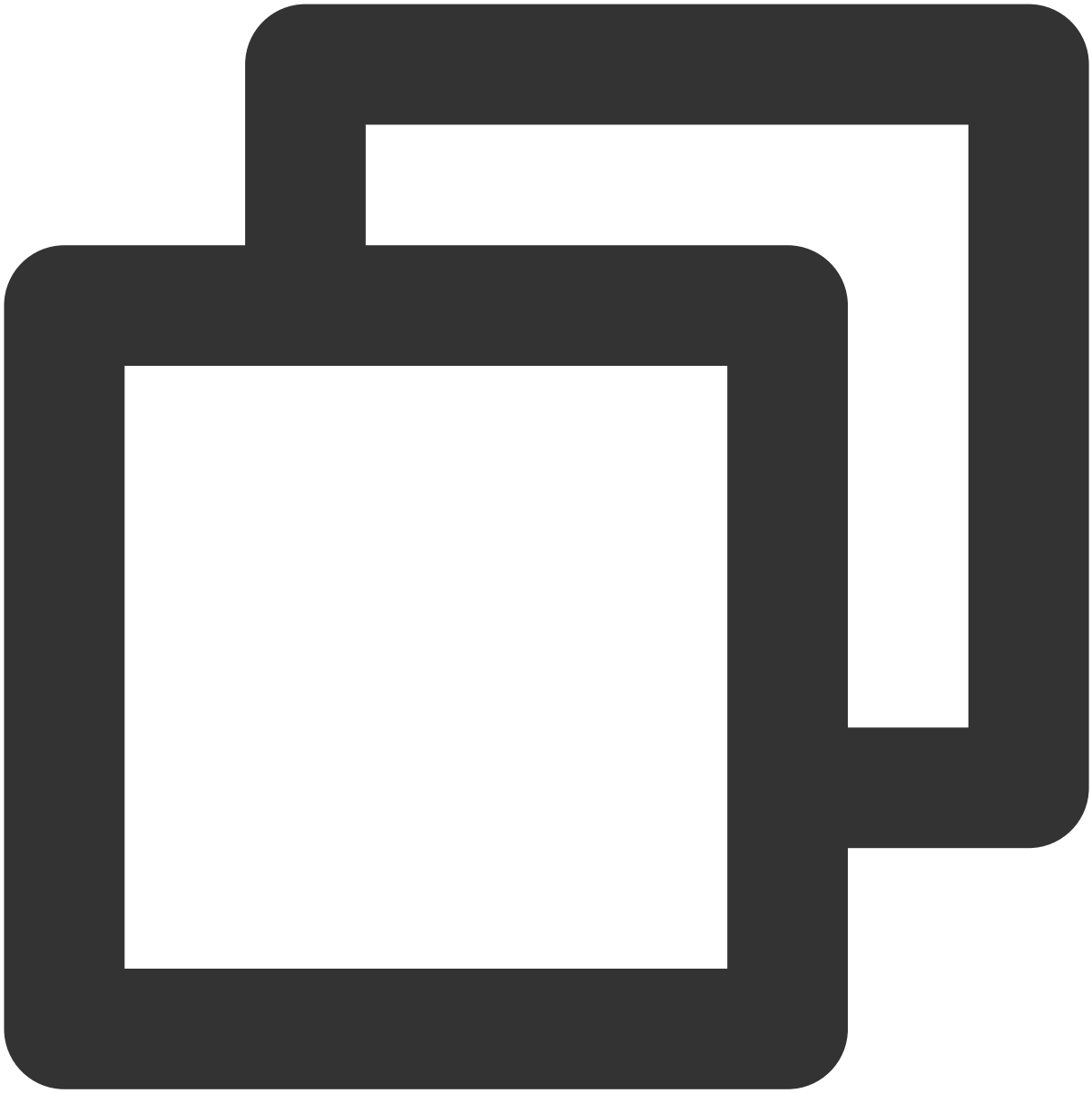
1. `level`：日志等级，例如：当 level 为 `'4'` 时代表错误日志；

2. msg`: 日志内容；

3. 全部日志等级如下：

```
{ level: '1', name: '接口请求日志（白名单日志）' }
{ level: '2', name: '一般日志' }
{ level: '4', name: 'JS 执行错误' }
{ level: '8', name: 'Promise 错误' }
{ level: '16', name: 'Ajax 请求异常' }
{ level: '32', name: 'JS 加载异常' }
{ level: '64', name: '图片加载异常' }
{ level: '128', name: 'css 加载异常' }
{ level: '256', name: 'console.error (未启用)' }
{ level: '512', name: '音视频资源异常' }
{ level: '1024', name: 'retcode 异常' }
{ level: '2048', name: 'aegis report' }
```

2. 当该钩子返回 `false` 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    if (log.level === '4' && log.msg && log.msg.indexOf('碍眼的错误') !== -1) {
      return false
    }
  }
});
throw new Error('碍眼的错误'); // 该错误将不会被上报
```

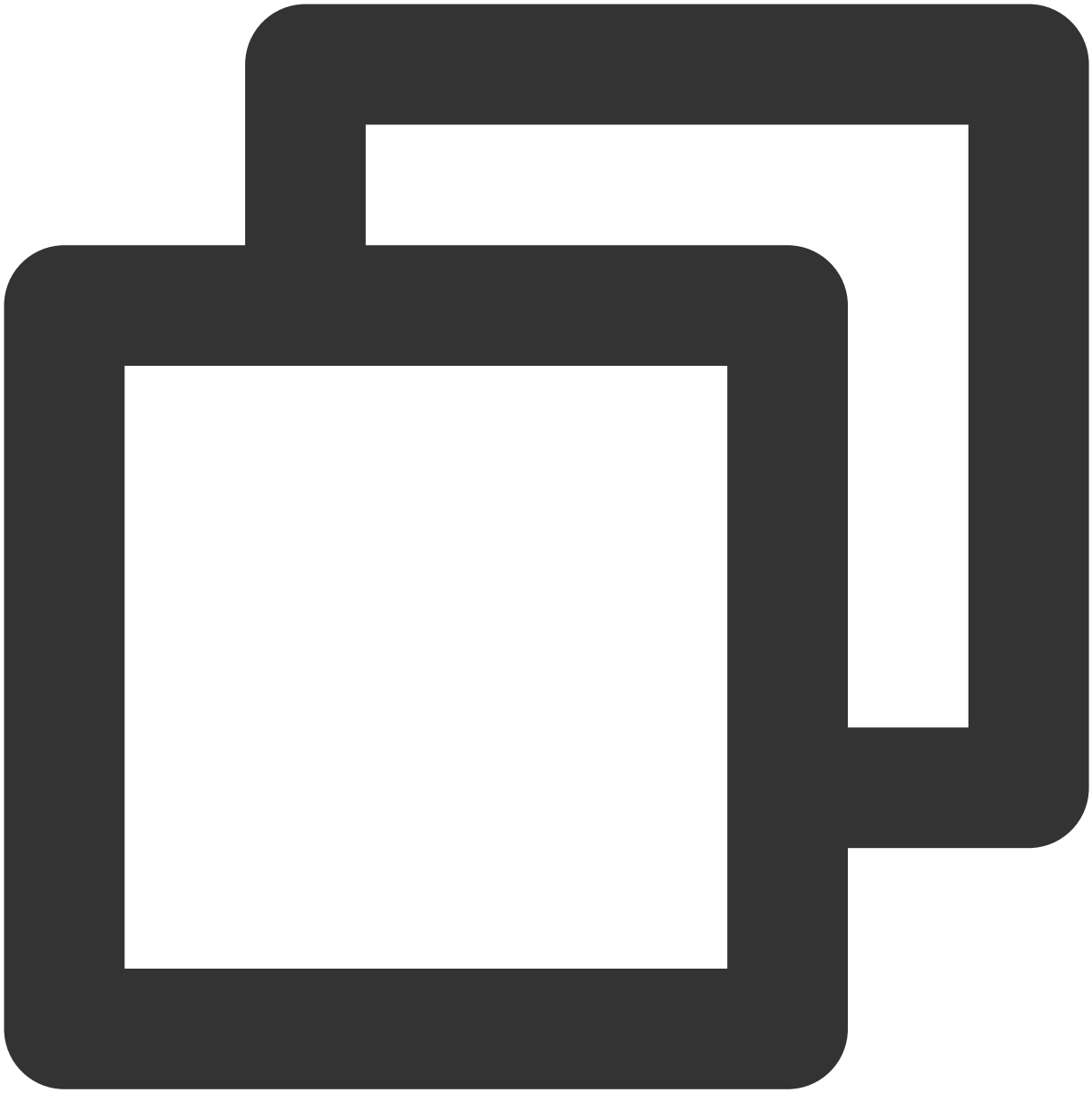
上面例子中，当上报的错误内容包含 `碍眼的错误` 几个关键字时，将不会上报至 RUM 后台中。

onReport

该钩子将在日志上报成功之后执行，用法类似 `beforeReport` 钩子，唯一不同点在于，该钩子接收到的所有参数都是已经上报完成的日志，而 `beforeReport` 钩子接收的参数是即将上报的日志。

beforeReportSpeed

1. 该钩子将会在测速数据上报前被执行，例如：



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    console.log(msg); // {url: "https://localhost:3001/example.e31bb0bc.js", method
    return msg
  }
});
```

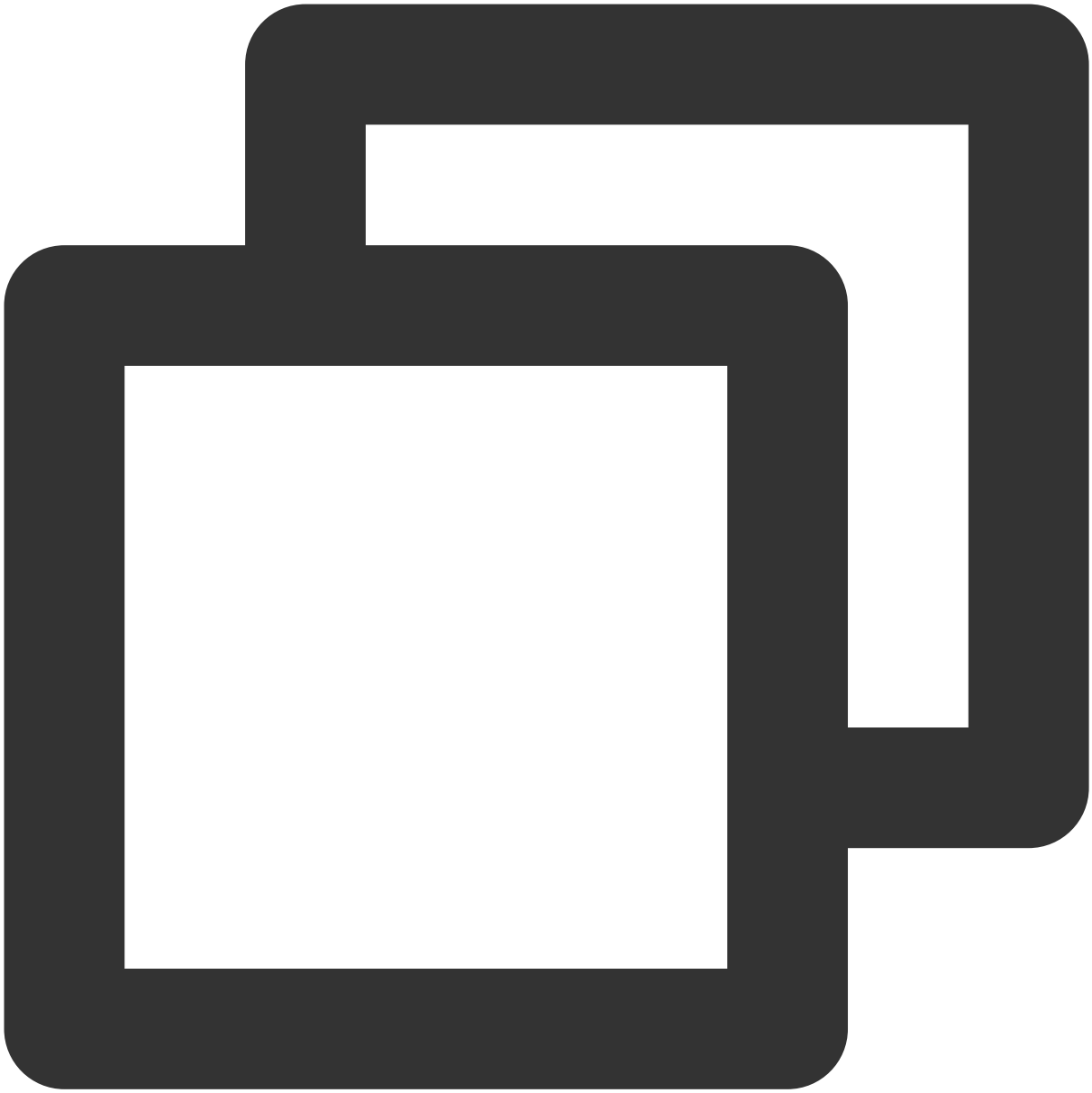
说明：

上述 `msg` 将会有以下几个字段：

1. `url`：该资源的请求地址。
2. `type`：该资源的类型，目前有 `fetch` 、 `static` 两种，当为 `fetch` 时，Aegis 将会把该资源当成 API 请求进行上报， `static` 时则视为静态资源。
3. `duration`：该资源请求耗时。
4. `method`：请求该资源时使用的 `http method`。
5. `status`：服务器返回状态码。

上面的例子中，每当 Aegis 收集到一个资源的加载详情时，将会以该资源的加载情况（上面的返回的 `msg` ）作为参数调用 `beforeReportSpeed` 钩子。

2. 如果您配置了该钩子，Aegis 最终的上报内容将以钩子的执行结果为准。例如：



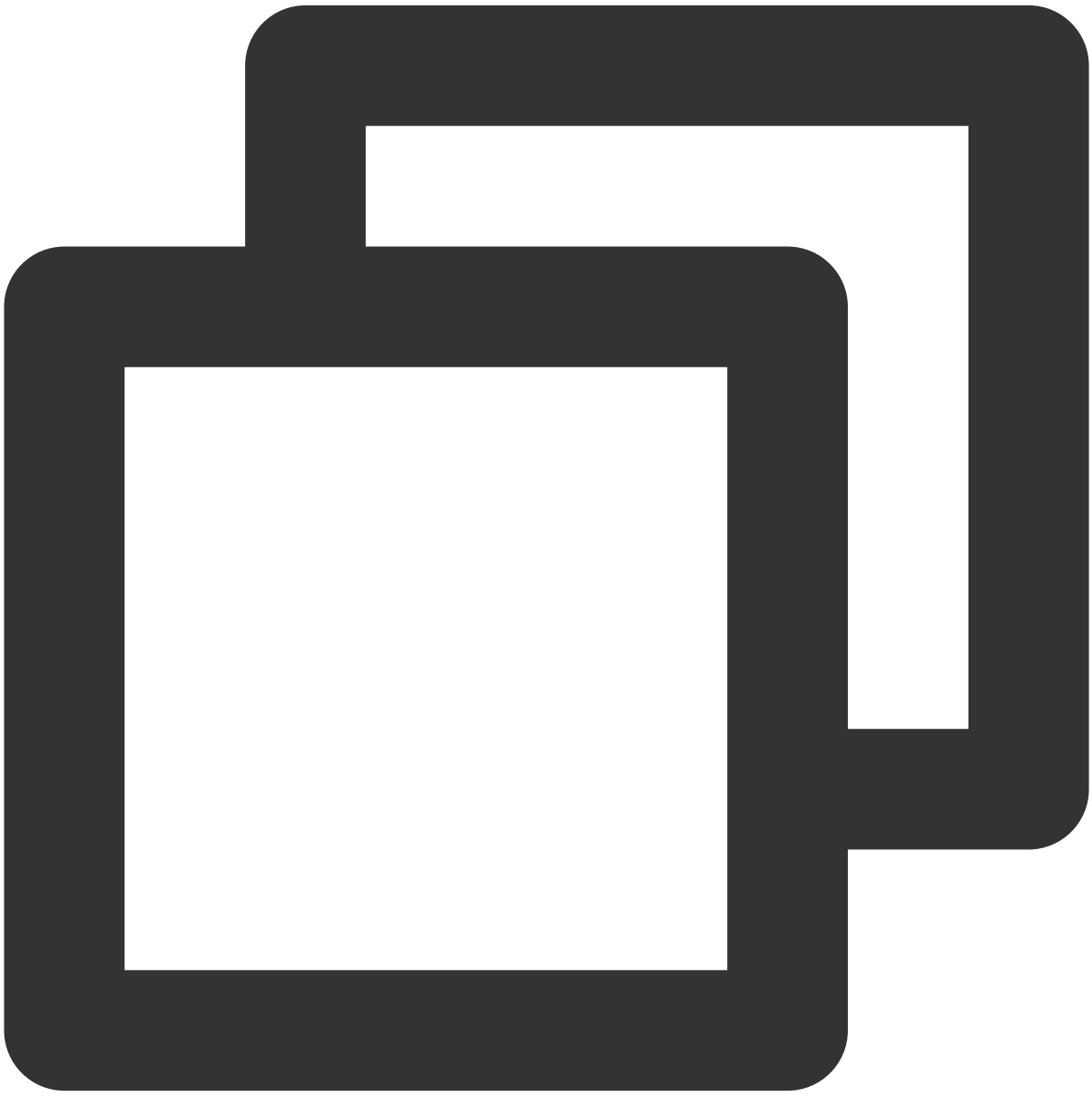
```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    msg.type = 'static';
  }
});
```

上述代码中，将所有的 `msg.type` 设置为 `static`，这意味着所有的资源都将被当成静态资源进行上报，API 请求也将被报至静态资源中。

3. 使用该钩子，您可以校准 Aegis 类型判断错误的请求。

示例

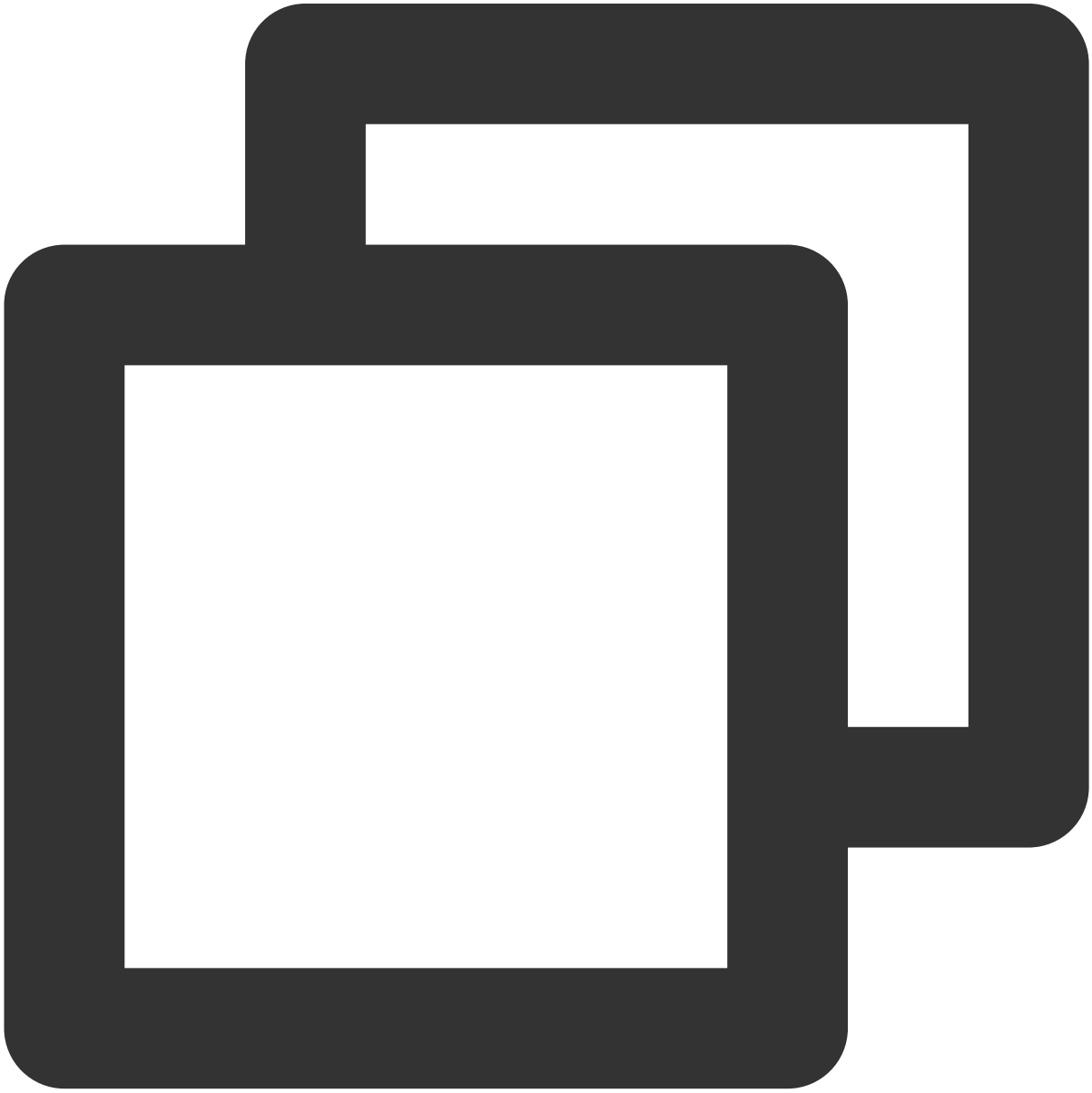
假如您有一条接口 `https://example.com/api`，该接口的响应头 `Content-Type` 为 `text/html`。正常情况下，RUM 会将该资源当成静态资源进行上报。但在您的业务中，该接口就必须视为 API 请求进行上报，您可以给 Aegis 配置如下钩子进行校正：



```
const aegis = new Aegis({
```

```
    id: 'pGUVFTCZyewxxxxx',
    reportApiSpeed: true,
    reportAssetSpeed: true,
    beforeReportSpeed(msg) {
      if (msg.url === 'https://example.com/api') {
        msg.type = 'fetch';
      }
    }
  });
```

1. 您还可以屏蔽某些资源的测速上报，例如：



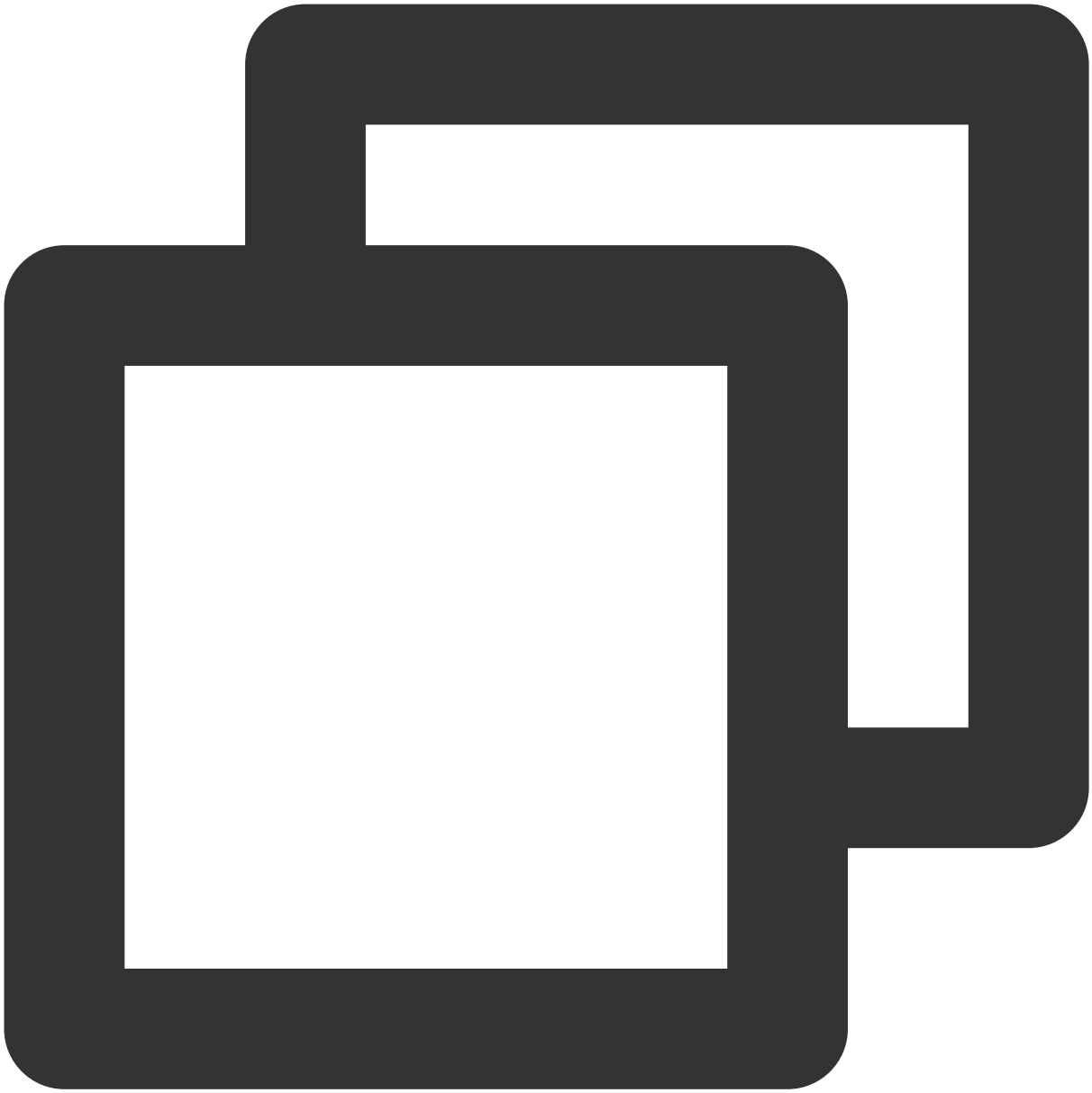
```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    // 地址中包含‘https://example.com/api’的都不上报
    if (msg.url.indexOf('https://example.com/api') !== -1) {
      // 返回 ‘false’ 将阻止本条测速日志的上报
      return false
    }
  }
});
```

beforeRequest

该钩子将会在日志上报前执行，例如：

注意：

SDK 版本应大于等于 1.24.44。



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeRequest: function(msg) {
    if (msg.logs && msg.logs.level === '4' && msg.logs.msg && msg.logs.msg.indexOf(
      return false
    }
    return msg;
  }
});
```

其中，msg 将会有以下几个字段：

1. logType：日志类型，有以下值：
custom：自定义测速
event：自定义事件
log：日志
performance：页面测速
pv：页面PV
speed：接口和静态资源测速
vitals：web vitals
2. logs：上报的日志内容：
当 logType 为 'custom' 时，logs 数据类型为 `{name: "白屏时间", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}` 。
当 logType 为 'event' 时，logs 数据类型为 `{name: "ios", ext1: "", ext2: "", ext3: ""}` 。
当 logType 为 'performance' 时，logs 数据类型为 `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 260, ssl: 4, tcp: 4, ttfb: 5}` 。
当 logType 为 'speed' 时，logs 数据类型为 `{connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "tatic", url: "https://xxxxxx", urlQuery: "max_age=1296000"}` 。
当 logType 为 'vitals' 时，logs 数据类型为 `{delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "CP", value: 1100}` 。
当 logType 为 'log' 时，logs 数据类型为 `{msg: "日志详情", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}` 。
说明：
其中 level 枚举值如下：

```
{ level: '1', name: '接口请求日志（白名单日志）' }
{ level: '2', name: '一般日志' }
{ level: '4', name: 'JS 执行错误' }
{ level: '8', name: 'Promise 错误' }
{ level: '16', name: 'Ajax 请求异常' }
{ level: '32', name: 'JS 加载异常' }
{ level: '64', name: '图片加载异常' }
{ level: '128', name: 'css 加载异常' }
{ level: '256', name: 'console.error（未启用）' }
{ level: '512', name: '音视频资源异常' }
{ level: '1024', name: 'retcode 异常' }
{ level: '2048', name: 'aegis report' }
```

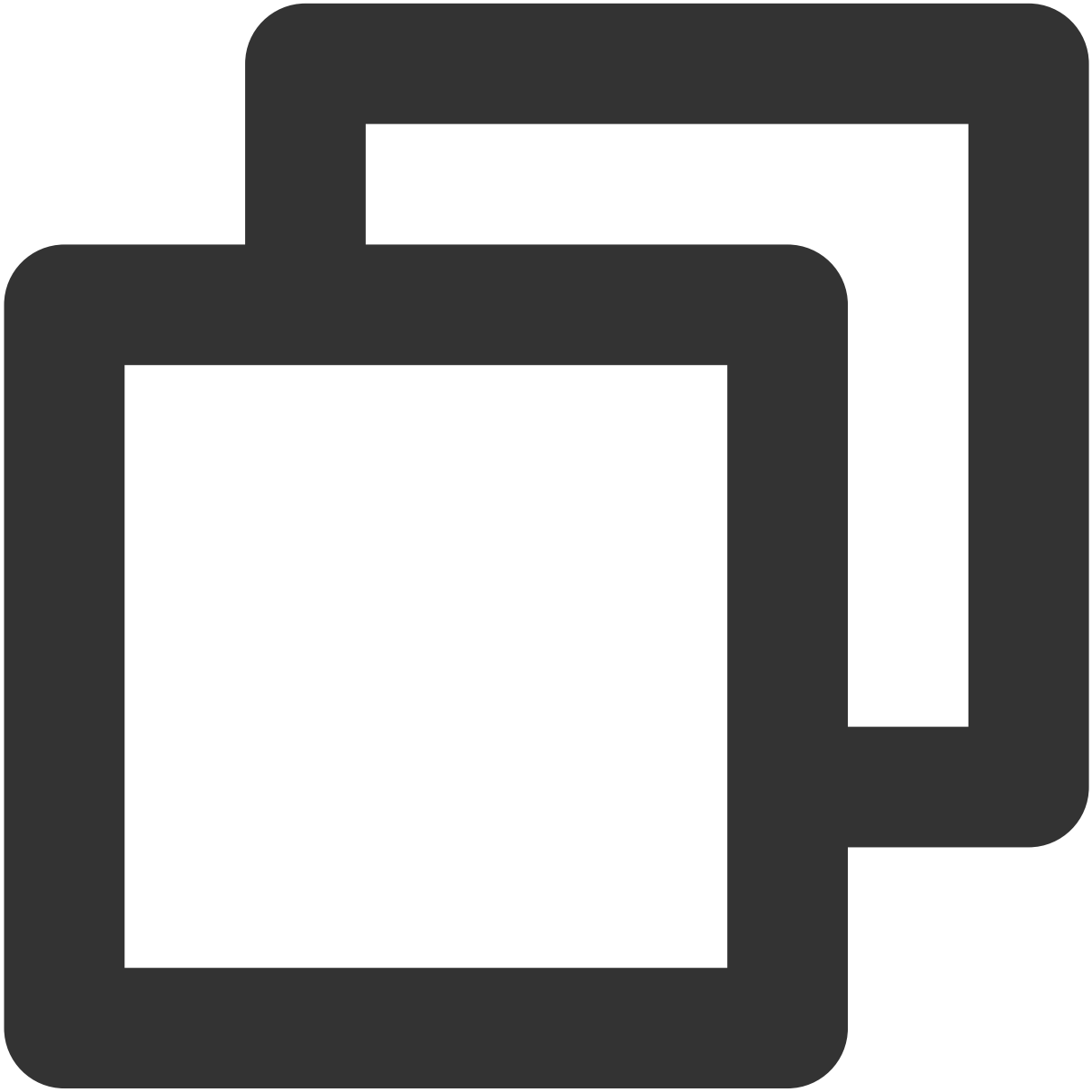
该钩子返回 false 时，本条日志将不会进行上报，该功能可用来过滤某些不需要上报的错误，可以用来过滤不希望上报的日志。

afterRequest

该勾子将会在测速数据上报后被执行，例如：

注意：

SDK 版本应大于等于 1.24.44。



```
const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx",
  afterRequest: function(msg) {
    // {isErr: false, result: Array(1), logType: "log", logs: Array(4)}
```

```
    console.log(msg);
  }
});
```

其中，msg 将会有以下几个字段：

1. isErr：请求上报接口是否错误。
2. result：上报接口的返回结果。
3. logs：上报的日志内容。
4. logType：日志类型，同 beforeRequest 中的 logType。

错误监控

最近更新时间：2024-01-22 19:39:30

前端性能监控的 Aegis 的实例会自动进行 JS 执行错误、Promise执行错误、Ajax（Fetch）请求异常等监控。本文将为您介绍各错误监控逻辑及处理方式。

注意：

Aegis 实例会对这些异常进行监控，当您只是引入了 SDK 而没有将其实例化时，Aegis 将不会上报数据。

JS 执行错误

Aegis 通过监听 `wx` 对象上的 `onerror` 事件来获取项目中的报错，并且通过解析错误和分析堆栈，将错误信息自动上报到后台服务中。该上报的上报等级为 error，所以当自动上报的错误达到阈值时，Aegis 将会自动告警，帮助您尽早发现异常。由于上报等级为 error，自动上报也将影响项目的评分。

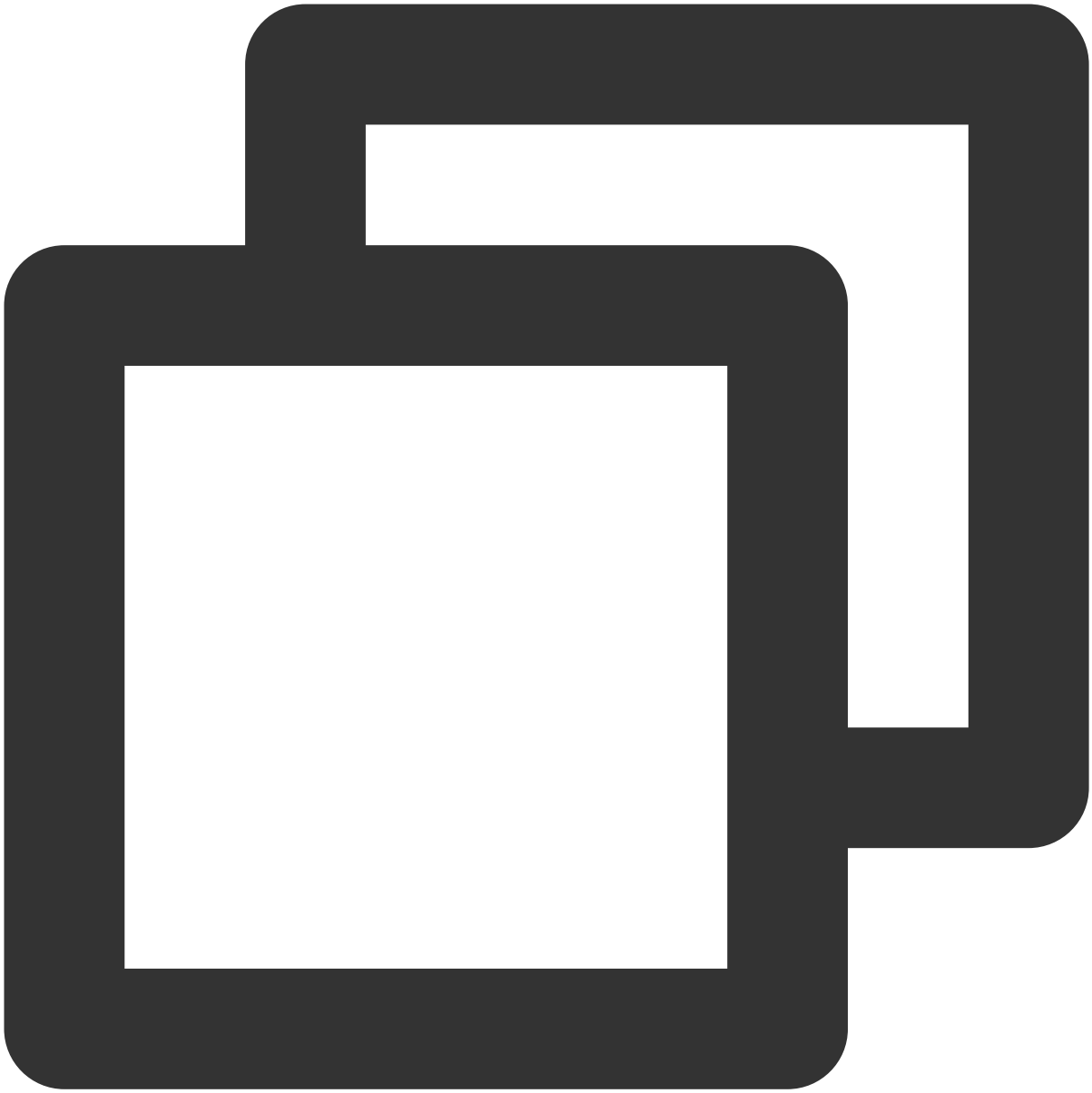
request 请求异常

Aegis 将会改写 `wx.request`（`qq.request`），之后监听每次接口请求，当 `statusCode` 不存在或者大于等于 400 时认为该请求是一个失败的请求，抑或是接口超时，abort 和其他类型的失败。

如果您的项目中使用了一些库也会重写 `wx.request`（`qq.request`），如：`miniprogram-api-promise` 或者自己有封装，请一定确保在引入这个库之前完成对 Aegis 的初始化，否则将无法监听到接口失败的情况。

注意：

Aegis SDK 在错误发生的时候，不会主动收集接口请求参数和返回信息，如果需要对进口信息进行上报，可以使用 API 参数里面的 apiDetail 进行开启。



```
new Aegis({
  api: {
    apiDetail: true,
  },
});
```

retcode异常

Aegis 在改写 `wx.request`（`qq.request`），在获得 API 返回的内容，并尝试在内容中获取到本次请求的 `retcode`，当 `retcode` 不符合预期的时候，会认为本次请求出现了异常，并进行上报。

说明：

如何获取 `retcode` 以及哪些 `retcode` 是正常的可以在 配置文档 中查看。

性能监控

最近更新时间：2024-01-22 19:39:31

您可以通过本文了解页面测速、接口测试等信息。

页面测速

Aegis 小程序 SDK 会自动收集页面性能数据并上报。包括：

1. 程序启动时间。
2. 代码注入时间。
3. 页面首次渲染时间。
4. 路由切换时间。

注意：

获取页面性能数据依赖小程序的 Performance API，请保证基础库版本大于 `2.11.0` 。

QQ 小程序目前暂时不支持 Performance API，所以不会上报页面性能数据。

接口测速

说明：

打开方式：初始化时传入配置 `reportApiSpeed: true`

Aegis 通过劫持 `wx.request || qq.request` 进行接口测速。

配置文档

最近更新时间：2024-01-22 19:39:30

配置说明

配置文档各配置项说明如下：

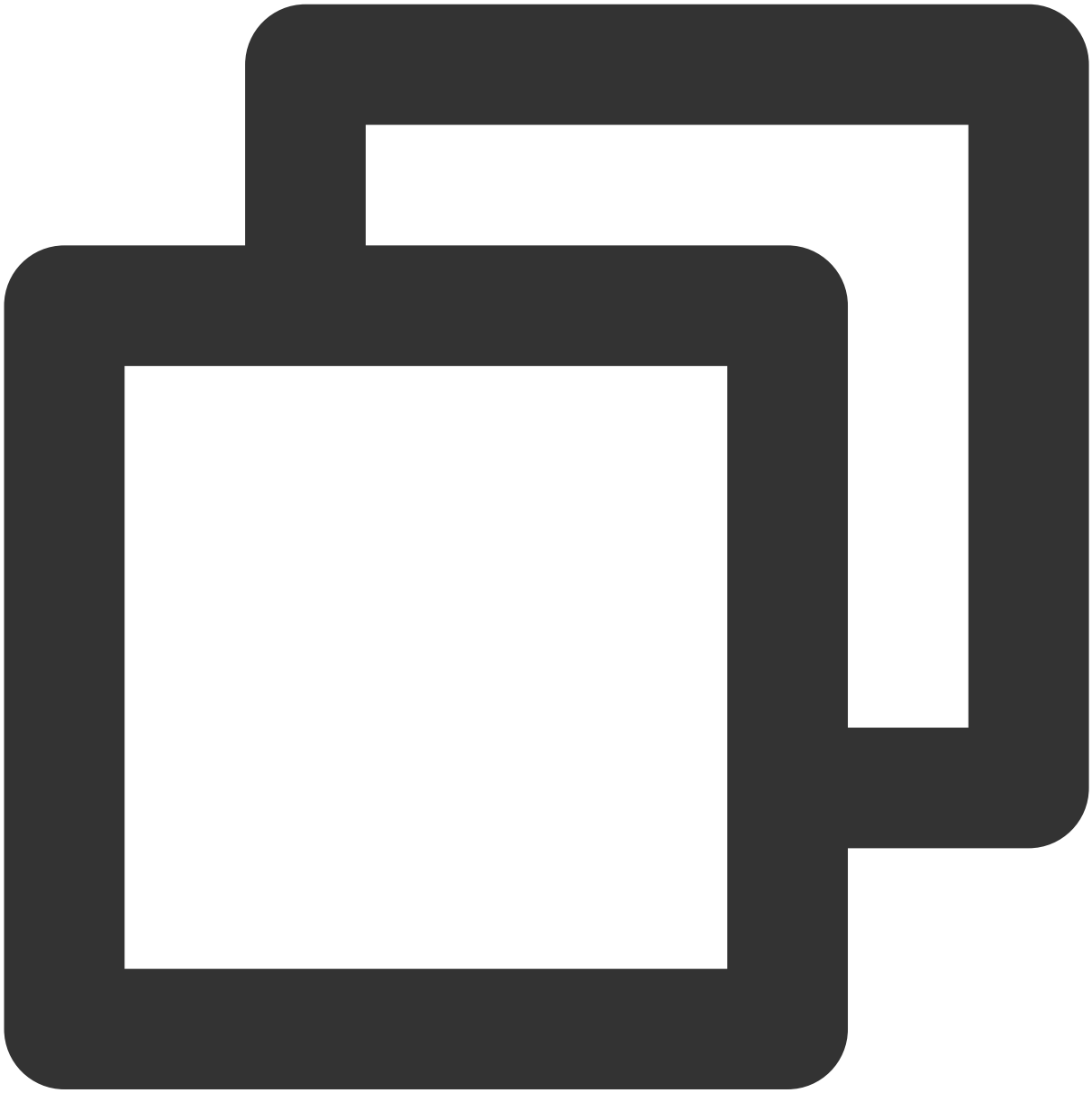
配置	描述
id	必须，number，默认 `无` 。 开发者平台分配的项目 key。
uin	建议，string，默认取 cookie 中的 UIN 字段。 当前用户的唯一标识符，白名单上报时将根据该字段判定用户是否在白名单中，字段仅支持 `字母数字@=._-` ，正则表达式: `/^[@=.0-9a-zA-Z_-]{1,60}$/` 。
reportApiSpeed	可选，boolean，默认 false。 是否开启接口测速。
version	可选，string，默认 sdk 版本号。 当前上报版本，当页面使用了 pwa 或者存在离线包时，可用来判断当前的上报是来自哪一个版本的代码，仅支持 `字母数字.,:_-` ，长度在 60 位以内 `/^[0-9a-zA-Z.,:_-]{1,60}$/` 。
delay	可选，number，默认 1000 ms。 上报节流时间，在该时间段内的上报将会合并到一个上报请求中。
repeat	可选，number，默认 5。 重复上报次数，对于同一个错误超过多少次不上报。
env	可选 enum，默认 Aegis.environment.production。当前项目运行所处的环境。
spa	可选，boolean，默认 false。 是否在小程序页面跳转时进行 PV 上报。
offlineLog	可选，boolean，默认 false。 是否使用离线日志。
offlineLogExp	可选，number，默认 3。 离线日志过期天数。
url	可选，string，默认 `//aegis.qq.com/collect` 。 日志上报地址。 设置为空字符串可以不进行日志上报。

pvUrl	可选，string，默认 `//aegis.qq.com/collect/pv` 。 pv 上报地址。 设置为空字符串可以不进行 pv 上报。
whiteListUrl	可选，string，默认 `//aegis.qq.com/collect/whitelist` 。 白名单确认接口， 设置为空字符串可以关闭白名单接口请求
offlineUrl	可选，string，默认 `//aegis.qq.com/collect/offline` 。 离线日志上报地址。 设置为空字符串可以不进行离线日志上报。
eventUrl	可选，string，默认 `//aegis.qq.com/collect/events` 。 自定义事件上报地址。 设置为空字符串可以不进行自定义事件上报。
speedUrl	可选，string，默认 `//aegis.qq.com/speed` 。 测速日志上报地址。 设置为空字符串可以不进行测速数据上报。
customTimeUrl	可选，string，默认 `//aegis.qq.com/speed/custom` 。 自定义测速上报地址。 设置为空字符串可以不进行自定义测速上报。
performanceUrl	可选，string，默认 `//aegis.qq.com/speed/performance` 。 页面性能日志上报地址。 设置为空字符串可以不进行页面性能上报。
setDataReportConfig	可选，object，默认为{}。相关的配置: disabled: 可选，Boolean，默认false。是否禁用setData数据上报； timeThreshold: 可选，Number，单位为ms，默认值为30。上报的耗时阈值，表示仅上报更新耗时超过该阈值的数据； withDataPaths: 可选，Boolean，默认为true。是否上报本次更新的字段信息；
api	可选，object，默认为{}。相关的配置： apiDetail：可选，boolean，默认 ：false。api 失败时，是否上报 api 的请求参数和返回值； retCodeHandler：Function， 返回码上报钩子函数。会传入接口返回数据，返回值为{isErr: boolean, code: string}。详情请参见示例 api.retCodeHandler。 reportRequest：boolean，默认：false。开启后，aegis.info 会变成全量上报，不需要白名单配置，并且会上报所有接口的信息（上报接口需开启 reportApiSpeed）
ext1	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext2	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。
ext3	可选，string，自定义上报的额外维度，上报的时候可以被覆盖。

示例

api.retCodeHandler

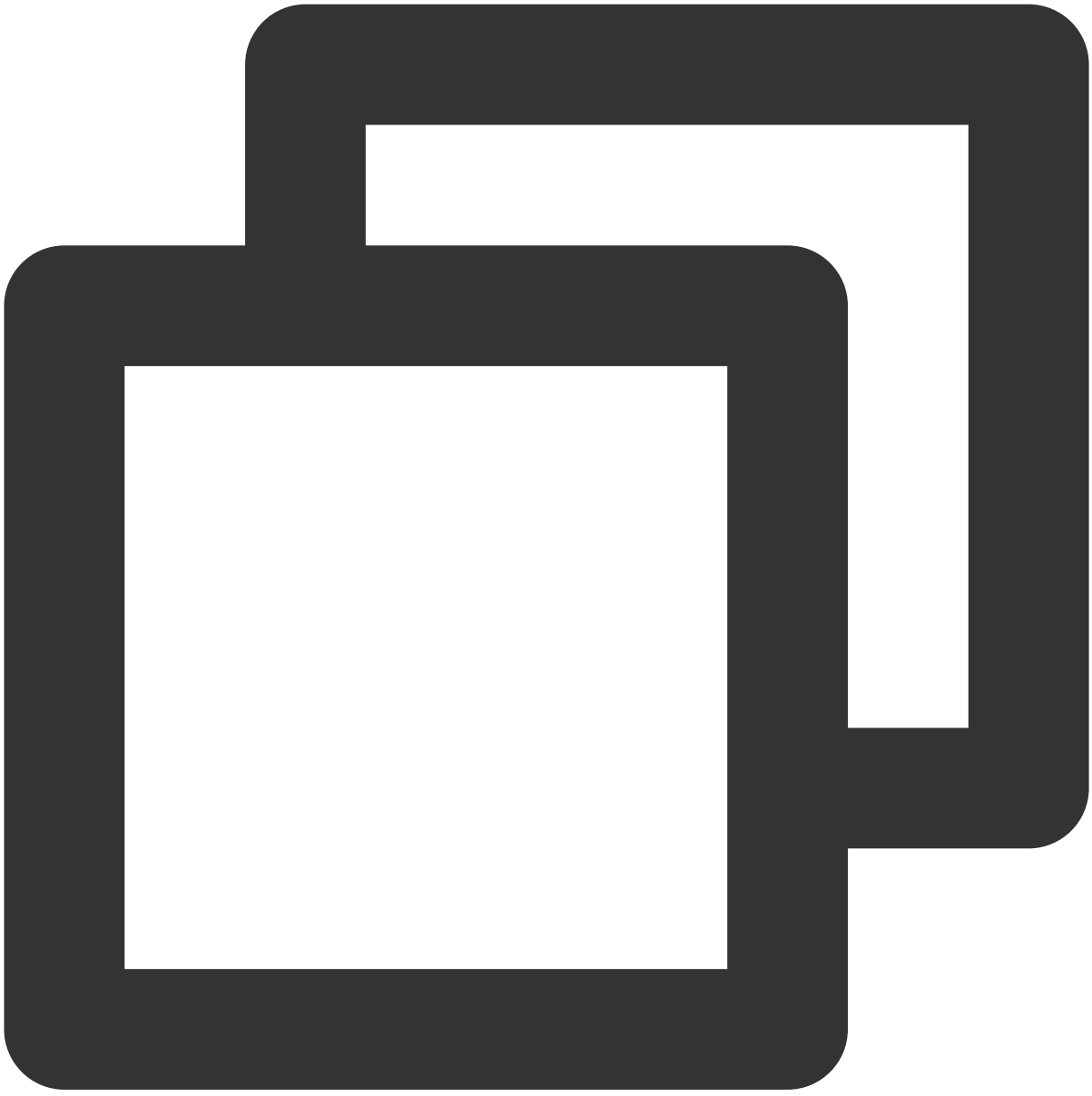
假如后台返回数据为:



```
{
  body: {
    code: 200,
    retCode: 0,
    data: {
```

```
        // xxx
    }
}
```

业务需要：code 不为200，或者 retCode 不为0，此次请求就是错误的。此时只需进行以下配置：



```
new Aegis({
// xxx
  reportApiSpeed: true, // 需要开两个，不然不会有返回码上报
  reportAssetSpeed: true,
  api: {
    retCodeHandler(data) {
```

```
// 注意这里拿到的data是string类型，如果需要对象需要手动parse下
      try {
        data = JSON.parse(data)
      } catch (e) {
      }
      return {
        isErr: data.body.code !== 200 || data.body.retCode !== 0,
        code: data.body.code
      }
    }
  }
})
```

IOS 应用场景
集成和初始化

最近更新时间：2024-04-25 11:42:15

本文指导您如何操作 SDK 集成与初始化。

操作步骤

步骤一：SDK 集成

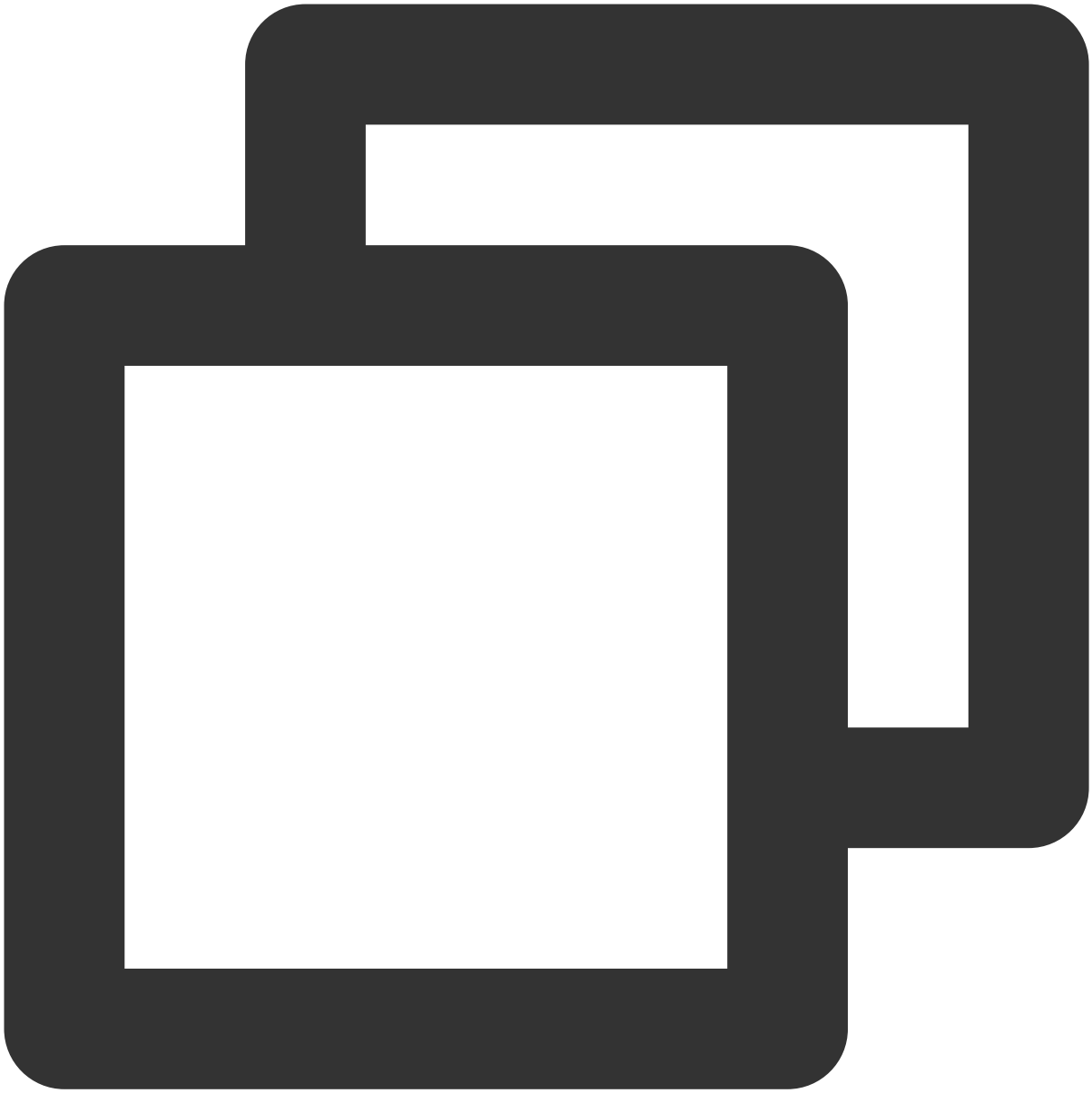
选择下列任意一种方式集成 SDK。

方法一：CocoaPods 集成

说明

暂时只支持本地的 cocoaPods 集成。

1. 安装 CocoaPods。
2. 通过 pod repo update 更新 qapm 的 Cocoapods 版本。
3. 在 Podfile 对应的 target 中，添加如下命令，并执行 pod install。



```
pod 'QAPM', :source => 'https://github.com/TencentCloud/QAPM-iOS-CocoaPods.git'
```

4. 在项目中使用 Cocoapods 生成的 .xcworkspace 运行工程,并将工程的 bitcode 参数设置为 NO。

注意：

在拉取过程中如果出现以下报错 ：

`[!] Unable to add asource with url https://github.com/TencentCloud/QAPM-iOS-CocoaPods.git named tencentcloud-qapm-ios-cocoapods. You cantry adding it manually in /Users/wxy/.cocoapods/repos or via pod repo add.` 可在终端执行以下指令：

`pod repo add tencentcloud-qapm-ios-cocoapods https://github.com/TencentCloud/QAPM-iOS-CocoaPods.git`

然后再执行：

`pod install`

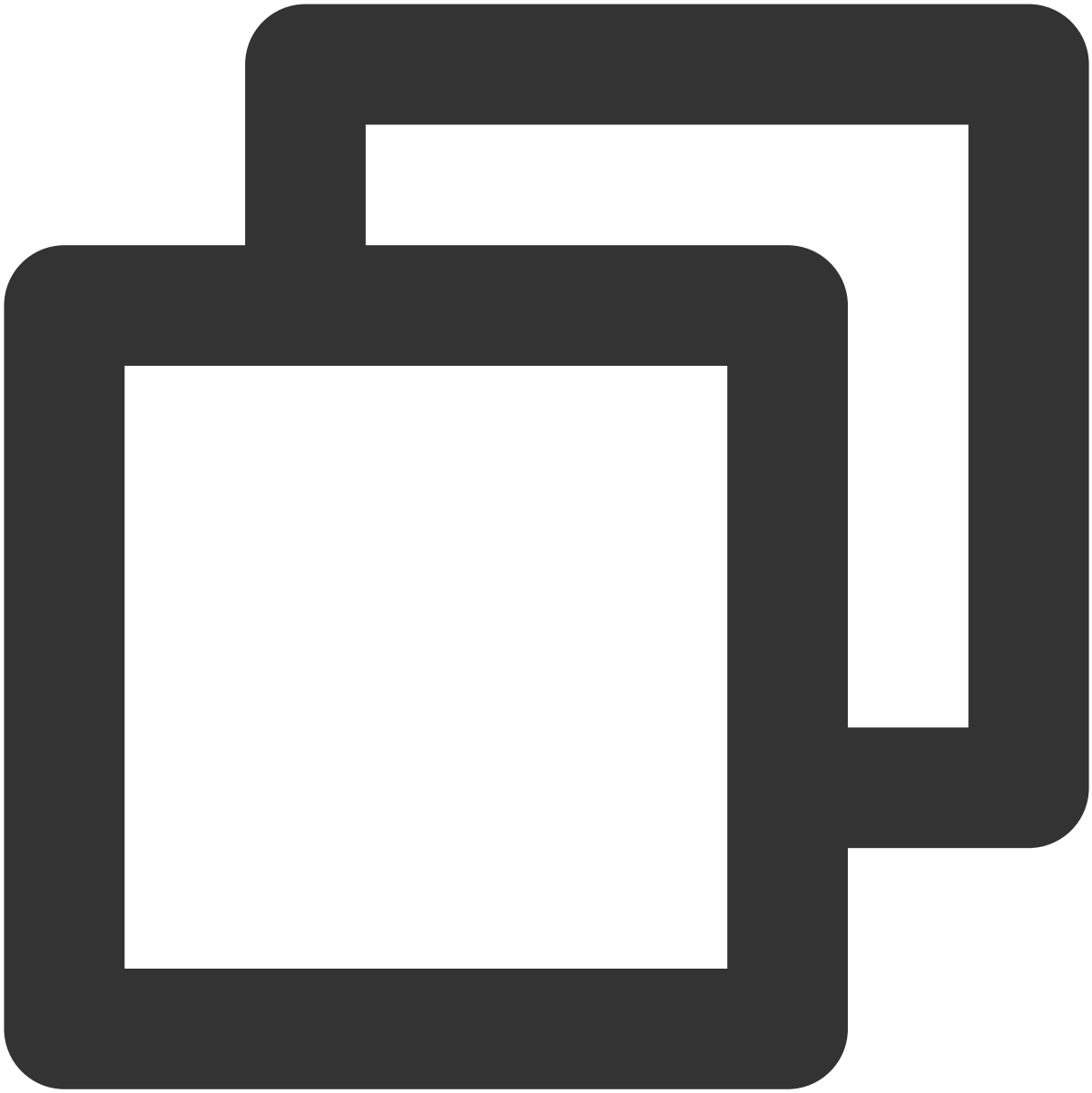
方法二：手动集成

1. 把 Demo 工程 QAPM_iOS_SDK_Demo下的 QAPM.framework 复制到业务工程。
2. 拖拽 QAPM.framework 文件到 Xcode 工程内（请勾选 Copy items if needed 选项)。
3. 在 TARGETS -> Build Phases-Link Binary Libraries加依赖库 libc++.tbd、 libz.tbd、CoreLocation。
4. 将 framework 里面的 QAPM.bundle、js_sdk.js 拖入到业务工程。
5. 在工程的 Other Linker Flags 中添加 -ObjC 参数。
6. 将工程的 bitcode 参数设置为 NO。

步骤二：初始化 SDK 配置及 Web 端环境配置

1. 登录 腾讯云可观测平台 控制台，选择左侧导航栏的**终端性能监控**，点击**应用管理 > 应用设置**后，获取 Appkey（上报 ID）。

2. 在 `AppDelegate.m` 文件引入头文件，并加入以下配置：



```
#import <QAPM/QAPM.h>
#import <QAPM/QAPMLaunchProfile.h>
如果是Swift工程，请在对应bridging-header.h中导入
初始化QAPM 在工程AppDelegate.m的application:didFinishLaunchingWithOptions:方法中初始化：

void loggerFunc(QAPMLoggerLevel level, const char* log) {

#ifdef RELEASE
   if (level <= QAPMLogLevel_Event) { ///外发版本log
       NSLog(@"%@", [NSString stringWithUTF8String:log]);
   }
```

```
#endif

#ifdef GRAY
    if (level <= QAPMLogLevel_Info) { ///灰度和外发版本log
        NSLog(@"%@", [NSString stringWithUTF8String:log]);
    }
#endif

#ifdef DEBUG
    if (level <= QAPMLogLevel_Debug) { ///内部版本、灰度和外发版本log
        NSLog(@"%@", [NSString stringWithUTF8String:log]);
    }
#endif
}

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    //请在同意相应的隐私合规政策后进行QAPM的初始化
    [self setupQapm];
}

- (void)setupQapm {

    //启动耗时自定义打点开始,业务自行打点
    [QAPMLaunchProfile setBeginTimestampForScene:@"finish"];

    [QAPM registerLogCallback:loggerFunc];
#ifdef DEBUG
    //设置开启QAPM所有监控功能
    [[QAPMModelStableConfig getInstance] setupModelAll:1];
#else
    [[QAPMModelStableConfig getInstance] setupModelStable:0.5];
#endif

    //用于查看当前SDK版本号信息
    NSLog(@"qapm sdk version : %@", [QAPM sdkVersion]);

    //上报地址，固定值。https://app.rumt-zh.com
    [QAPMConfig getInstance].host = @"https://app.rumt-zh.com";

    // 设置用户标记，默认值为10000,userID会作为计算各功能的用户指标率，请进行传值
    [QAPMConfig getInstance].userId = @"请正确填写用户唯一标识";

    // 设置设备唯一标识，默认值为10000,deviceID会作为计算各功能的设备指标率，请进行传值
    [QAPMConfig getInstance].deviceID = @"请正确填写设备的唯一标识";
    // 设置App版本号
    [QAPMConfig getInstance].customerAppVersion = @"请正确填写业务APP版本";
    [QAPM startWithAppKey:@"请正确填写申请到的appkey"];
```

```
    //启动耗时自定义打点结束，业务自行打点
    [QAPMLaunchProfile setEndTimestampForScene:@"finish"];
}
```

说明

1. 白名单分为用户白名单和设备白名单。初始化配置代码里的 userId 或者 deviceID 需要通过**终端性能监控 > 应用管理**页面添加到白名单里，移动端的webview、卡慢监控、掉帧、网络监控数据才会开启上报。
2. 更多高级功能配置请参见 Demo 工程，以及 shell 脚本及相关文档文件夹中的文档。

注意：

1. 从版本5.2.8开始，出于优化考虑，SDK 如果检测不到 embedded.mobileprovision 文件，则不会输出 SDK 相关日志。在进行接入测试时，可以手动创建空白的 embedded.mobileprovision 文件跳过 SDK 检查(需要通过 XCode 引入或者创建该文件，SDK 通过 pathForResource 获取该文件)。
2. 卡慢和掉帧监控在模拟器上不会开启，建议使用真机测试。

安卓应用场景
集成和初始化

最近更新时间：2024-04-25 11:42:15

操作场景

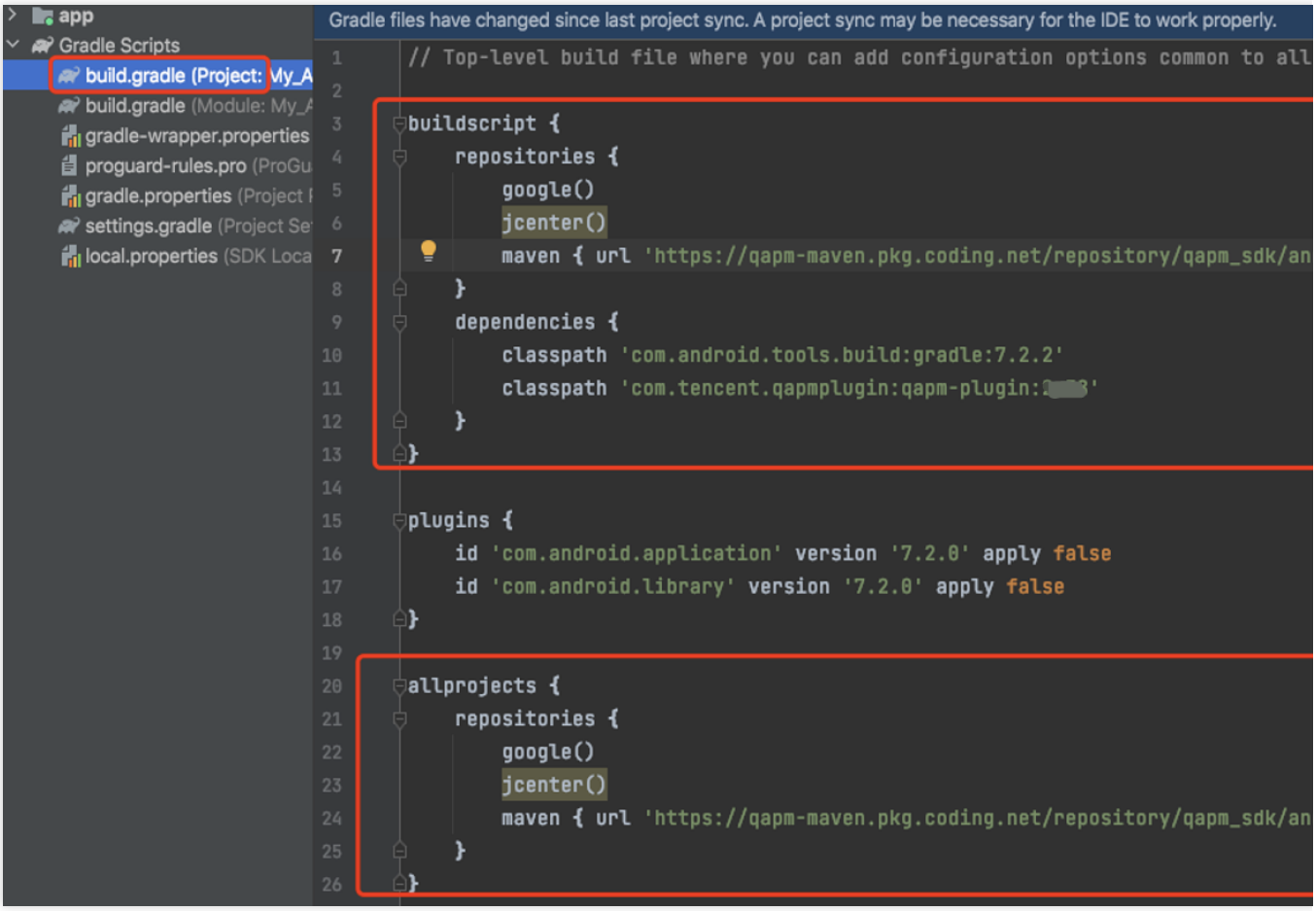
本文指导您使用 Android SDK 的集成与初始化。

操作步骤

步骤一：Gradle 集成

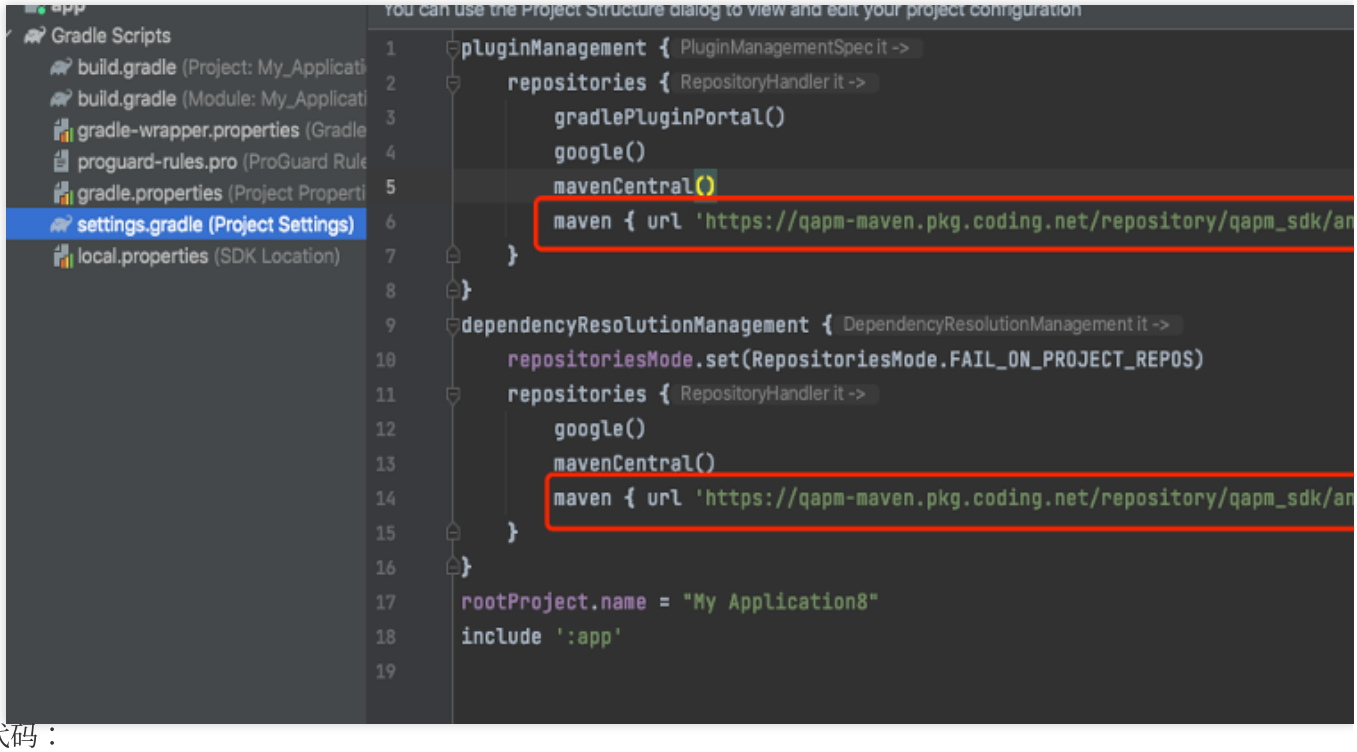
1. **在工程级的 build.gradle 中加入 maven 依赖**。
   i. 添加 buildscript 和 allprojects（gradle 7.0以上不需要添加 allprojects）

gradle 7.0以下版本采用如下配置：

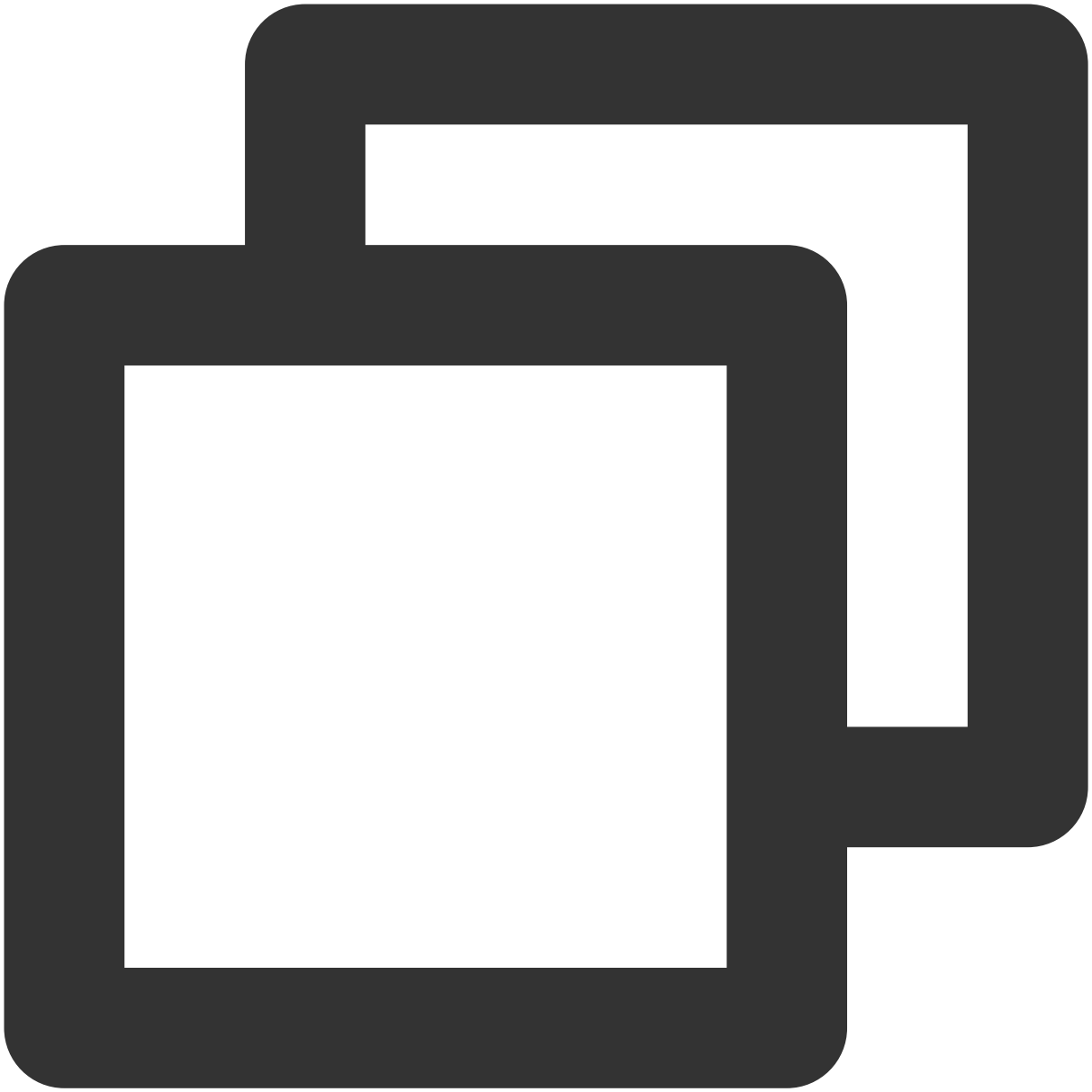


gradle 7.0以上版本采用如下配置：

gradle 7.0以上版本不支持 allprojects。allproject 的 maven 依赖需要在 setting.gradle 中配置。如下图：


Gradle Scripts
build.gradle (Project: My_Applicati
build.gradle (Module: My_Applicati
gradle-wrapper.properties (Gradle
proguard-rules.pro (ProGuard Rule
gradle.properties (Project Properti
settings.gradle (Project Settings)
local.properties (SDK Location)
You can use the Project Structure dialog to view and edit your project configuration
pluginManagement { PluginManagementSpec it ->
repositories { RepositoryHandler it ->
gradlePluginPortal()
google()
mavenCentral()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an
}
}
dependencyResolutionManagement { DependencyResolutionManagement it ->
repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
repositories { RepositoryHandler it ->
google()
mavenCentral()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an
}
}
rootProject.name = "My Application8"
include ':app'

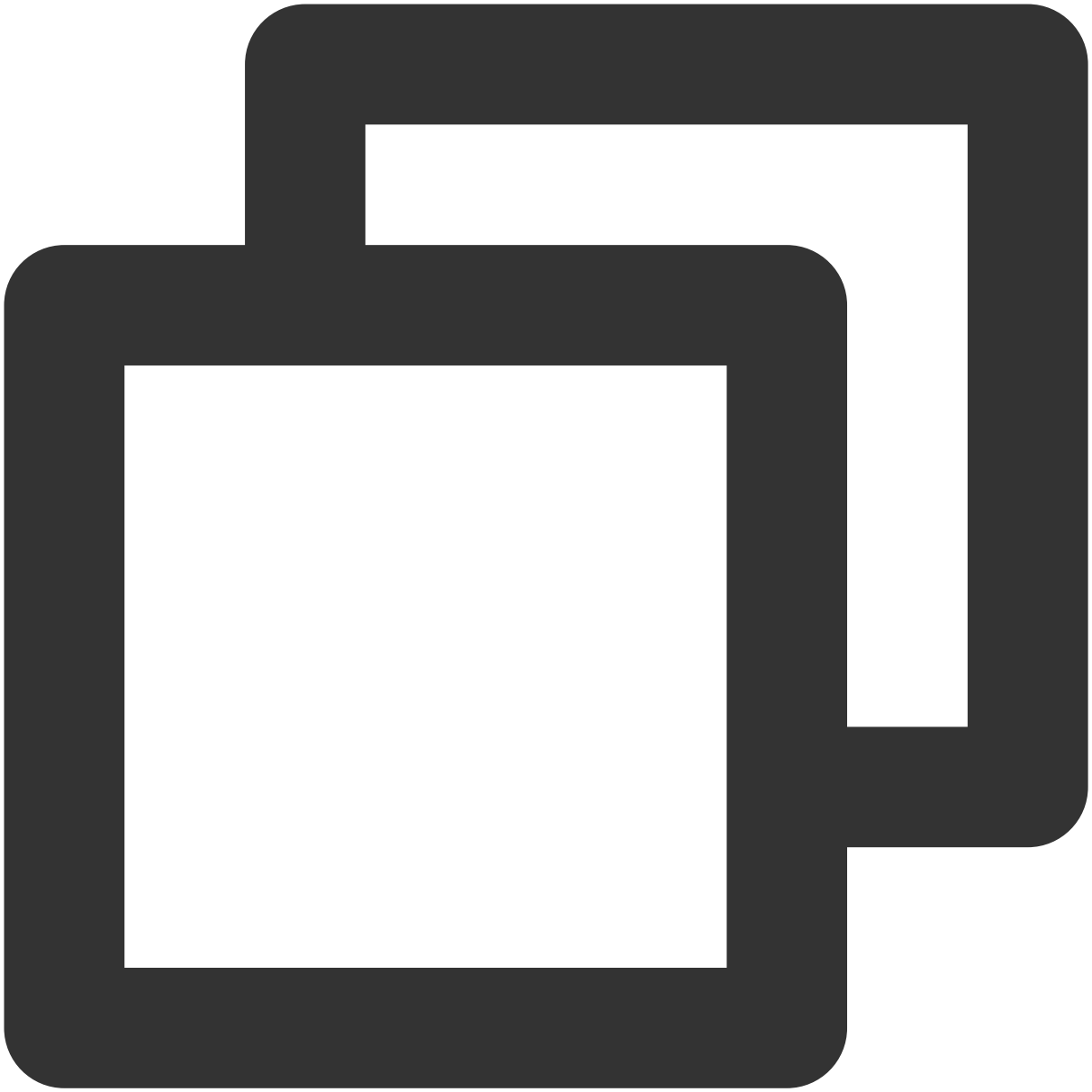
参见代码：



```
maven {url'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_release/'}
```

ii. 在 buildscript 中的 `com.android.tools.build:gradle:*.*.*` 要填写成您的 gradleplugin 版本。如下图：

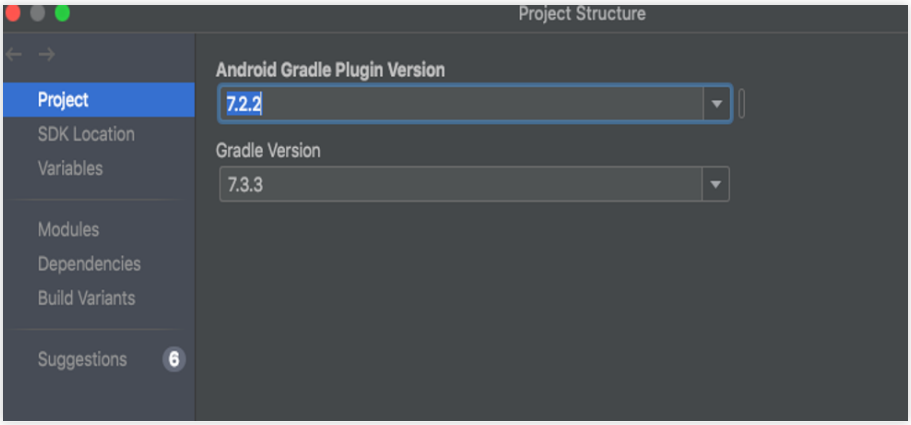

buildscript {
repositories {
google()
jcenter()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/a
}
dependencies {
classpath 'com.android.tools.build:gradle:7.2.2'
classpath 'com.tencent.qapmplugin:qapm-plugin:
}
}
gradle plu



```
classpath 'com.tencent.qapmplugin:qapm-plugin:2.39'
```

说明：

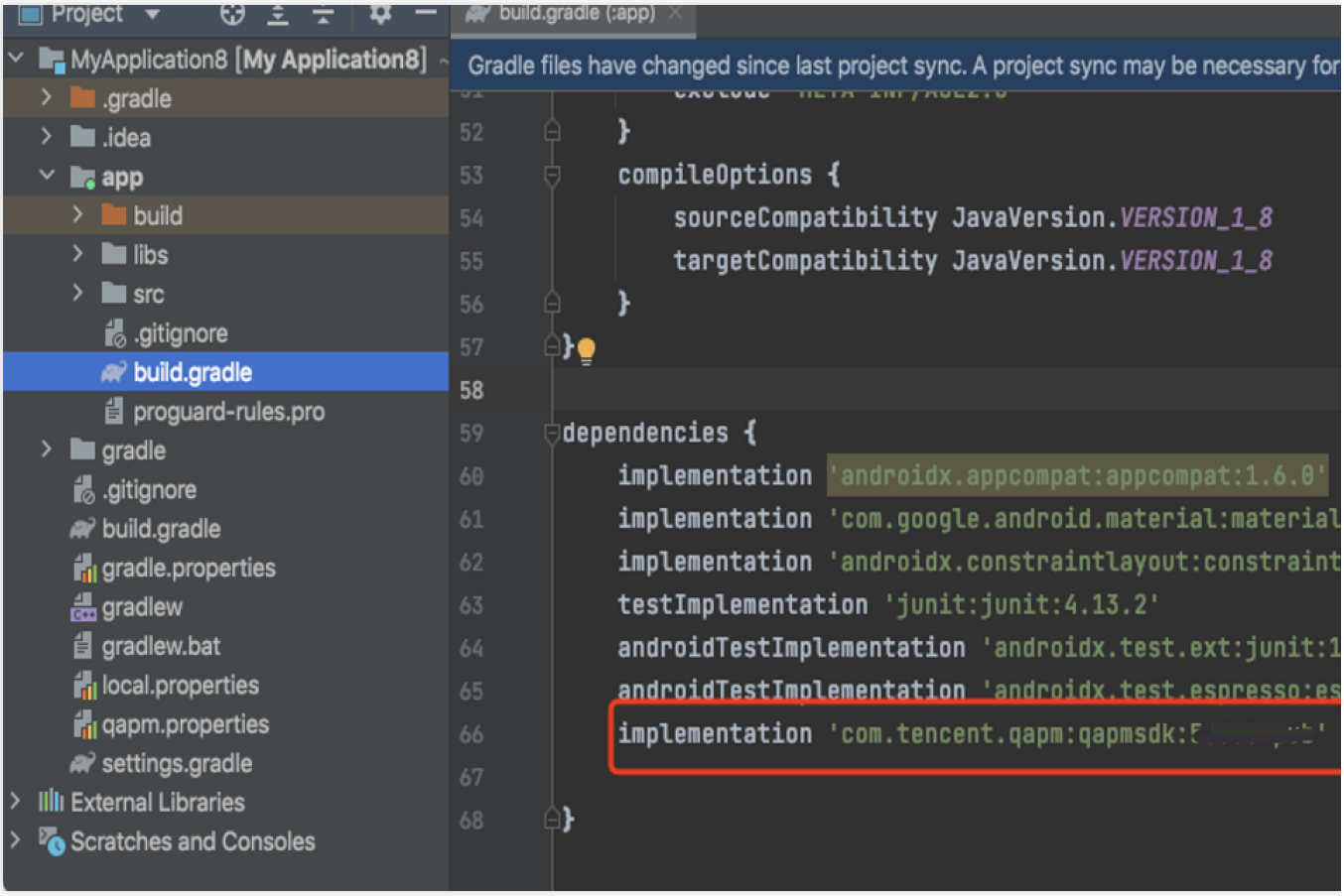
1. Gradle 版本和 gradle plugin 版本可以在菜单 file > project structure 中查看。如下图：



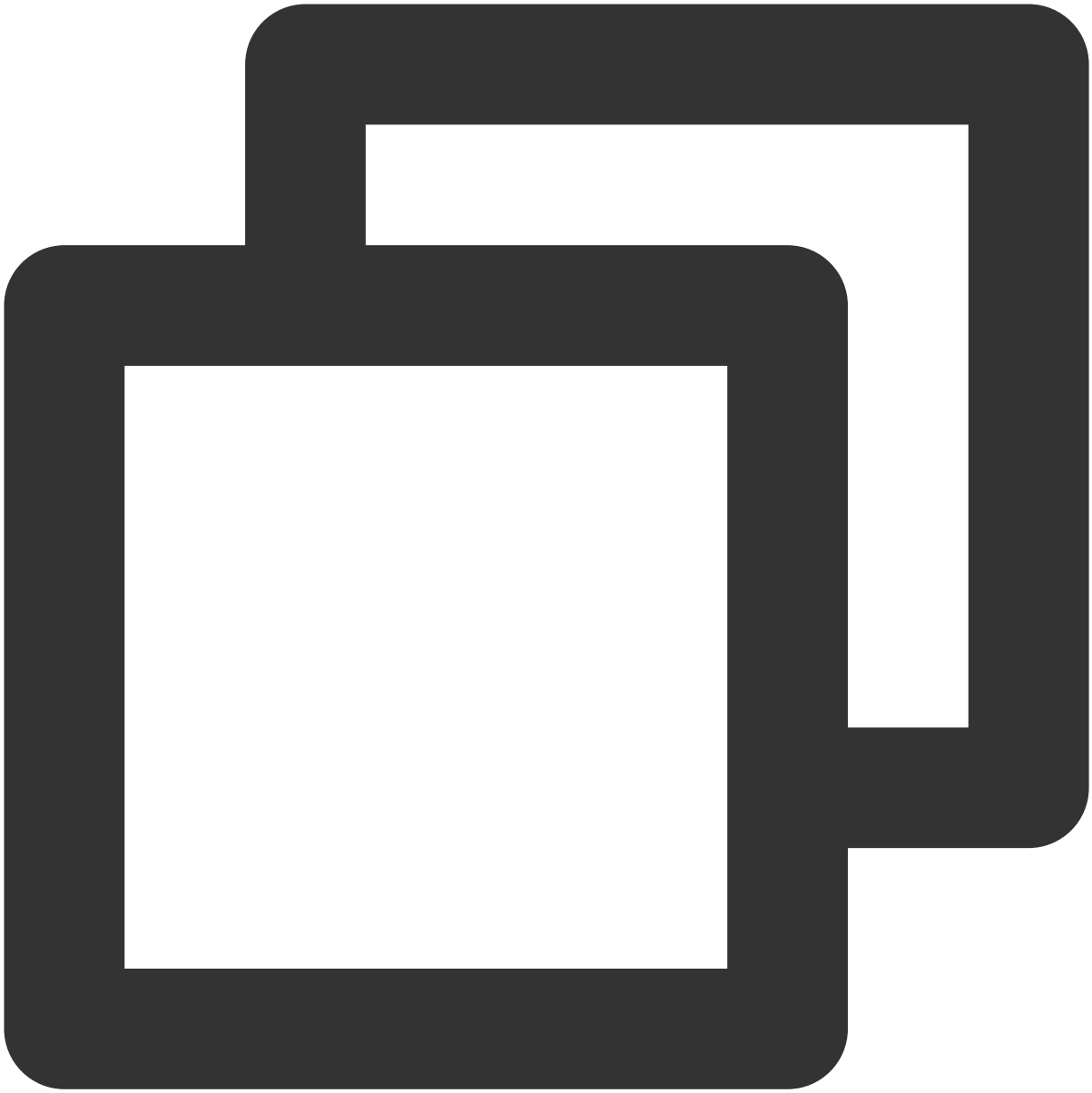
2. Gradle 和 gradle plugin 相对应关系。

插件版本	所需的最低 Gradle 版本
8.1	8.0
8.0	8.0
7.4	7.5
7.3	7.4
7.2	7.3.3
7.1	7.2
7.0	7.0
4.2.0+	6.7.1

2. **在 App 的 build.gradle 中引入模块（studio 版本在3.0以下的请使用 compile 的引用头）**。


Project
MyApplication8 [My Application8]
.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
settings.gradle
External Libraries
Scratches and Consoles
build.gradle (:app)
Gradle files have changed since last project sync. A project sync may be necessary for
52 }
53 compileOptions {
54 sourceCompatibility JavaVersion.VERSION_1_8
55 targetCompatibility JavaVersion.VERSION_1_8
56 }
57 }
58
59 dependencies {
60 implementation 'androidx.appcompat:appcompat:1.6.0'
61 implementation 'com.google.android.material:material
62 implementation 'androidx.constraintlayout:constraint
63 testImplementation 'junit:junit:4.13.2'
64 androidTestImplementation 'androidx.test.ext:junit:1
65
66 implementation 'com.tencent.qapm:qapmsdk:
67
68 }

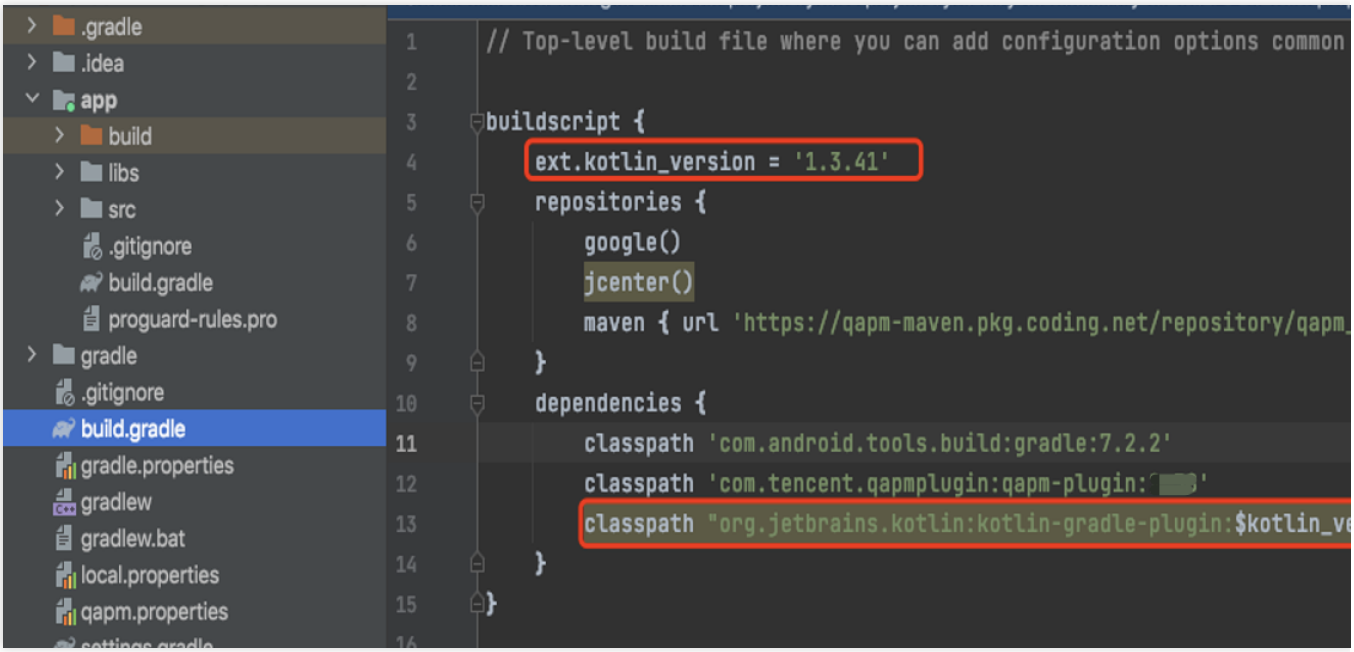
参见代码：



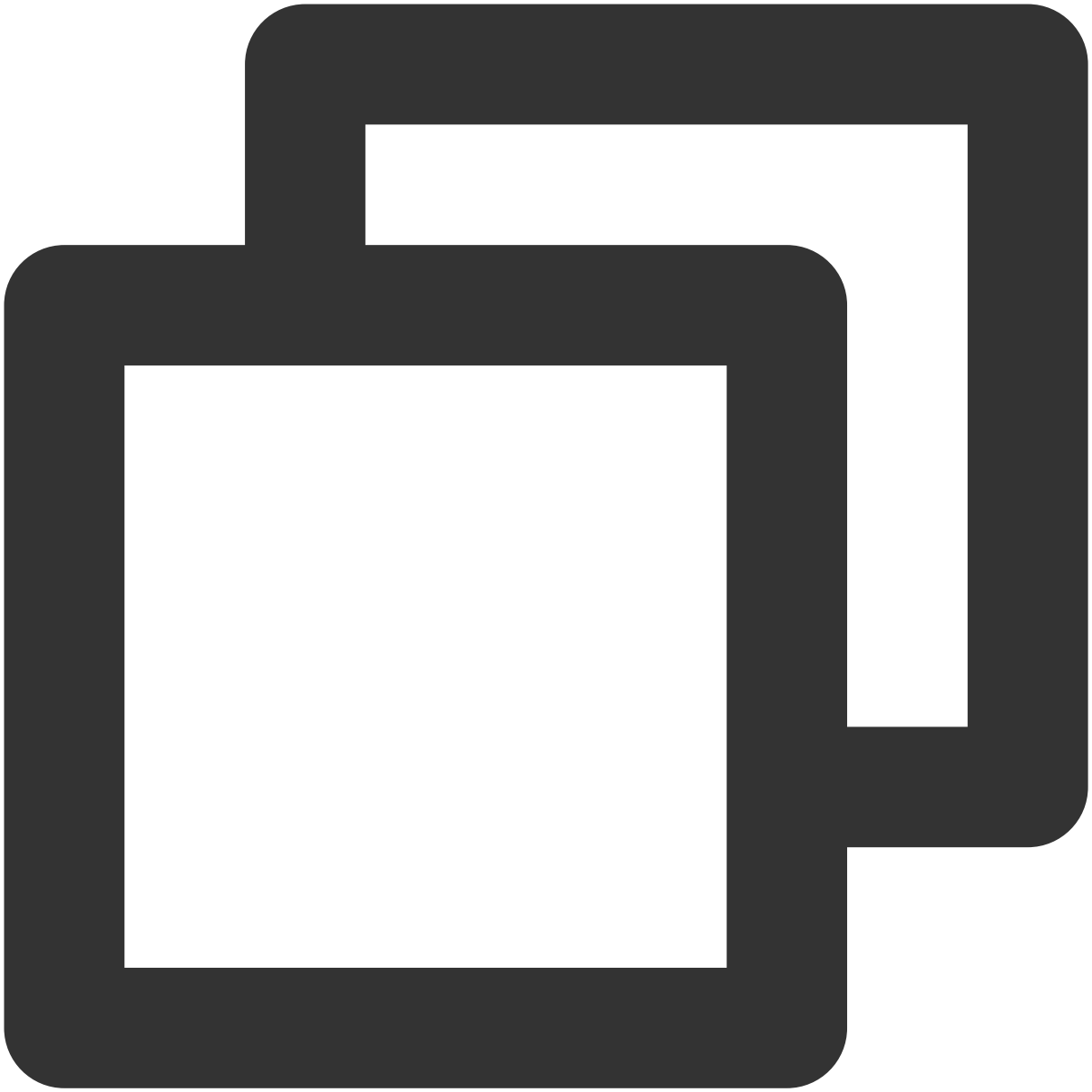
```
implementation 'com.tencent.qapm:qapmsdk:5.3.9-pub'
```

3. **引入 kotlin 依赖**。

i. 在工程级的 build.gradle 下加入如下代码：


.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
// Top-level build file where you can add configuration options common
buildscript {
ext.kotlin_version = '1.3.41'
repositories {
google()
jcenter()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm
}
dependencies {
classpath 'com.android.tools.build:gradle:7.2.2'
classpath 'com.tencent.qapmplugin:qapm-plugin:
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_ve
}
}

参见代码：

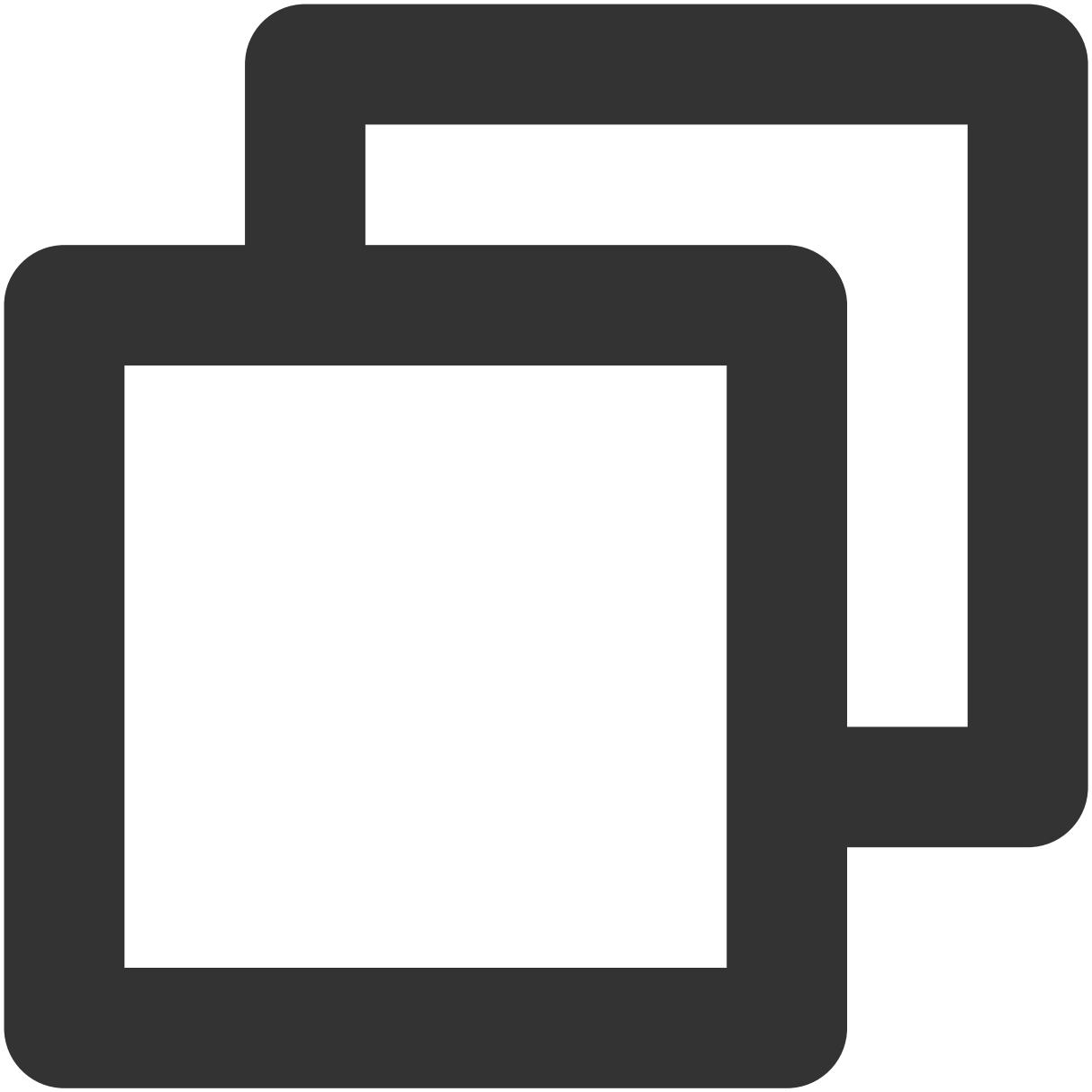


```
ext.kotlin_version = '1.3.41'
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
```

ii. 在 App 的 build.gradle 下加入如下代码：

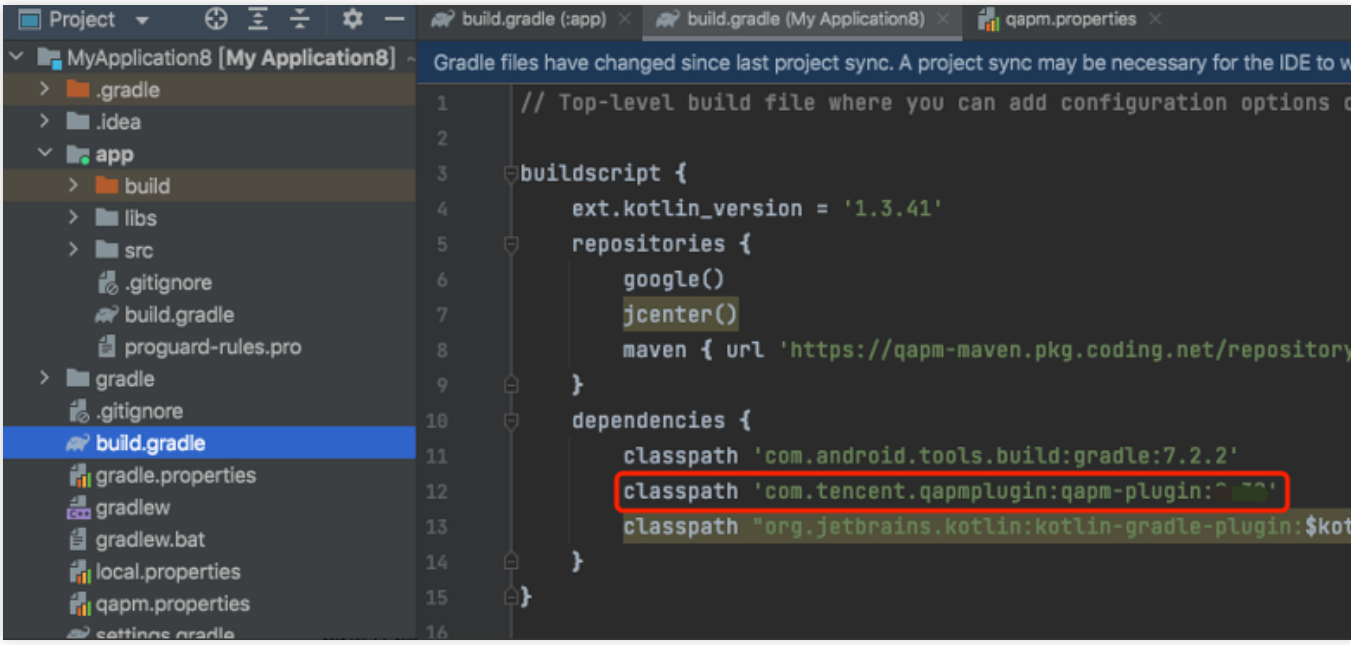
```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.sdkapp"
        minSdkVersion 16
        targetSdkVersion 26
```

参见代码：

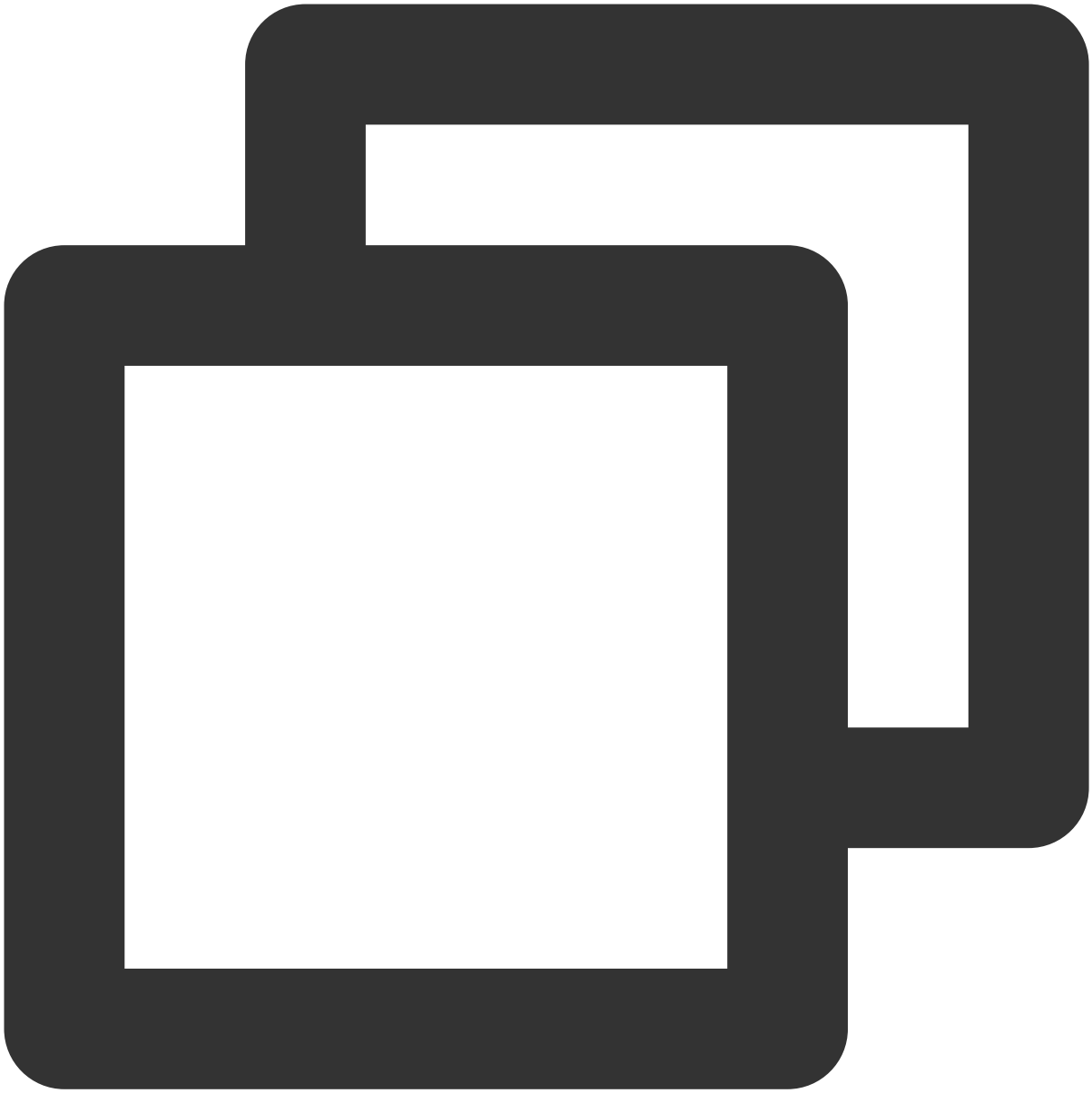


```
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
```

4. **在工程级的 build.gradle 文件中加入以下配置**。


Project
MyApplication8 [My Application8]
.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
build.gradle (:app)
build.gradle (My Application8)
qapm.properties
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to w
// Top-level build file where you can add configuration options
buildscript {
ext.kotlin_version = '1.3.41'
repositories {
google()
jcenter()
maven { url 'https://qapm-maven.pkg.coding.net/repository
}
dependencies {
classpath 'com.android.tools.build:gradle:7.2.2'
classpath 'com.tencent.qapmplugin:qapm-plugin:
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kot
}
}

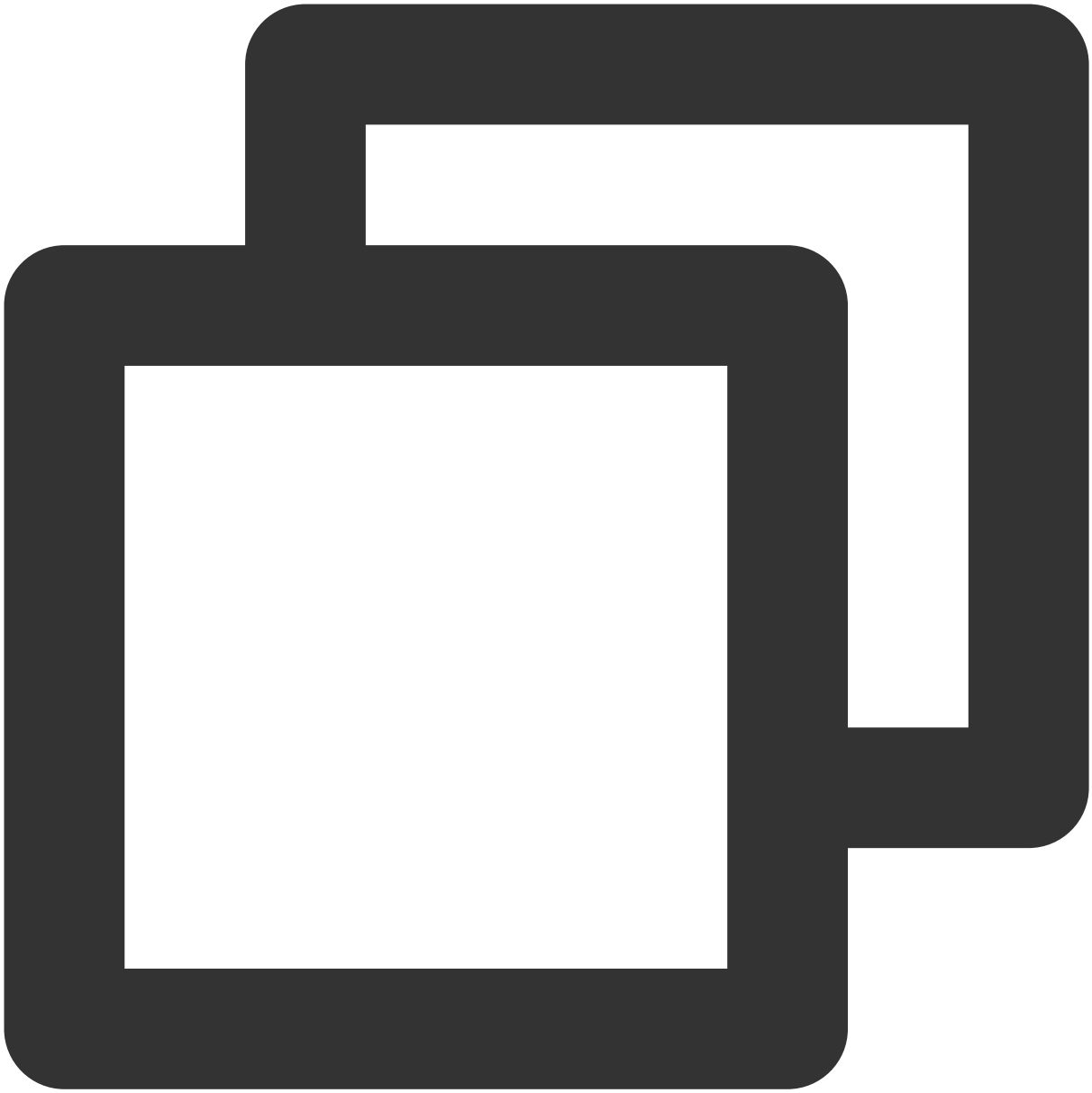
参见代码：



```
classpath 'com.tencent.qapmplugin:qapm-plugin:2.39'
```

5. **在 App 的 build.gradle 文件中加入以下配置。**

参见代码：



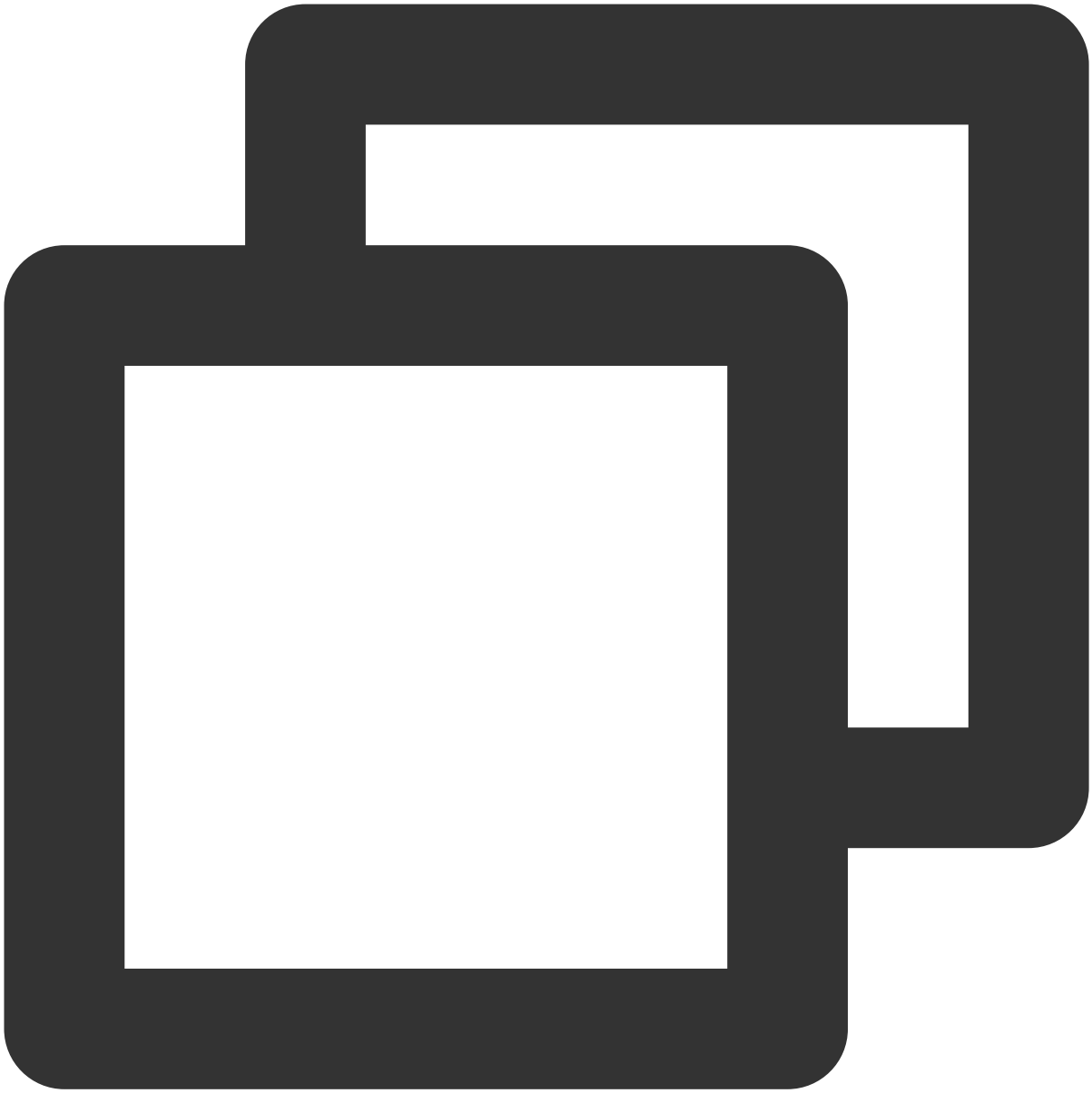
```
    apply plugin: 'qapm-plugin'
    QAPMPluginConfig{
    // 可选,默认为空,请在Application所在的类中输入attachBaseContext,看有没有这个的重写方法，如
}
```

说明：

若未配置4-5项，则会影响启动、网络监控以及自动上传符号表的使用。

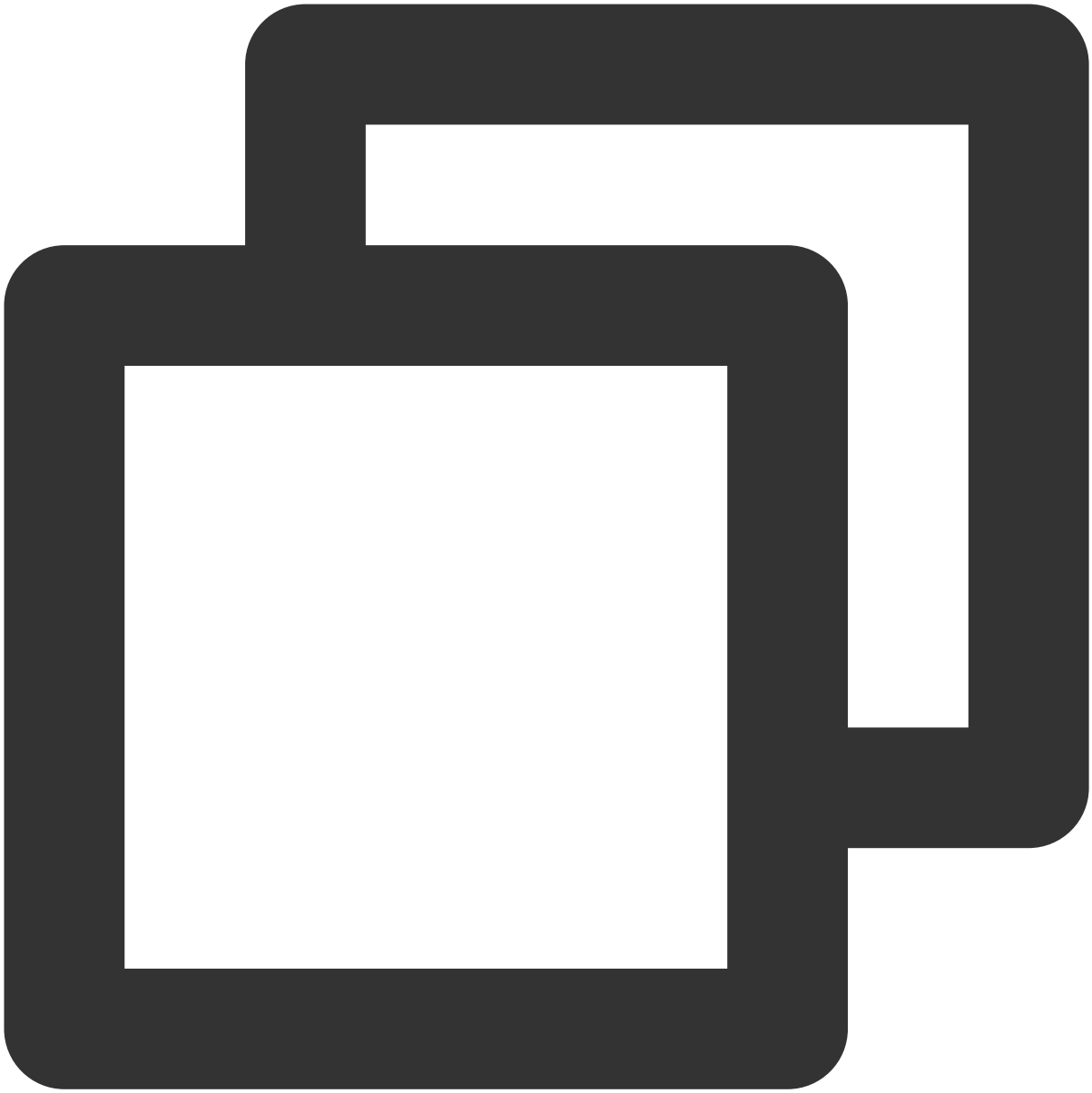
步骤二：参数配置

1. 在 AndroidManifest.xml 中添加以下权限。



```
<!--上报信息所需-->
<uses-permission android:name="android.permission.INTERNET" />
<!--采集信息所需-->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

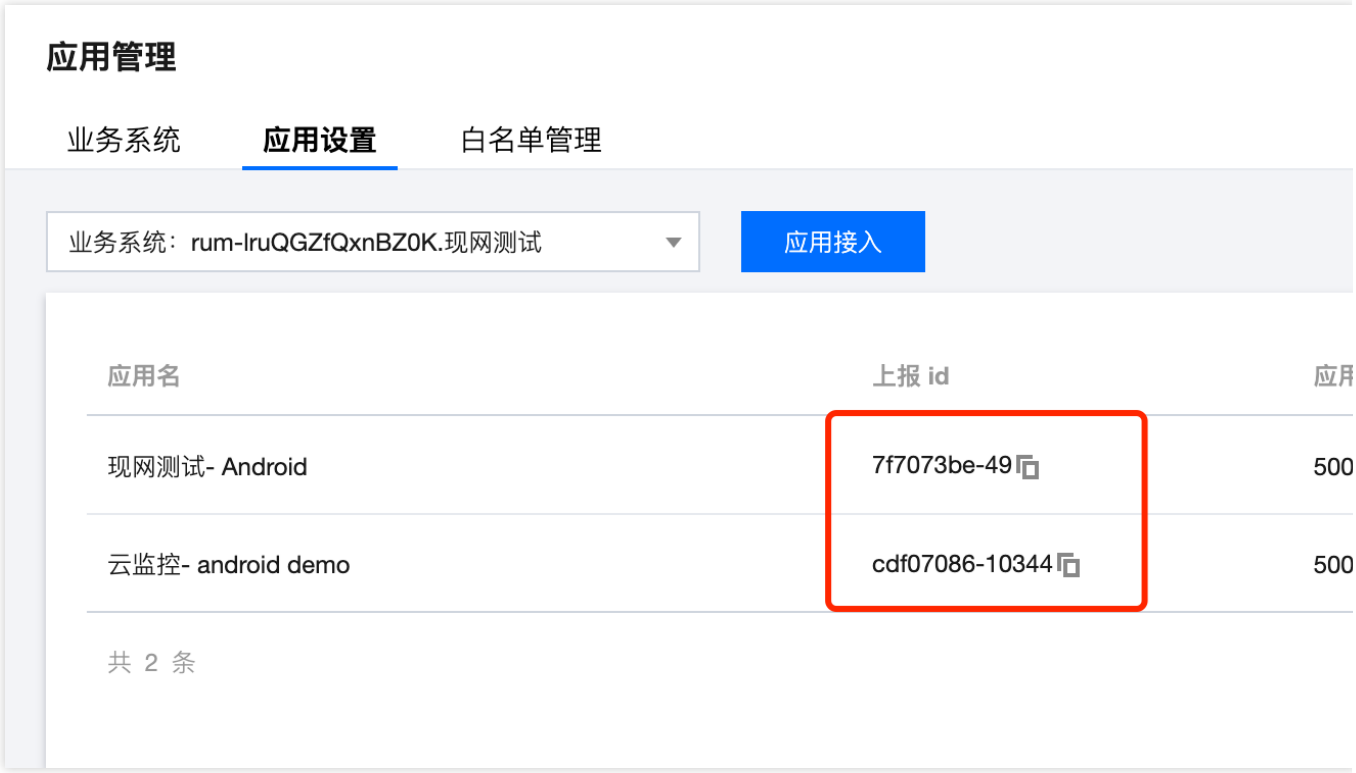
2. 为了避免混淆 SDK，在 App 的 proguard-rules.pro 文件中增加以下配置：



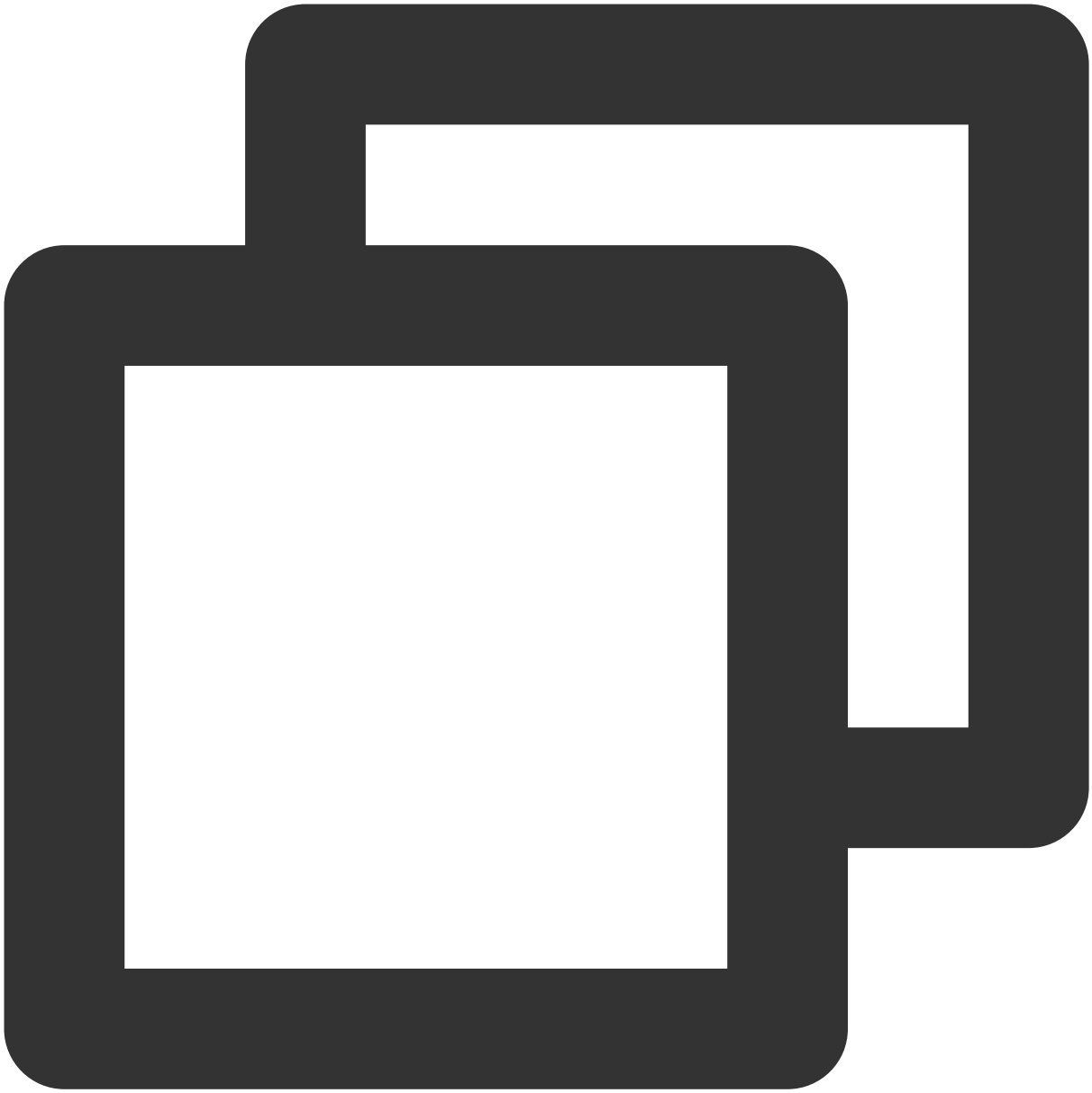
```
-keep class com.tencent.qapmsdk.**{*;}
# 如需要网络监控，请确保okhttp3不被混淆
-keep class okhttp3.**{*;}
```

步骤三：初始化 SDK

1.登录 腾讯云可观测平台 控制台，选择左侧导航栏的**终端性能监控**，点击**应用管理 > 应用设置**后，获取 Appkey（上报 ID）。



1. 拷贝下面代码，并修改其中部分字段。下列项均是必需的接口设置，其余接口配置请参考初始化的接口分析（建议在 Application 中初始化 QAPM）。



```
// 设置手机型号和设备ID。
// 需要传入设备的标识，任意字符串。deviceId（必需！！）
// deviceId可以用来开启白名单，避免数据被抽样上报(崩溃和启动以外的数据抽样率为0.1%)
QAPM.setProperty(QAPM.PropertyKeyDeviceId, "设备的标识");
// 需要传入手机型号（必需！！）
QAPM.setProperty(QAPM.PropertyKeyModel, "填写手机型号");

// 设置Application（必需）
QAPM.setProperty(QAPM.PropertyKeyAppInstance, getApplication());
// 设置AppKey（必需，用于区分上报的产品,该值由终端性能监控的产品配置页面获取，可参考上一步骤 ）
QAPM.setProperty(QAPM.PropertyKeyAppId, "YourAppKey");
```

```
// 设置产品版本，用于后台检索字段（必需）
QAPM.setProperty(QAPM.PropertyKeyAppVersion, "YourApp Version");
// 设置UUID，用于拉取被混淆堆栈的mapping （必需，若使用了QAPM符号表上传插件，可以直接使用该变量）
// 该变量会在build时生成，爆红不用理
QAPM.setProperty(QAPM.PropertyKeySymbolId, BuildConfig.QAPM_UUID);
// 设置用户ID(userId)，用于后台检索字段（必需）
// userId可以用来开启白名单，避免数据被抽样上报(崩溃和启动以外的数据抽样率为0.1%)
QAPM.setProperty(QAPM.PropertyKeyUserId, "123456");
// 设置Log等级，（可选），线上版本请设置成QAPM.LevelOff或者 QAPM.LevelWarn
QAPM.setProperty(QAPM.PropertyKeyLogLevel, QAPM.LevelInfo);
// 设置QAPM的外网上报域名（必需）。固定值，https://app.rumt-zh.com
QAPM.setProperty(QAPM.PropertyKeyHost,"https://app.rumt-zh.com");
// 启动QAPM
QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeStable);
```

说明：

AppKey 可参考前一个步骤，在 **终端性能监控 > 应用管理**页面获取。

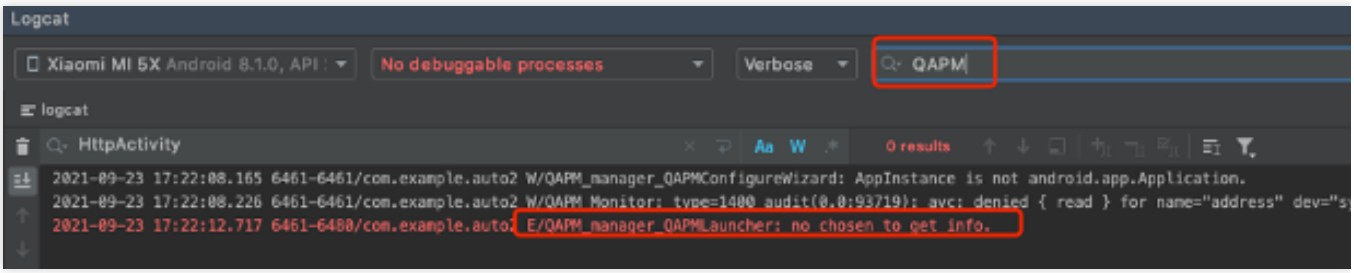
崩溃和启动数据是全量上报，其他数据因为数据数目过多，采取抽样上报，抽样率为0.1%（千分之一）。如果需要全量上报，可以开启白名单，App 将会在下次启动时改变抽样率。

可以将设置好 userId 或者 deviceId 通过**终端性能监控 > 应用管理**页面页面添加白名单里，开启白名单。

多个进程需要各自初始化 QAPM。

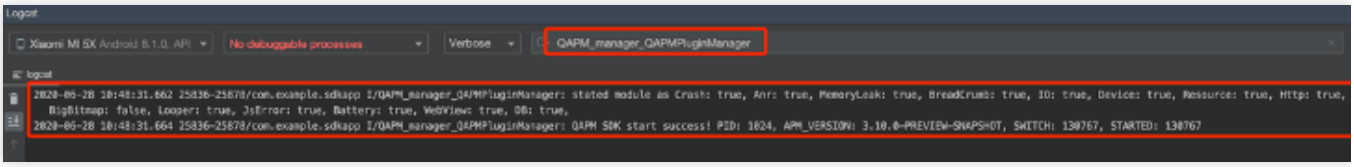
步骤四：接入验证

1. 若打印以下日志，代表该用户未被抽样命中，需重新设置下抽样率。



参见 TAG : QAPM_manager_QAPMLauncher。

2. 若打印以下日志，则代表初步接入成功，可以验证数据上报/尝试开启高级功能。



参见 TAG : QAPM_manager_QAPMPluginManager。

初始化的接口分析

1. 接口名称：public static QAPM setProperty(int key, Stringvalue)
用途：设置 QAPM 的相关参数
参数：key - 需要设置的 Key
可选项：
QAPM.PropertyKeyLogLevel：开启日志等级（建议 Debug 版本开启 QAPM.LevelDebug，release 版本开启 QAPM.LevelWarn）。
QAPM.PropertyNeedTranslate：堆栈是否需要翻译，这里默认是需要翻译的，如果 apk 是没有混淆的需要传入 false，否则前端可能会全部展示为 unTranslated。
2. public static boolean beginScene(String sceneName, int mode)
用途：开启监控
参数： sceneName-场景名，mode-开启的功能。
可选项：
QAPM. ModeAll：开启全功能（建议研发流程内开启。包含区间性能、crash、anr、webview（页面加载、JsError、网络）、http 网络）
QAPM.ModeStable：开启部分功能（建议外发版本开启。包含区间性能、crash、anr、webview（页面加载、JsError、网络）、http 网络）
QAPM.ModeWebView：开启 WebView 页面加载监
QAPM.ModeJsError：开启 WebView 的 JS 异常监控
QAPM.ModeHTTPInWeb：开启 WebView 的网络监控
QAPM.ModeHTTP：开启网络监控
注意：
1. 正式版本建议开启 QAPM.ModeStable，研发版本建议开启 QAPM.ModeAll。
2. 默认会开启 ModeStable 功能，可基于 ModeStable 使用或运算的方式增加需要的功能，如开启 Stable 和内存触顶： beginScene(“Stable&触顶”，QAPM.ModeStable| QAPM.ModeCeiling)。可使用异或运算符排除不需要的功能，如关闭网络：QAPM.ModeStable^QAPM.ModeHTTP。
3. public static boolean endScene(String sceneName, long mode)
用途： 结束监控（只针对掉帧和区间性能采集有效）
参数：sceneName-需要关掉的场景名（与 beginScene 的要相对应）

可选项：

QAPM.ModeDropFrame：关闭掉帧监控

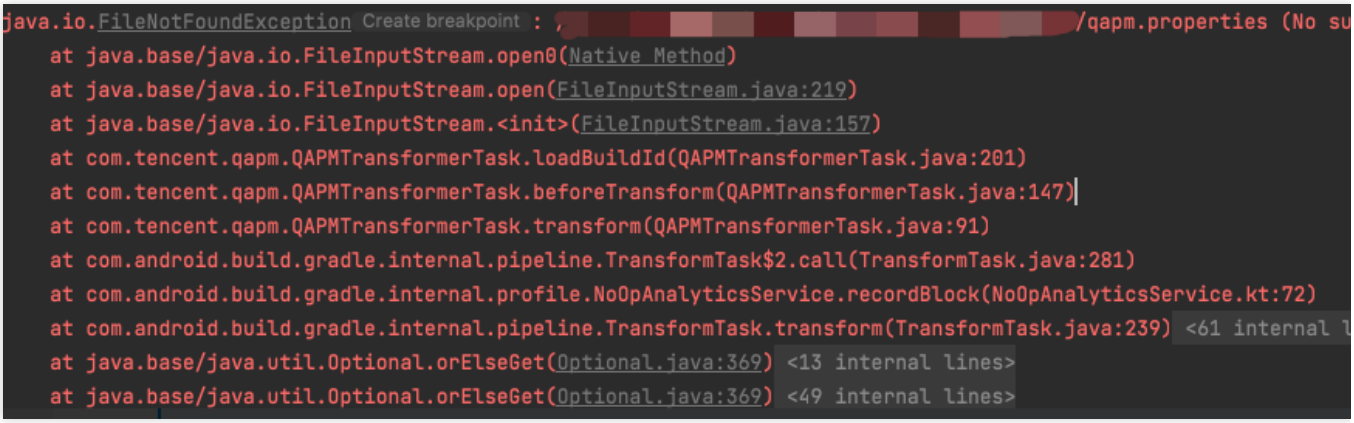
QAPM.ModeResource：关闭区间性能监控

其他问题

说明：

通过 qapm 插件编译打包 App 时，App 需要一个 uuid 作为构建 id，如果项目目录下存在 qapm.properties 文件，并且文件里 qapm_uuid 属性的值存在，该值将被作为构建 id，否则插件会随机生成一个构建 id。

qapm-plugin 2.39及之前版本在编译 App 的过程中会报 IO 错误：java.io.FileNotFoundException，qapm.properties (No such file or directory)。



该报错仅在编译期间产生，可以忽略，不会影响 App 运行。