

Real User Monitoring

Connection Guide

Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Connection Guide

Web Use Cases

Installation and Initialization

Log Reporting

aid

Instance Methods

Allowlist

Hook Functions

Error Monitoring

Performance Monitoring

Environment Control

Configuration Guide

Mini Program Use Cases

Installation and Initialization

Log Reporting

aid

Instance Methods

Allowlist

Hook Functions

Error Monitoring

Performance Monitoring

Configuration Guide

IOS Use Cases

Integration and Initialization

Android Use Cases

Integration and Initialization

Connection Guide

Web Use Cases

Installation and Initialization

Last updated : 2024-01-22 19:25:42

You can install the SDK through CDN or npm.

Installing and Initializing SDK Through CDN

1. Import the following code to the `<head></head>` tags on the HTML page:



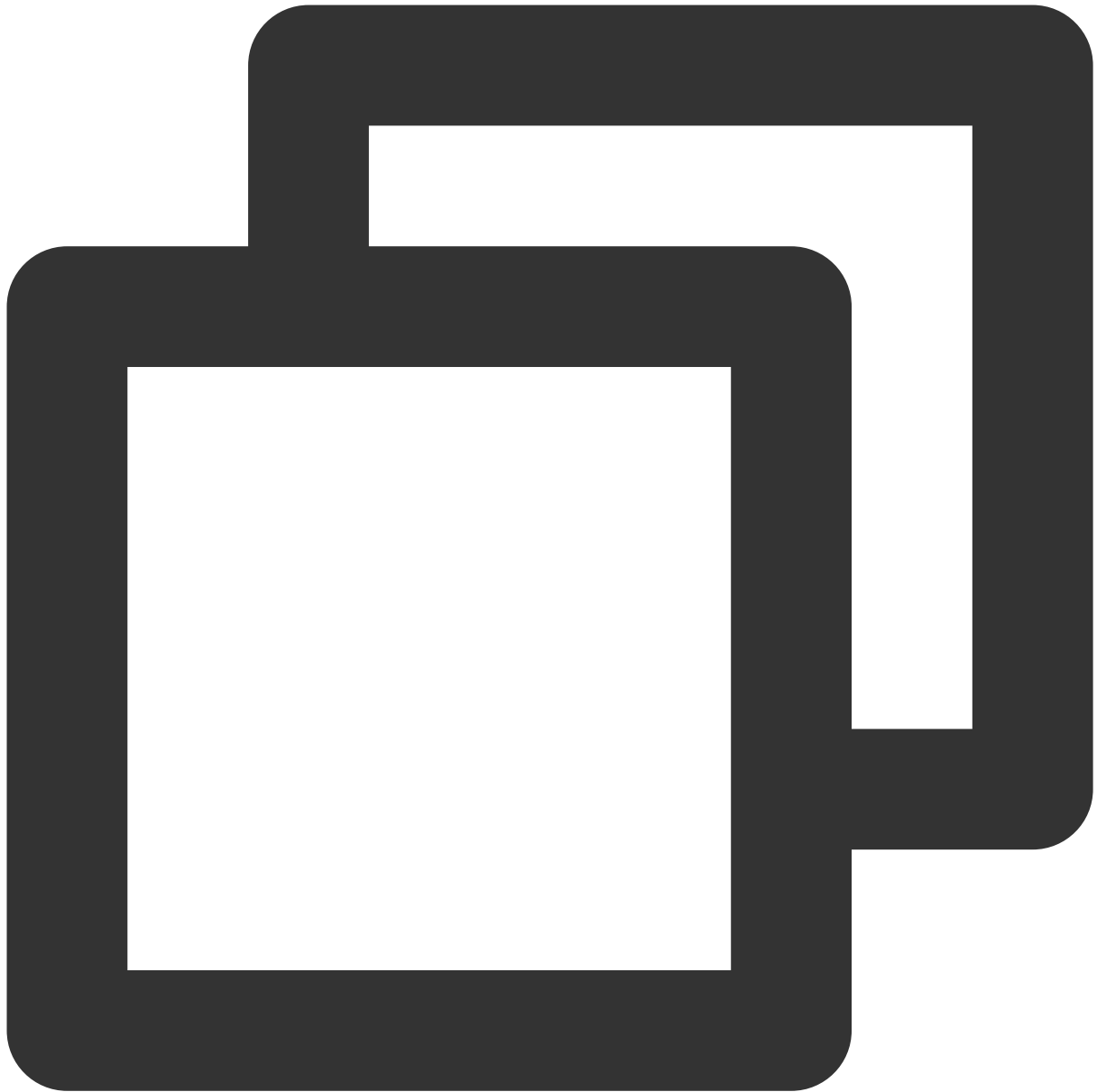
```
<script src="https://cdn-go.cn/aegis/aegis-sdk/latest/aegis.min.js"></script>
```

As CDN uses the h3-Q050 protocol, `cache-control` is set to `max-age=666` by default. If you need to modify `cache-control`, add the `max_age` parameter. For example:



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

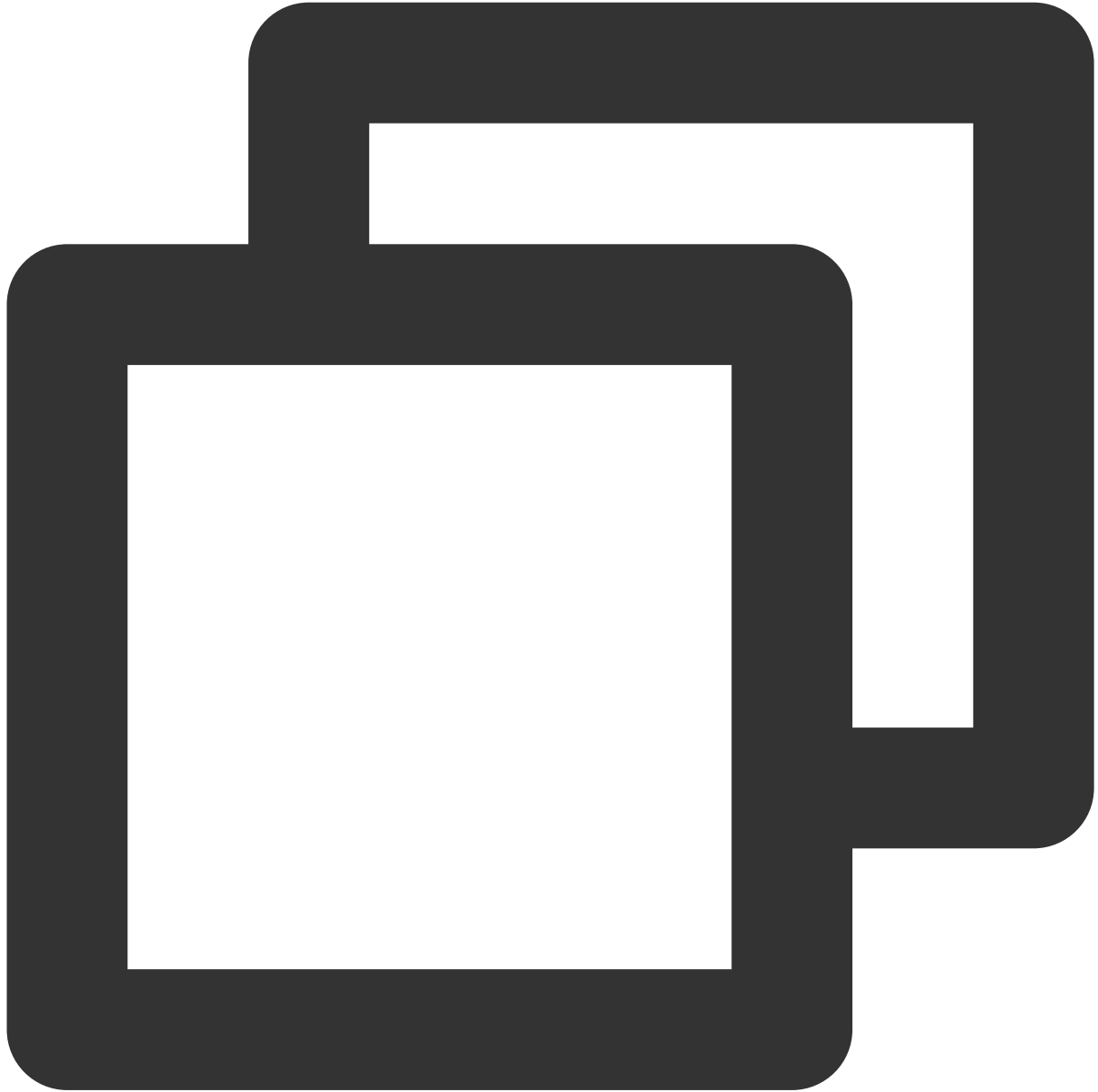
2. Create an Aegis instance and pass in the corresponding configuration in the following steps to initialize the SDK:



```
const aegis = new Aegis({
  id: 'pGUVFTCyewhxxxxxx', // Application ID, that is, reported ID
  uin: 'xxx', // User UIN (optional)
  reportApiSpeed: true, // API speed test
  reportAssetSpeed: true, // Static resource speed test
  spa: true // Enable PV calculation during SPA application page redirect
});
```

Installing and Initializing SDK Through npm

1. Run the following command to install `aegis-web-sdk` in your npm repository:



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

2. Create an Aegis instance and pass in the corresponding configuration in the following steps to initialize the SDK:



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

const aegis = new Aegis({
  id: 'pGUVFTCZyewhxxxxxx', // Application ID, that is, reported ID
  uin: 'xxx', // User UIN (optional)
  reportApiSpeed: true, // API speed test
  reportAssetSpeed: true, // Static resource speed test
  spa: true // Enable PV calculation during SPA application page redirect
})
```

Note:

To avoid missing any data, you should initialize the SDK as early as possible. After you install and initialize the SDK, you can use the following RUM features:

Error monitoring: Monitoring of JavaScript execution errors, Promise errors, Ajax request exceptions, resource loading failures, and return code exceptions, PV reporting, and allowlist detection.

Speed test features: Speed tests for page performance, APIs, and static resources.

Statistics collection and analysis: You can analyze data in multiple dimensions in [Data Overview](#).

Note:

`aegis-sdk` uses `https://aegis.qq.com` as the domain name to be reported by default. You can use the `hostUrl` parameter to configure the domain name to be reported.

For regions in the Chinese mainland, you can choose `https://tamaegis.com` as the domain name to be reported.

For Singapore region, you can choose `https://rumt-sg.com` as the domain name to be reported.

Log Reporting

Last updated : 2024-01-22 19:25:42

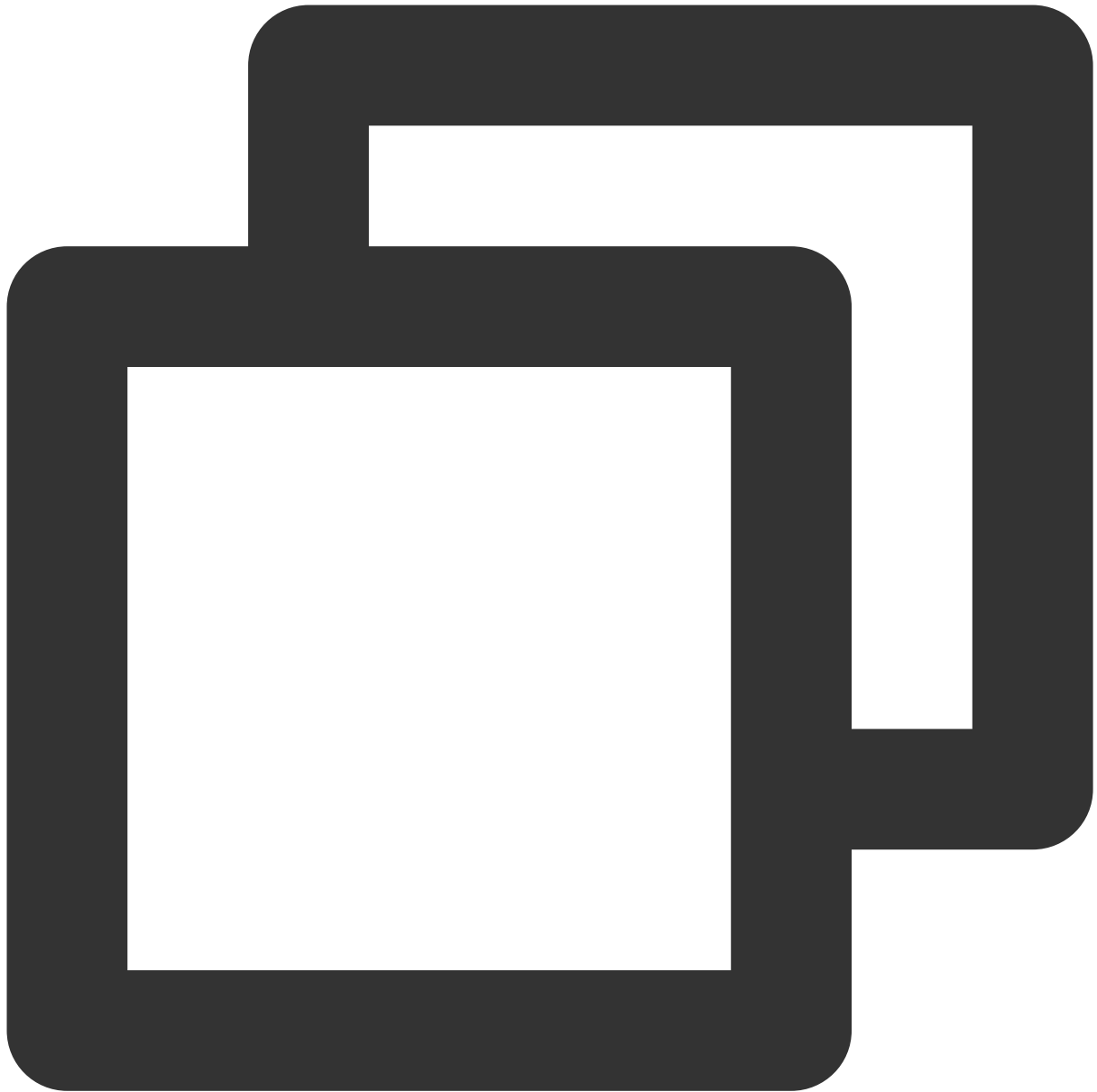
The log query and offline log features can be used normally only after logs are reported. This document describes how to report logs to RUM.

Prerequisites

Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

Log Reporting

Pass in the following parameters to configure the SDK and report logs:



```
// `info` can report any string, number, array, and object, but it reports data only on the console
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// You can also report a specified object and pass in the `ext` and `trace` parameters
// You must pass in the `msg` field in this case
aegis.info({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
```

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

// Different from `info`, `infoAll` reports full data
aegis.infoAll({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// `error` indicates a JavaScript error log, whose full data will also be reported.
aegis.error({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
aegis.error(new Error('Actively report an error'));

// The default log type of `aegis.report` is `report`, but currently you can pass i
aegis.report({
  msg: 'This is an Ajax error log',
  level: Aegis.LogType.AJAX_ERROR,
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
```

aid

Last updated : 2024-01-22 19:25:42

`aid` is a unique ID assigned by the Aegis SDK to each user device. It is stored in the `localStorage` of the browser and used to distinguish between users for UV calculation. It will be updated only when the user clears the browser cache.

Prerequisites

Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

aid

You can use the following algorithm to customize the `aid` reporting rules:



```
async getAid(callback: Function) {  
  // In some cases, an error will occur when `localStorage` is manipulated  
  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?.'';  
  }  
}
```

Note:

To use the `aid` constructed by yourself as a reporting rule, you need to configure an `aid` verification rule in the format of `/^[@=.0-9a-zA-Z_-]{4,36}$/` on the backend.

Instance Methods

Last updated : 2024-01-22 19:25:42

RUM provides various instance methods for data reporting. You can use them to modify the instance configuration and customize the events and resources for speed test to be reported.

Currently, RUM provides the following Aegis instance methods:

Parameter	Description
setConfig	Passes in the configuration object, which contains information such as user ID and UIN.
info	Is a main reporting field to report allowlist logs. A log will be reported to the backend only in the following cases: 1. The user who opens the page is in the allowlist. 2. The page has an error.
infoAll	Is a main reporting field to report allowlist logs. The only difference between it and <code>info</code> is as follows: <code>info</code> reports the logs of only specified users, while <code>infoAll</code> reports the logs of all users.
error	Is a main reporting field to report the error information.
report	Reports the information of a log in any type.
reportEvent	Reports a custom event.
reportTime	Reports a custom resource for speed test.
time	Reports a custom resource for speed test and is used together with <code>timeEnd</code> to calculate and report data between two time points.
timeEnd	Reports a custom resource for speed test and is used together with <code>time</code> to calculate and report data between two time points.
destroy	Terminates the Aegis instance.

Prerequisites

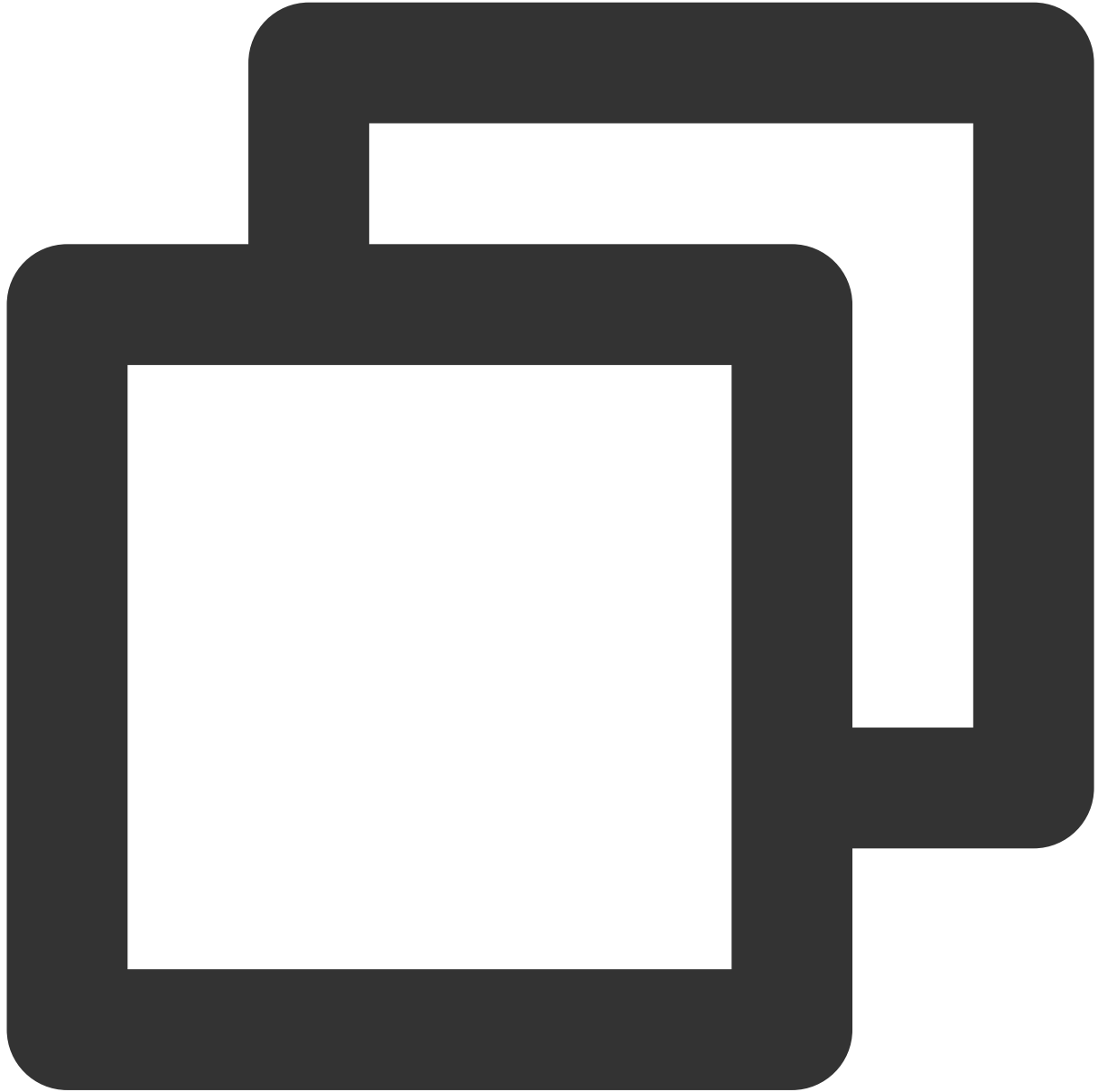
Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

Instance Methods

setConfig

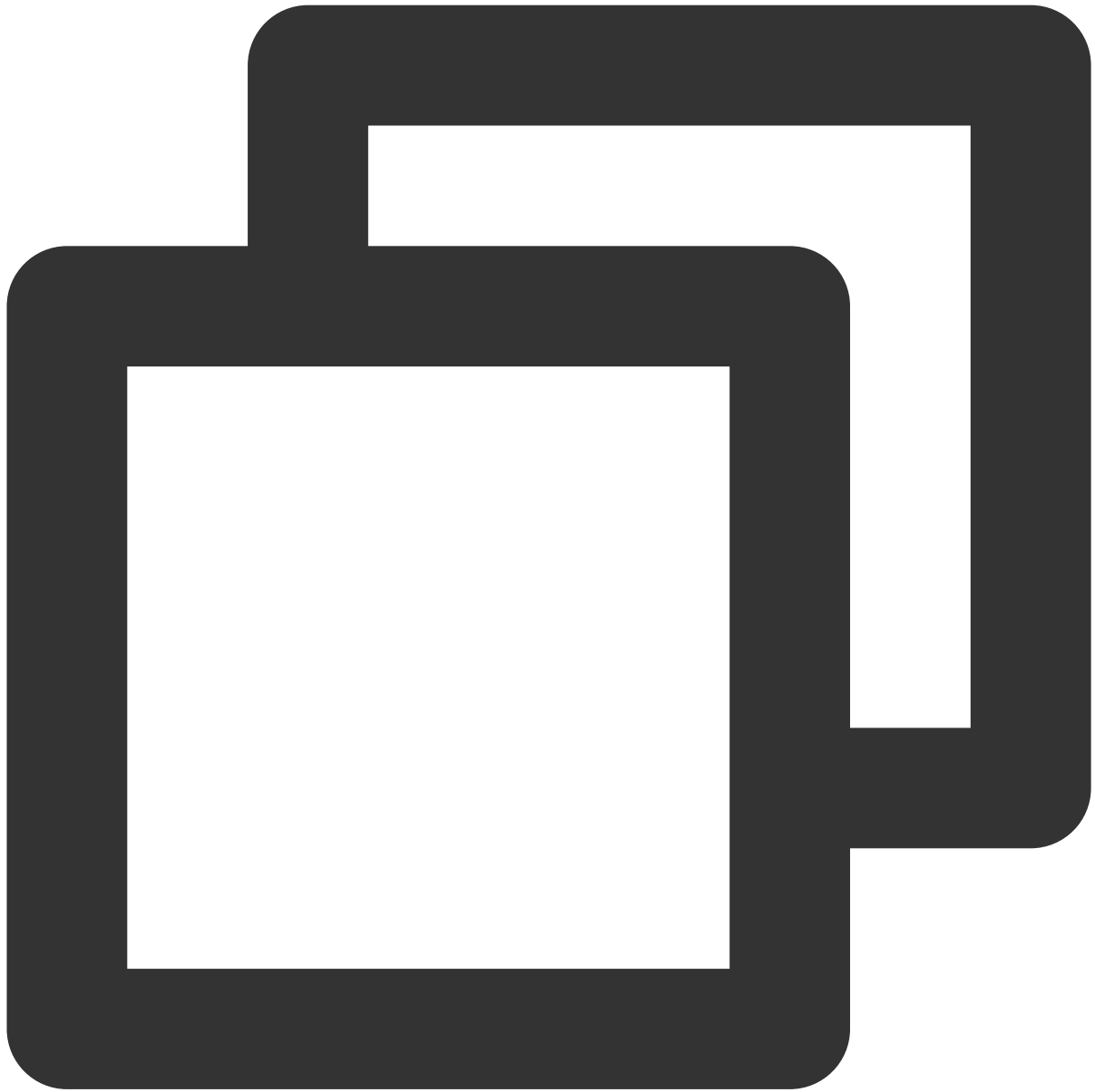
This method is used to modify the instance configuration in the following use cases:

1. You can get the user UIN and pass in two instance objects of user ID and UIN at the same time for instantiation:



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

2. Generally, the `uin` cannot be directly obtained in the beginning. If instantiation cannot be completed during the period when `uin` is obtained, RUM cannot listen on the errors occurring in this period. To solve this, you can pass in the ID first for instantiation and then import `setConfig` to pass in `uin` as follows:



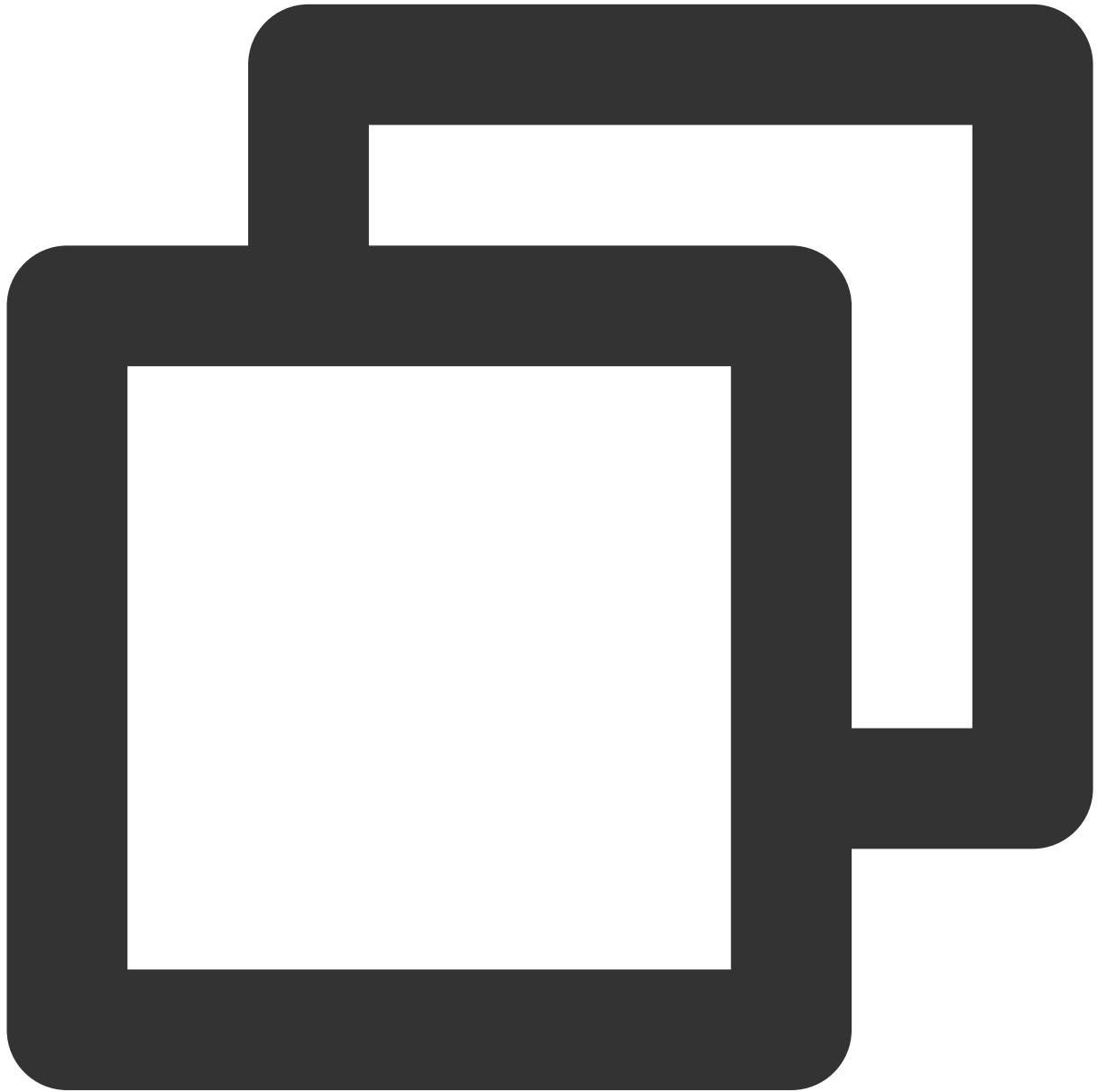
```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx'
})

// After `uin` is obtained
aegis.setConfig({
  uin: '6666'
```

```
})
```

`info` , `infoAll` , `error` , and `report`

These are the main reporting methods provided by RUM.



```
// `info` can report any string, number, array, and object, but it reports data only on the client side
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// You can also report a specified object and pass in the `ext` and `trace` parameters
```

```
// You must pass in the `msg` field in this case
aegis.info({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// Different from `info`, `infoAll` reports full data
aegis.infoAll({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// `error` indicates a JavaScript error log, whose full data will also be reported.
aegis.error({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
aegis.error(new Error('Actively report an error'));

// The default log type of `aegis.report` is `report`, but currently you can pass i
aegis.report({
  msg: 'This is an Ajax error log',
  level: Aegis.LogType.AJAX_ERROR,
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
```

reportEvent

This method can be used to report a custom event, and the system will automatically collect the metrics of the reported event, such as PV and platform distribution.

`reportEvent` supports the string and object data types for the parameters to be reported.

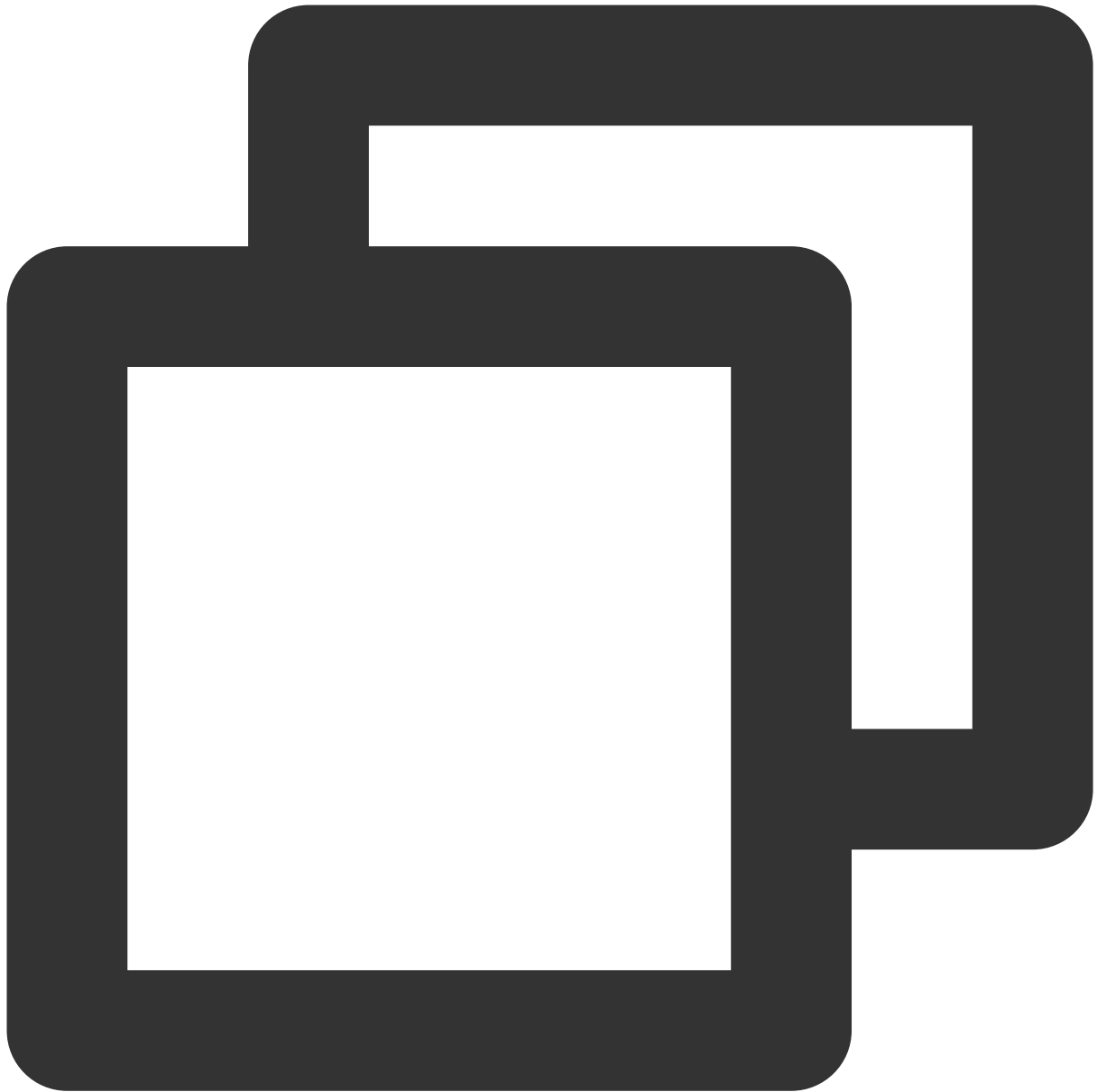
String data type



```
aegis.reportEvent('The XXX request succeeded');
```

Object data type

`ext1` , `ext2` , and `ext3` use the parameters passed in when you use `new Aegis` . During custom event reporting, you can overwrite their default values.



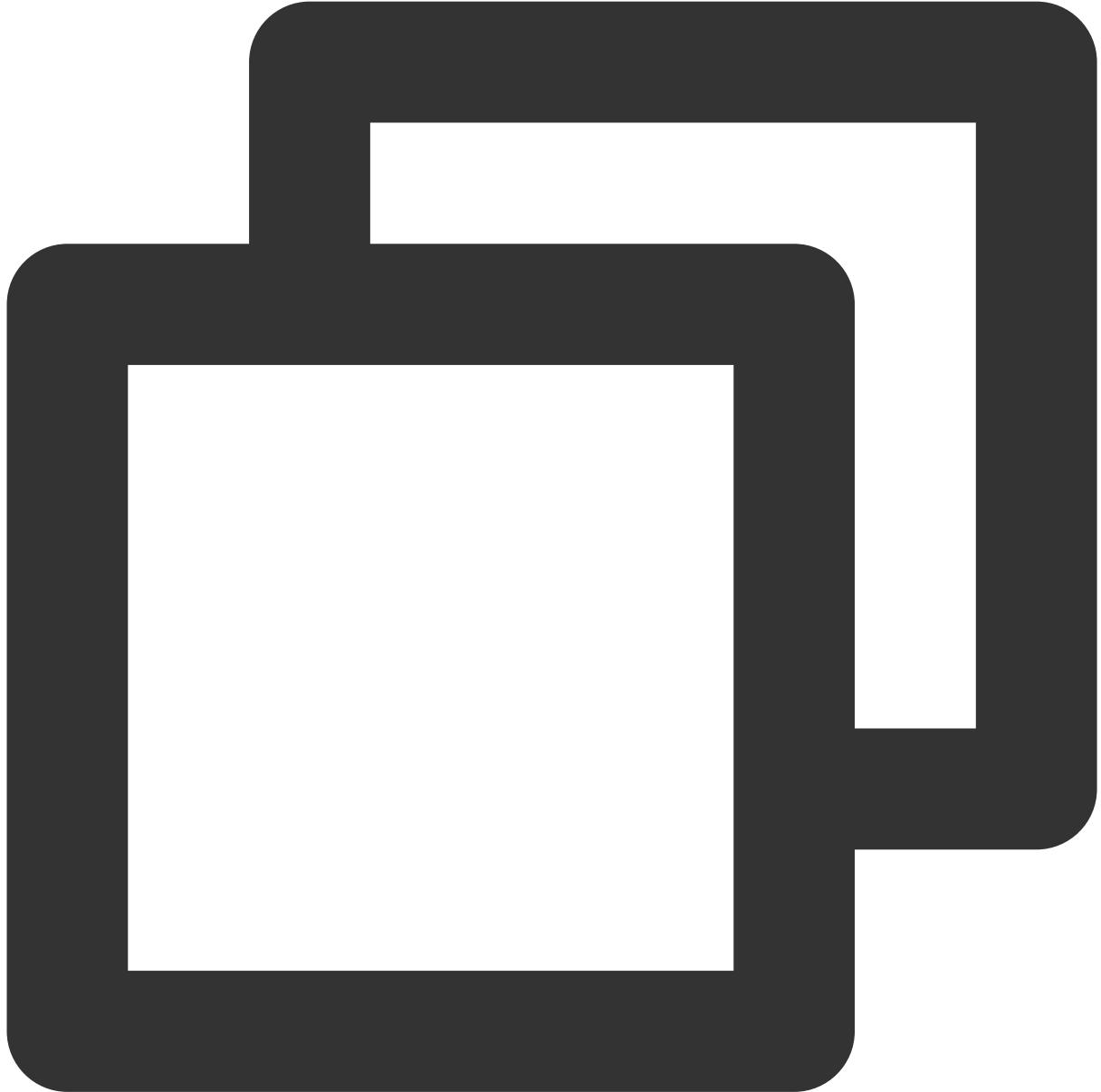
```
aegis.reportEvent({
  name: 'The XXX request succeeded', // Required
  ext1: 'Additional parameter 1',
  ext2: 'Additional parameter 2',
  ext3: 'Additional parameter 3',
})
```

Note:

The keys of the three additional parameters are fixed, which can only be `ext1` , `ext2` , and `ext3` currently.

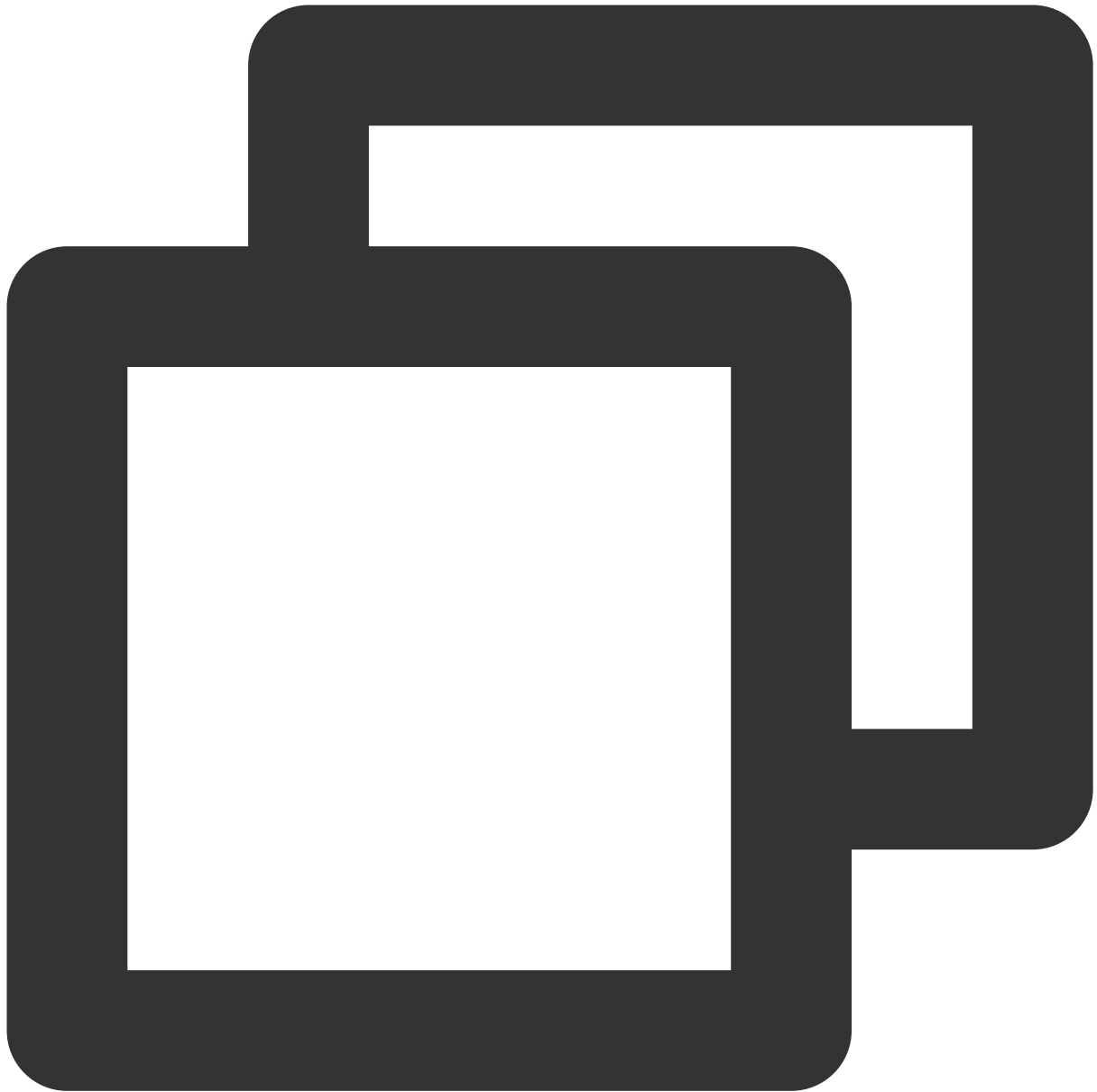
reportTime

This method can be used to report custom speed test as follows:



```
// Suppose the time of `onload` is 1 second  
aegis.reportTime('onload', 1000);
```

If you want to use additional parameters, you can pass them in by using the object type, and they will overwrite the default values of `ext1` , `ext2` , and `ext3` .



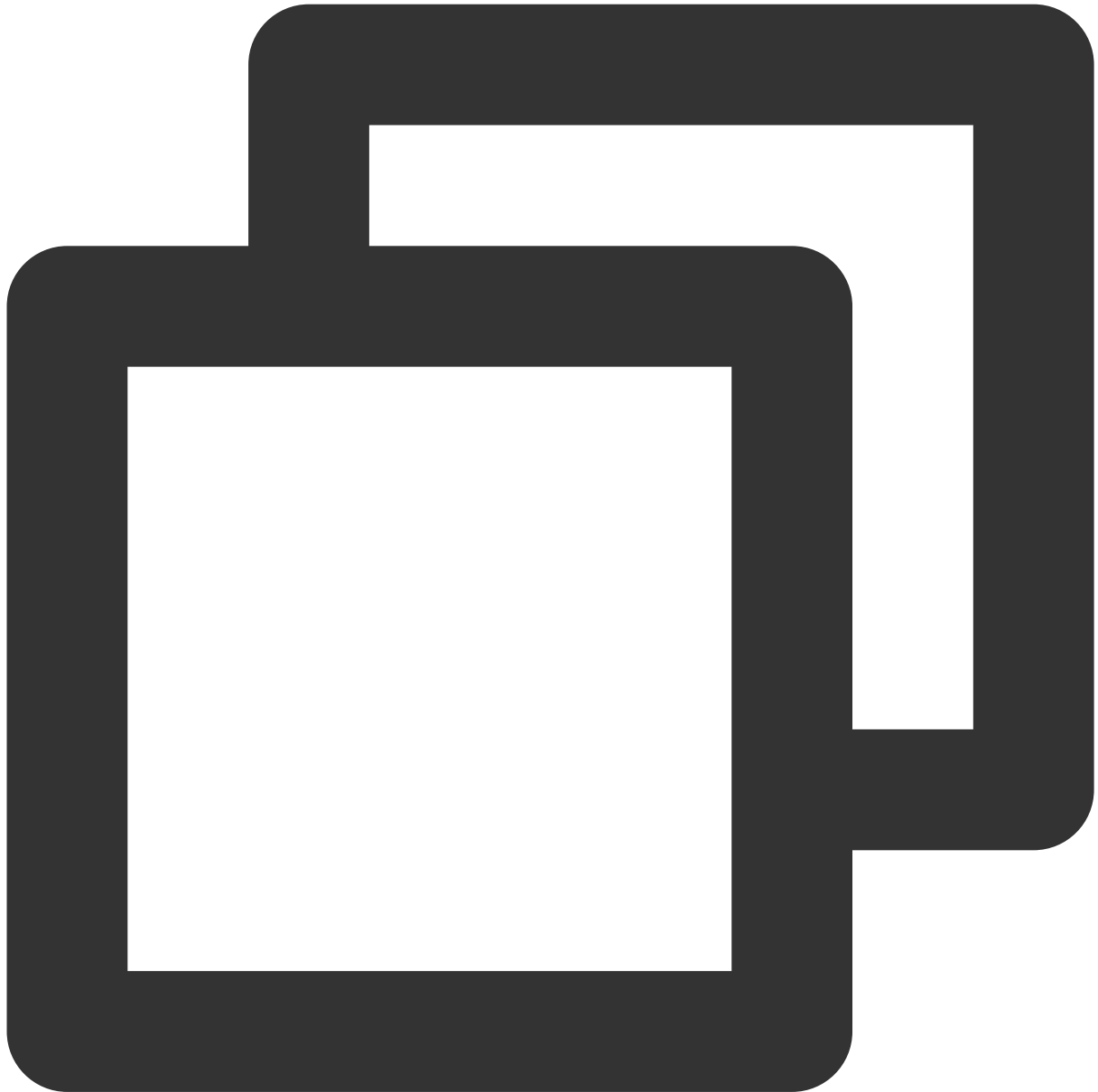
```
aegis.reportTime({
  name: 'onload', // Custom speed test name
  duration: 1000, // Custom speed test duration. Value range: 0-60000
  ext1: 'test1',
  ext2: 'test2',
  ext3: 'test3',
});
```

Note:

`onload` can be renamed.

time and **timeEnd**

These methods can be used to report a custom resource for speed test and are suitable for calculating and reporting a duration between two time points.



```
aegis.time('complexOperation');  
/**  
 * .  
 * .  
 * After complicated operations are performed for a long time  
 * .  
 * .  
 */
```

```
*/  
aegis.timeEnd('complexOperation'); /** At this point, the log has been reported**/
```

Note:

`complexOperation` can be renamed.

In custom speed test, you can report any values, and the server will collect and calculate them. As the server cannot process dirty data, we recommend you restrict the statistics values passed in to prevent the system from being affected by dirty data.

Currently, Aegis can calculate values only between 0 and 60000. If you use a greater value, we recommend you adjust it reasonably.

destroy

This method is used to terminate the instance process, after which no data will be reported, and Aegis will stop collecting user data.



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

Allowlist

Last updated : 2024-01-22 19:25:42

The allowlist feature is used to allow specified users to report more information. To filter out certain users and reduce the number of user API requests, RUM provides this feature and configures its logic.

The allowlist logic is as follows:

1. For users in the allowlist, all API information will be reported, including API request and response.
2. For users in the allowlist, the `info` API can be used to report data.
3. `info` and `infoAll` : in the actual development experience, for users in the allowlist, more logs can be added and reported with the `info` API. `infoAll` reports the data of all users without discrimination; therefore, it may result in a large volume of log data.
4. You can use the `whitelist` API to check whether the current user is in the allowlist, and if so, the response (`aegis.isWhiteList`) of the user will be bound to the Aegis instance for your use.
5. To reduce your workload, the allowlist takes effect at the business system level. You can go to **Application Management > Allowlist Management** to create an allowlist, and it will take effect for all applications in the current business system.

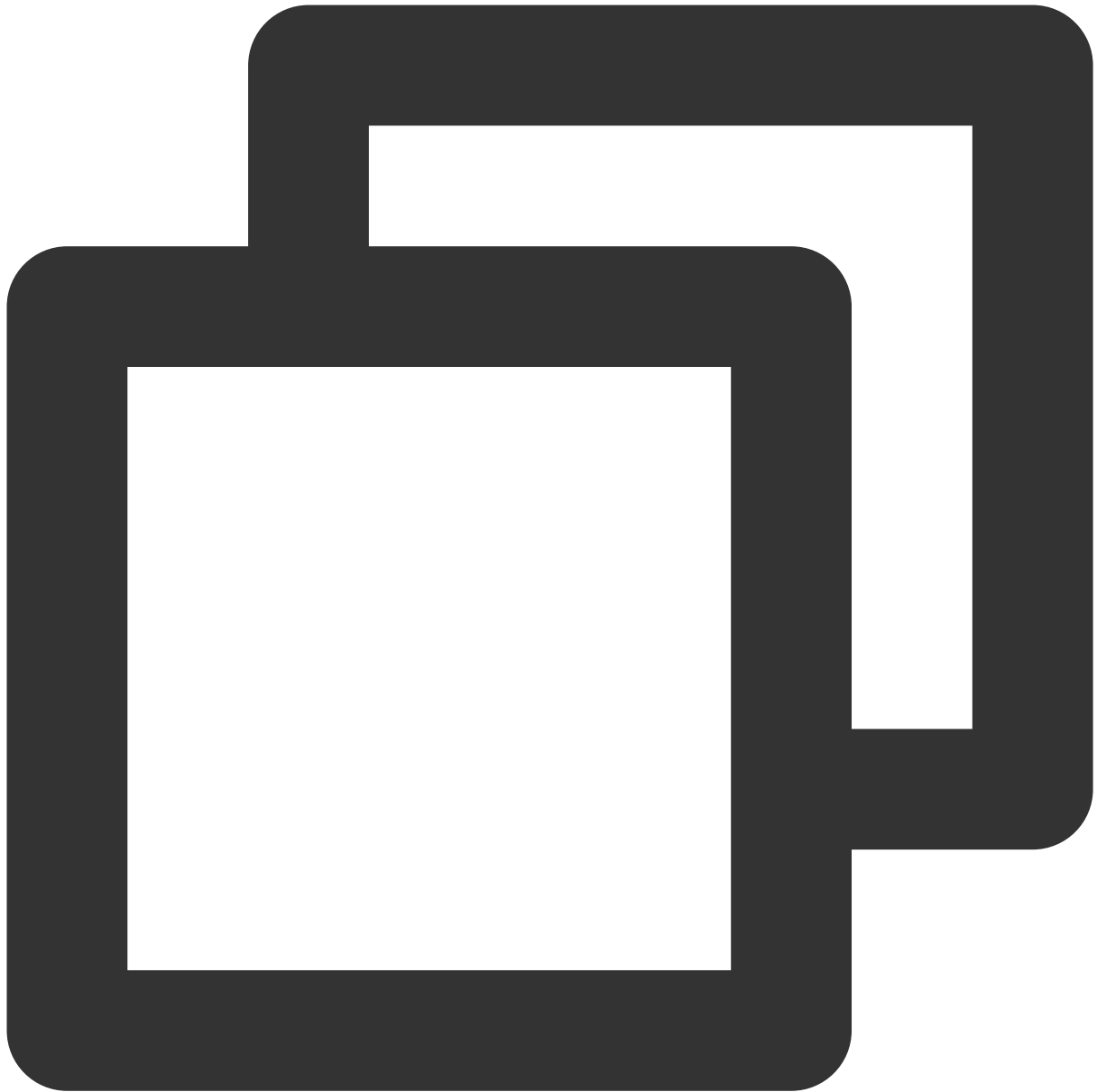
Hook Functions

Last updated : 2024-01-22 19:25:42

You can use a hook function to customize the configuration for resource speed test data reporting, and the system will calculate and collect the function execution duration. Then, you can use custom speed test to analyze the function execution duration in multiple dimensions.

onBeforeRequest

This hook function will be called before all requests are sent, and all request content will be passed in to its parameters. The content to be reported must be returned.



```
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') {
    // Page speed test. Here, you can modify the content of `log`; for example, y
    log.url = changeURLArg(log.url, 'platform', type)
  }
  return log
}
});

// SEND_TYPE {
//   LOG = 'log', // Log
//   SPEED = 'speed', // API and static resource speed test
//   PERFORMANCE = 'performance', // Page speed test
//   OFFLINE = 'offline', // Offline log upload
//   WHITE_LIST = 'whiteList', // Allowlist
//   VITALS = 'vitals', // Web Vitals
//   PV = 'pv', // Custom PV
//   EVENT = 'event', // Custom event
//   CUSTOM = 'custom', // Custom speed test
//   SDK_ERROR = 'sdkError', // SDK error
// }
```

beforeReport

1. This hook will be executed before log reporting (by the `/collect?` API); for example:



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    // Listen on the reported error below
    console.log(log); // {level: "4", msg: "An error occurred."}
    return log;
  }
});

throw new Error('An error occurred.');
```

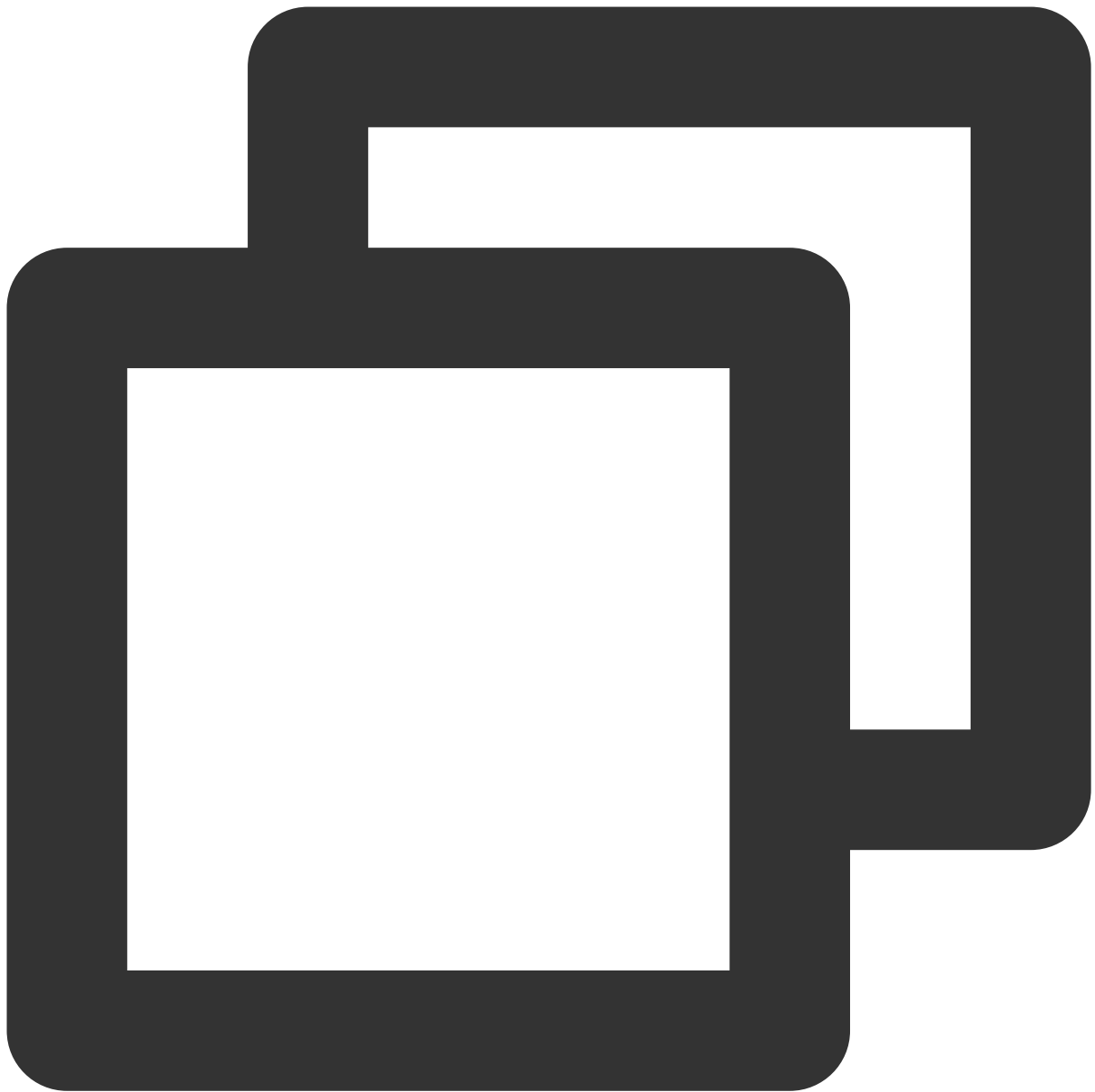
Note:

The above `log` will have the following fields:

1. `level` : log level. For example, `'4'` indicates error log.
2. `msg` : log content.
3. Below are all log levels:

```
{ level: '1', name: 'API request log (allowed log)' }  
{ level: '2', name: 'General log (aegis.info or aegis.infoAll)' }  
{ level: '4', name: 'JavaScript execution error' }  
{ level: '8', name: 'Promise error' }  
{ level: '16', name: 'Ajax request exception' }  
{ level: '32', name: 'JavaScript loading exception' }  
{ level: '64', name: 'Image loading exception' }  
{ level: '128', name: 'CSS loading exception' }  
{ level: '256', name: 'console.error (reserved)' }  
{ level: '512', name: 'Audio/Video resource exception' }  
{ level: '1024', name: 'retcode exception' }  
{ level: '2048', name: 'aegis report' }  
{ level: 4096, name: 'PV' }  
{ level: 8192, name: 'Custom event' }
```

2. If this hook returns `false`, the log won't be reported. This feature can be used to filter out errors that don't need to be reported; for example:



```
const aegis = new Aegis({
  id: 'pGUVFTCYewxxxxx',
  beforeReport(log) {
    if (log.level === '4' && log.msg && log.msg.indexOf('An error that doesn't need
      return false
    }
    return log;
  }
});
throw new Error('An error that doesn't need to be reported'); // This error will no
```

In the above sample code, if the error content contains the `An error that doesn't need to be reported` keywords, it won't be reported to the RUM backend.

onReport

This hook is executed after a log is successfully reported, and its usage is similar to that of `beforeReport`. Their only difference is that all parameters received by this hook are of an already reported log, but the parameters received by `beforeReport` are of a log to be reported.

beforeReportSpeed

1. This hook will be executed before the speed test data is reported (`/speed?`); for example:



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

Note:

The above `msg` will have the following fields:

1. `url` : request address of this resource.
2. `type` : resource type. Valid values: `fetch` : Aegis will report the resource as an API request; `static` : Aegis will report the resource as a static resource.
3. `duration` : resource request duration.
4. `method` : `http method` used when the resource is requested.
5. `status` : status code returned by the server.
6. `payload` : complete resource request information provided to you (this field won't be reported to the Aegis backend and can be manipulated by you)

The complete data structure is as follows:

`payload.type` : resource request type, which is used to distinguish between the original request types. Valid values: `'fetch'` and `'xhr'` .

`payload.sourceURL` : complete URL request connection.

`payload.status` : request status code.

`payload.headers` : all request headers, whose values are all strings.

`payload.data` : complete request resource, which you can customize. If the request type is `fetch` , it is the `response` object; if the request type is `xhr` , it is the `XMLHttpRequest` object.

In the above sample code, every time after Aegis collects the loading details of a resource, it will use such details (i.e., `msg` returned above) as parameters to call the `beforeReportSpeed` hook.

2. If you have configured this hook, the final content reported by Aegis is subject to the execution result of the hook; for example:

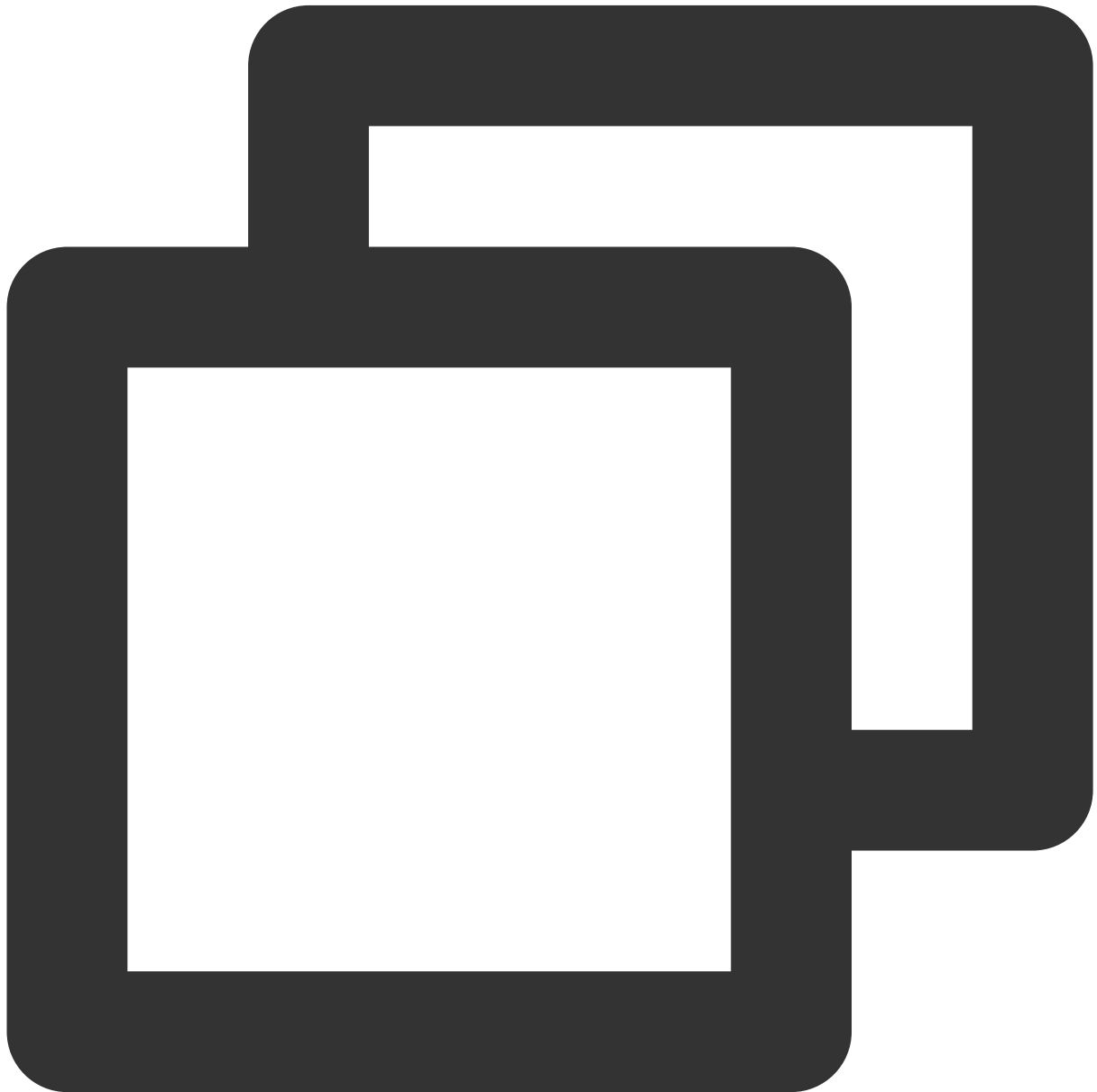


```
const aegis = new Aegis({
  id: 'pGUVFTCYewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    msg.type = 'static';
    return msg;
  }
});
```

In the above sample code, all `msg.type` parameters are set to `static`, indicating that all resources, even API requests, will be reported as static resources.³ You can use this hook to correct the Aegis type and judge incorrect requests.

Samples

You have an API `https://example.com/api` whose response header `Content-Type` is `text/html`. In normal case, RUM will report this API as a static resource; however, it must be reported as an API request in your business. Then, you can configure Aegis with following hook for correction:

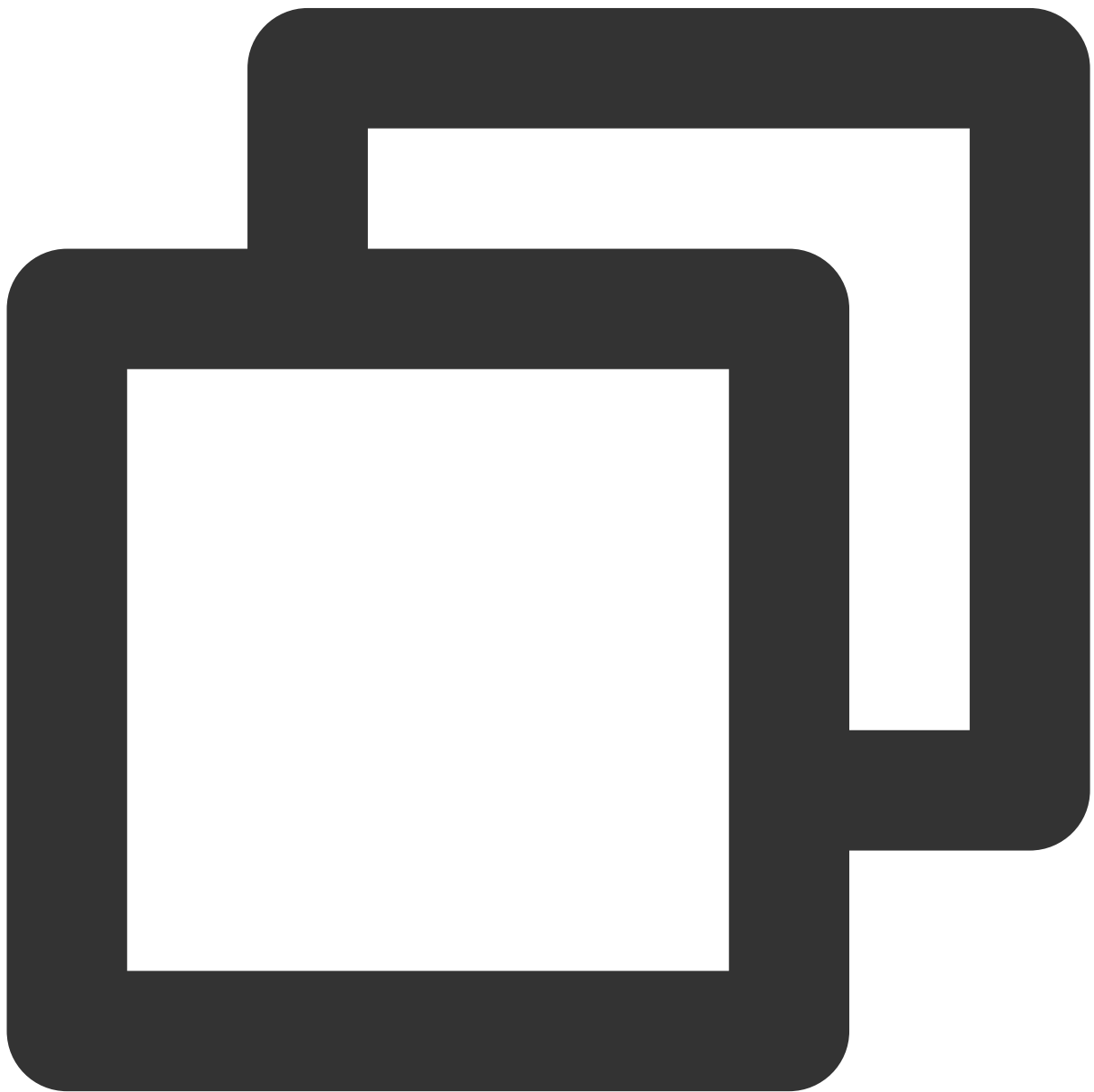


```
const aegis = new Aegis({
```



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

1. You can also block the speed test data reporting of a resource as follows:



```
const aegis = new Aegis({
  id: 'pGUVFTCYewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    // Speed test logs for resources whose address contains `https://example.com/ap
    if (msg.url.indexOf('https://example.com/api') !== -1) {
      // If `false` is returned, the reporting of the speed test log will be blocke
      return false
    }
  }
});
```

beforeRequest

This hook will be executed before log reporting; for example:

Note:

The SDK version must be 1.24.44 or above.



```
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;
  }
});
```

Here, `msg` will have the following fields:

1. `logType`: log type. Valid values:

`custom`: custom speed test.

`event`: custom event.

`log`: log.

`performance`: page speed test.

`pv`: PV.

`speed`: API and static resource speed test.

`vitals`: Web Vitals.

2. `logs`: reported log content:

If `logType` is `'custom'`, `logs` will be in the format of `{name: "white screen time", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}`.

If `logType` is `'event'`, `logs` will be in the format of `{name: "ios", ext1: "", ext2: "", ext3: ""}`.

If `logType` is `'performance'`, `logs` will be in the format of `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 2660, ssl: 4, tcp: 4, ttfb: 5}`.

If `logType` is `'speed'`, `logs` will be in the format of `{connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "static", url: "https://xxxxxx", urlQuery: "max_age=1296000"}`.

If `logType` is `'vitals'`, `logs` will be in the format of `{delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "LCP", value: 1100}`.

If `logType` is `'log'`, `logs` will be in the format of `{msg: "log details", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}`.

Note:

Valid values of `level`:

```
{ level: '1', name: 'API request log (allowed log)' }
{ level: '2', name: 'General log (aegis.info or aegis.infoAll)' }
{ level: '4', name: 'JavaScript execution error' }
{ level: '8', name: 'Promise error' }
{ level: '16', name: 'Ajax request exception' }
{ level: '32', name: 'JavaScript loading exception' }
{ level: '64', name: 'Image loading exception' }
{ level: '128', name: 'CSS loading exception' }
{ level: '256', name: 'console.error (reserved)' }
{ level: '512', name: 'Audio/Video resource exception' }
```

```
{ level: '1024', name: 'retcode exception' }
```

```
{ level: '2048', name: 'aegis report' }
```

```
{ level: 4096, name: 'PV' }
```

```
{ level: 8192, name: 'Custom event' }
```

If this hook returns `false`, the log won't be reported. This feature can be used to filter out errors and logs that don't need to be reported.

afterRequest

This hook will be executed after speed test data reporting; for example:

Note:

The SDK version must be 1.24.44 or above.



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

Here, `msg` will have the following fields:

1. `isErr`: whether the request reporting API has an error.

2. result: response of the reporting API.
3. logs: reported log content.
4. logType: log type, which is the same as `logType` in `beforeRequest` .

Error Monitoring

Last updated : 2024-01-22 19:25:42

The Aegis instance of RUM automatically monitors exceptions such as JavaScript and Promise execution errors and Ajax (fetch) request exceptions. This document describes the monitoring logic of each error type and how to handle them.

Note:

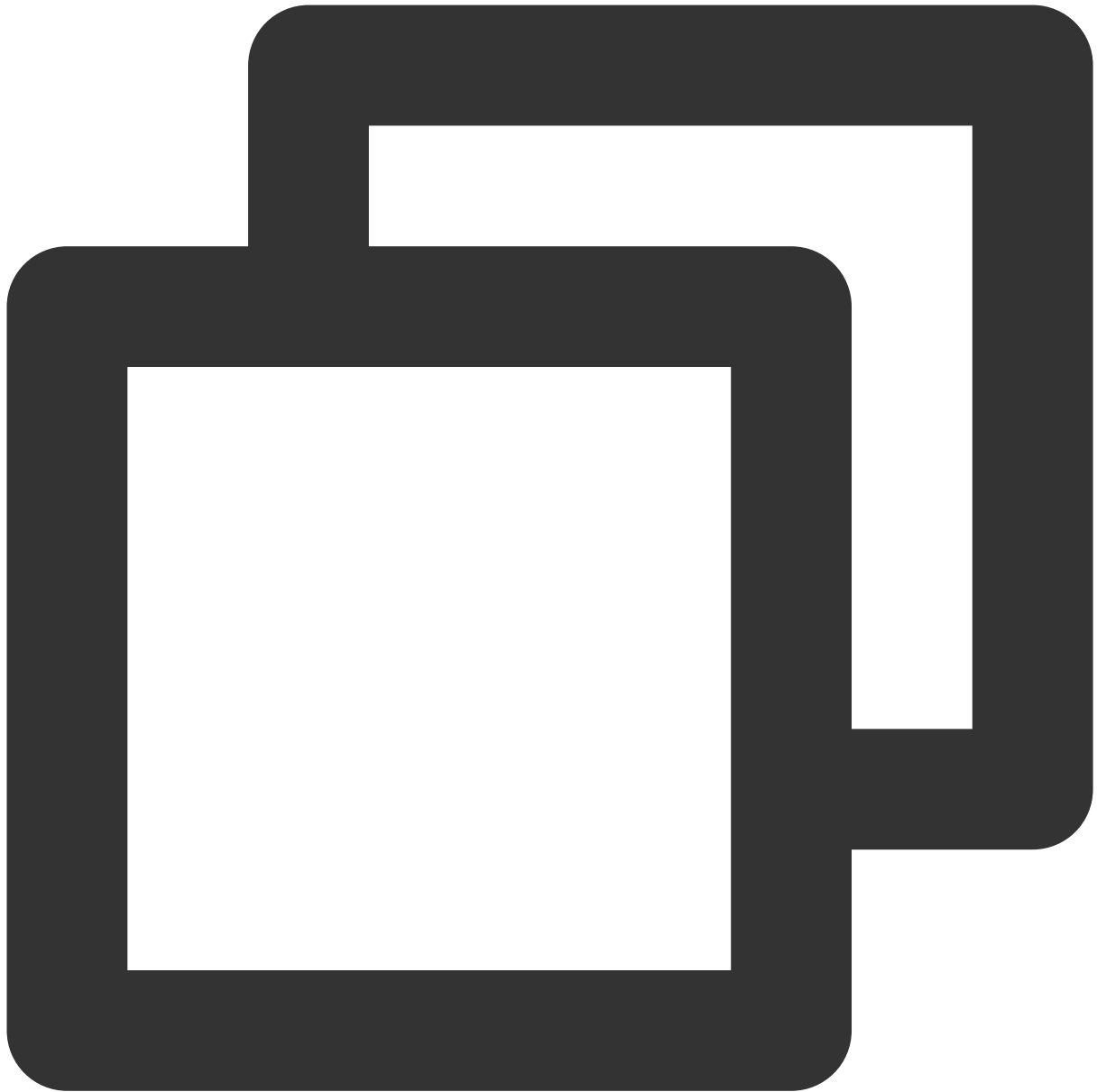
The Aegis instance monitors such exceptions. If you only import the Aegis SDK but don't instantiate it, it won't report any data.

JavaScript Execution Error

Aegis listens on the `onerror` event in the `window` object to capture project errors. Then, it parses errors and analyze the heaps and stacks to automatically report the error information to the backend service. In this case, the report level is `error`; therefore, when the number of automatically reported errors reaches the threshold, Aegis will automatically trigger an alarm to help you find the exceptions promptly. As the report level is `error`, automatic reporting will also affect the project score.

If a cross-origin JavaScript script is imported on the page, you need to add the `crossorigin` attribute to the `script` tag; otherwise, Aegis cannot get the error details.

If the user uses the Vue framework, import the following code to get and actively report errors:



```
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 Execution Error

RUM listens on the `unhandledrejection` event to capture Promise errors that are not caught through `catch`. To ensure page stability, we recommend you `catch` all Promise errors.

Ajax (`fetch`) Request Exception

Aegis rewrites the `XMLHttpRequest` object to listen on each API request and will consider the following cases an exception:

`http status` is greater than or equal to 400.

A request times out or is aborted, cross-origin, or canceled.

`http status` is still 0 when a request ends, which usually occurs when the request fails.

Note:

When an error occurs, the Aegis SDK doesn't actively collect API request and response parameters. If you want such API information to be reported, you can use the `apiDetail` API parameter to enable reporting.



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

retcode Exception

Aegis rewrites the `XMLHttpRequest` object to get the API response. It also tries to get the `retcode` of the request in the response. If the `retcode` doesn't meet the expectation, the request will be considered exceptional and will be reported.

Note:

You can get `retcode` and judge which `retcode` values are normal as instructed in [Configuration Guide](#).

Resource Loading Failure

If a request sent by a page element fails, it will be captured by the `window.onerror` event. Aegis uses this feature to listen on the loading failures of following resources:

CSS and font requested by the `<link>` tag.

Scripts requested by the `<script>` tag.

Multimedia resources requested by the `<audio>` and `<video>` tags.

Performance Monitoring

Last updated : 2024-01-22 19:25:42

This document describes the statistical methods and configurations passed in for page, API, and resource speed tests.

Page Speed Test

Note:

RUM enables the page speed test feature for you by default.

After you successfully install and initialize the SDK, the Aegis instance will report the following metrics by default:

1. **DNS query:** domainLookupEnd - domainLookupStart;
2. **TCP connection:** connectEnd - connectStart
3. **SSL connection establishment:** requestStart - secureConnectionStart
4. **Request response:** responseStart - requestStart
5. **Content transfer:** responseEnd - responseStart
6. **DOM parsing:** domInteractive - domLoading
7. **Resource loading:** loadEventStart - domInteractive
8. **FMP:** RUM listens on the first screen DOM changes within 3 seconds after a page is opened and takes the time when the number of DOM changes reaches the highest as the time when the first screen framework rendering is completed. (setTimeout is used to start first screen element collection 3 seconds after SDK initialization. As JavaScript is executed in a single-thread environment, the collection time point may be more than 3 seconds after SDK initialization.)
9. **Complete page loading duration:** sum of 1–7 (DNS query, TCP connection, SSL connection establishment, request response, content transfer, DOM parsing, and resource loading)

Note:

For more information on how to calculate page open performance metrics 1–7, see [PerformanceTiming](#). You can print

`aegis.firstScreenInfo` to view the corresponding DOM element of `firstScreenTime`. If a DOM element cannot represent the first screen, you can add the `<div AEGIS-FIRST-SCREEN-TIMING></div>` attribute to recognize it as the key first screen element, so that the SDK will consider that the first screen is loaded if it is displayed on the first screen. You can also add the `<div AEGIS-IGNORE-FIRST-SCREEN-TIMING></div>` attribute to add the element to the blocklist.

RUM provides the page loading waterfall plot based on the above metrics.

Note:

In server scenarios, the `firstScreenTime` in the waterfall plot may be longer than the DOM parsing duration, which is caused by incompatibility of mobile devices. As some devices cannot get the durations of DNS query, TCP

connection, and SSL connection establishment, the three metrics will have a smaller average value after being aggregated, causing metrics except `firstScreenTime` to shift to the left.

API Speed Test

Note:

How to enable: pass in `reportApiSpeed: true` during initialization.

Aegis tests the API speed by hijacking `XHR` and `fetch`.

Resource Speed Test

Note:

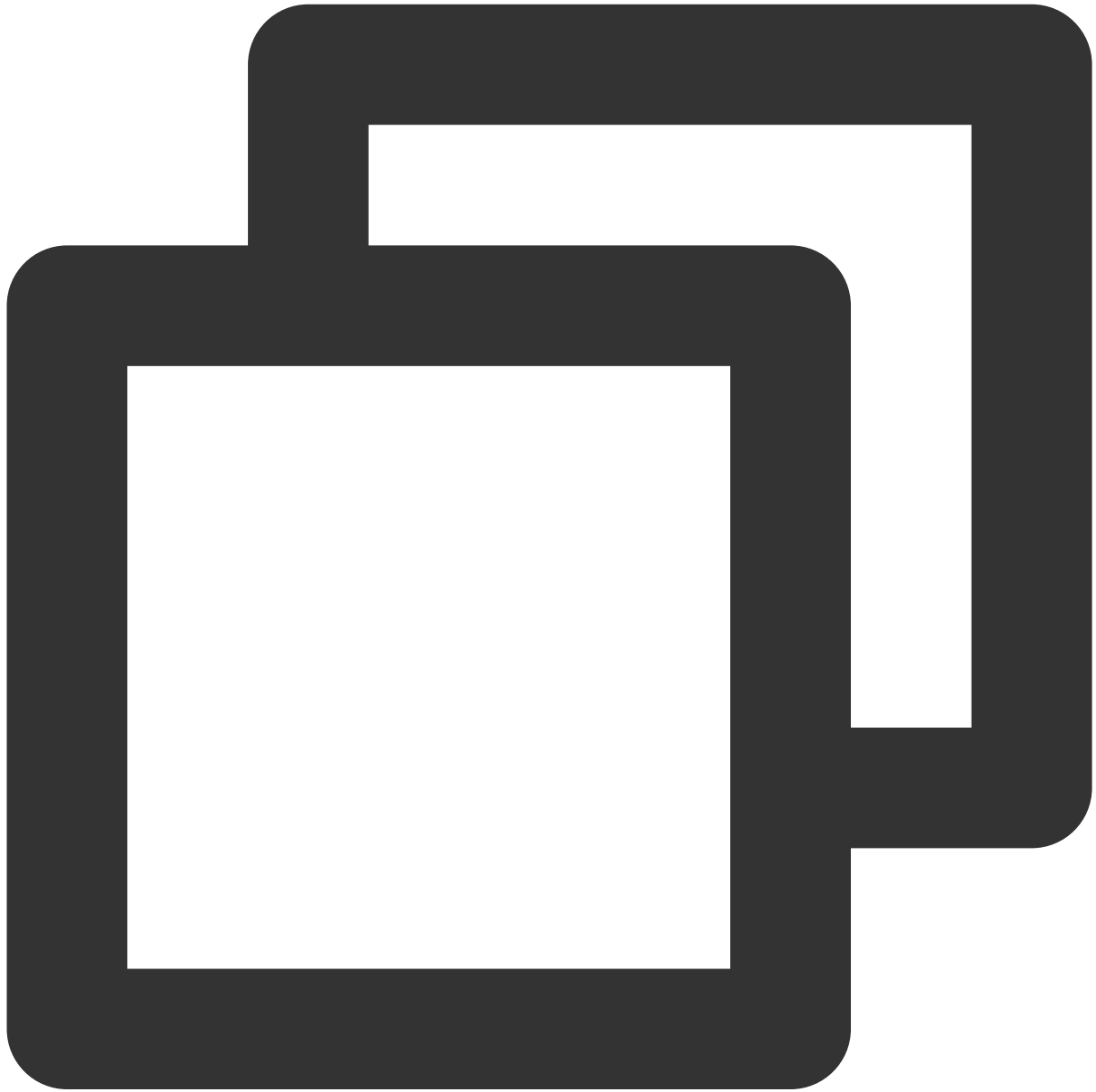
How to enable: pass in `reportAssetSpeed: true` during initialization.

Aegis uses `PerformanceResourceTiming` provided by the browser to test the resource speed.

Environment Control

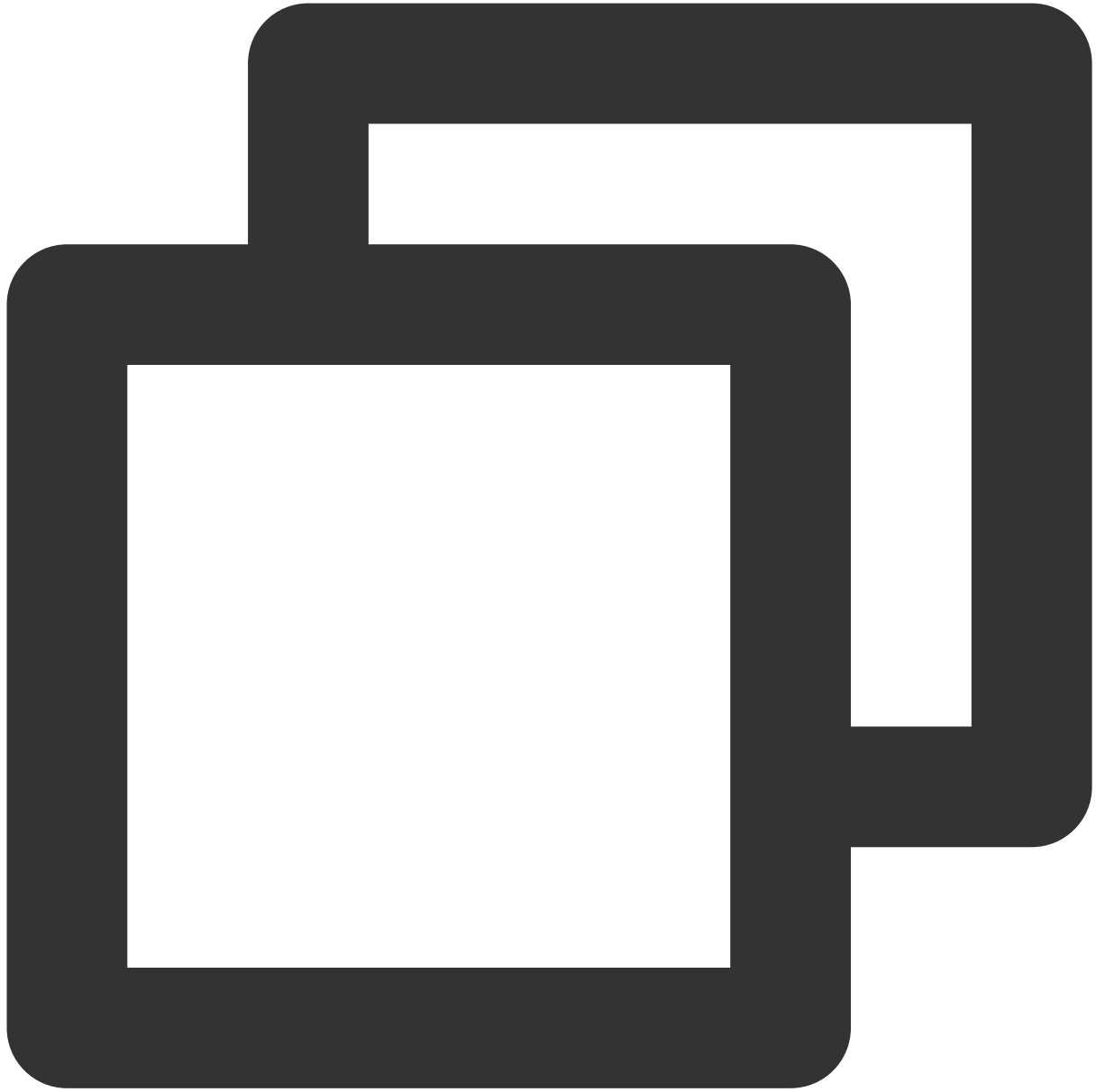
Last updated : 2024-01-22 19:25:42

Aegis will report the data environment as `production` , and you can modify it through the `env` parameter.



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

Valid values of `Aegis.environment` :



```
export enum Environment {  
  production = 'production', // Production environment  
  gray = 'gray', // Beta test environment  
  pre = 'pre', // Prerelease environment  
  daily = 'daily', // Daily release environment  
  local = 'local', // Local environment  
  test = 'test', // Test environment  
  others = 'others' // Other environments  
}
```


After you modify the `env` parameter, the data reported by Aegis will carry it to help you distinguish between the data from different environments. However, only data in the `production` environment can participate in the calculation of the project score.

Configuration Guide

Last updated : 2024-01-22 19:25:42

Configuration Description

The configuration items in the configuration file are as detailed below:

Configuration Item	Description
id	It is a required string and is empty by default. It is the project ID assigned by RUM.
uin	It is a recommended string and is the UIN field in the cookie by default. It is the unique ID of the current user. When a log is reported, it will be used to check whether the user is in the allowlist. Its value can contain only letters, digits, and <code>@=._-</code> , and its regular expression is <code>/^[@=.0-9a-zA-Z_-]{1,60}\$/</code> .
reportApiSpeed	It is an optional boolean value or <code>object</code> and is <code>false</code> by default. It specifies whether to enable API speed test.
reportAssetSpeed	It is an optional boolean value and is <code>false</code> by default. It specifies whether to enable static resource speed test.
pagePerformance	It is an optional boolean value or <code>object</code> and is <code>true</code> by default. It specifies whether to enable page speed test.
webVitals	It is an optional boolean value and is <code>true</code> by default. It specifies whether to enable Web Vitals speed test.
onError	It is an optional boolean value and is <code>true</code> by default. It specifies whether to enable error listening in the current instance to get error logs.
aid	It is an optional boolean value and is <code>true</code> by default. It specifies whether to generate <code>aid</code> in the current instance.
random	It is an optional number and is 1 by default. It is the sample rate. Value range: 0-1.
spa	It is an optional boolean value and is <code>false</code> by default. It specifies whether the current page is an SPA page. If the value is <code>true</code> , <code>hashchange</code> and the history API will be listened on to report the PV during page redirect.
version	It is an optional string and is the SDK version number by default.

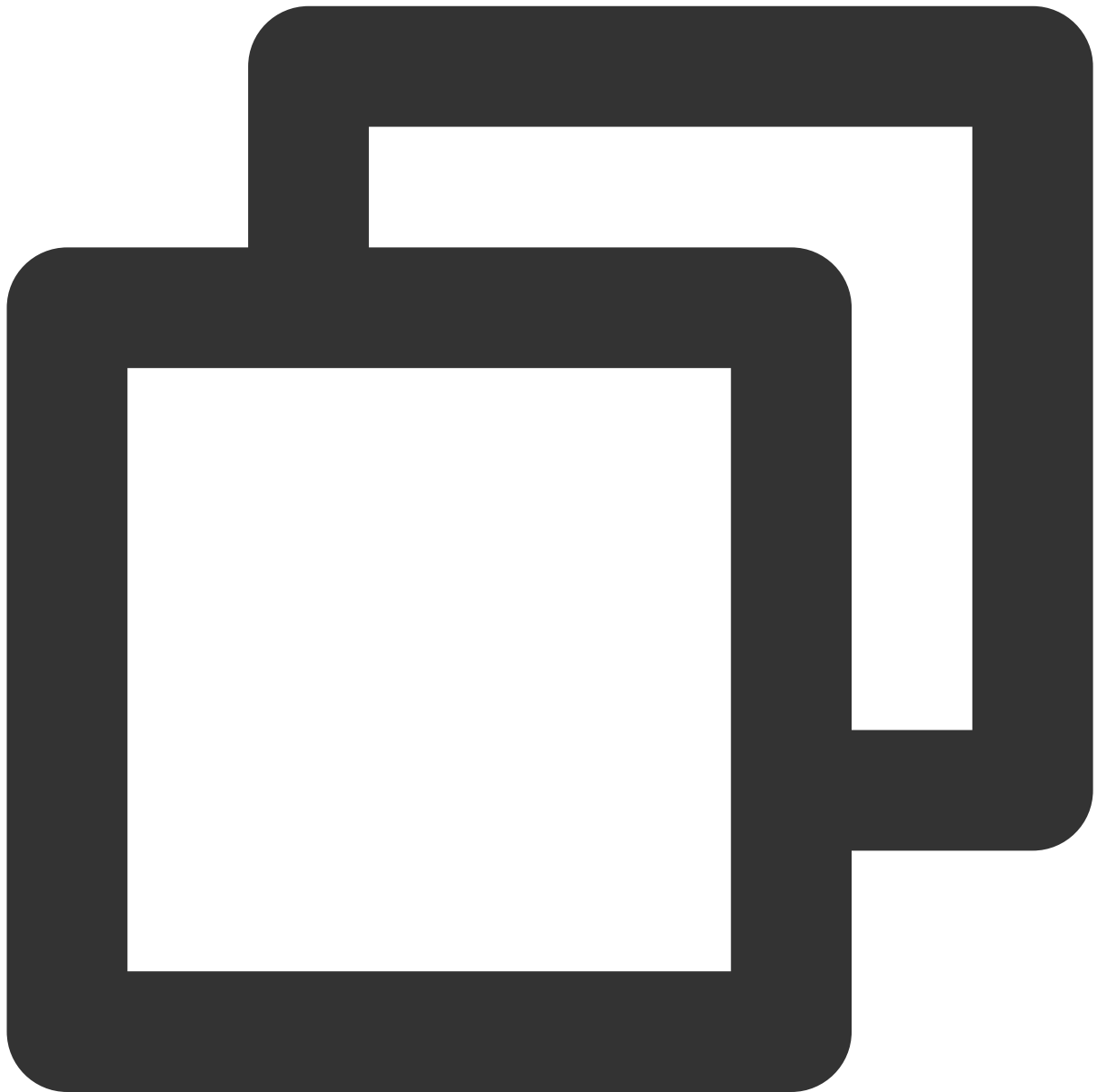
	It is the version of the currently reported content. If the page uses PWA or an offline package, it can be used to judge the version of the code where the currently reported content is from. Its value can contain up to 60 letters, digits, and <code>=._-</code> , and its regular expression is <code>/^[0-9a-zA-Z.,:_-]{1,60}\$/</code> .
delay	It is an optional number and is 1000 ms by default. It is the time period for reducing reporting traffic, within which multiple reports will be merged into one reporting request.
repeat	It is an optional number and is 5 by default. It is the number of repeated reports. After it is exceeded, the same error will not be reported again.
env	It is an optional enum and is <code>Aegis.environment.production</code> by default. It is the current environment where the project runs.
offlineLog	It is an optional boolean value and is <code>false</code> by default. It specifies whether to use offline log.
offlineLogExp	It is an optional number and is 3 by default. It is the offline log validity period.
api	It is an optional object and is <code>{}</code> by default. Fields: apiDetail: it specifies whether to report the API request parameters and returned value if an API fails. It is an optional boolean value and is <code>false</code> by default. retCodeHandler: <code>Function(data: String, url: String, xhr: Object): {isErr: boolean, code: string}</code> : it is the hook function for return code reporting and will pass in the API response data, request <code>url</code> , and <code>xhr</code> object. <code>resourceTypeHandler: Function</code> : it is the request resource type correction hook function and will pass in the API <code>url</code> . Its returned value is <code>static</code> or <code>fetch</code> . For more information, see api.retCodeHandler . reportRequest: it is a boolean value and is <code>false</code> by default. After it is enabled, <code>aegis.info</code> will report the full data with no need to configure the allowlist, and information of all APIs will be reported (you need to enable <code>reportApiSpeed</code> in the reporting API).
speedSample	It is an optional boolean value and is <code>true</code> by default. It specifies whether to sample the speed test logs (that is, each URL reports only one speed test log).
hostUrl	It is optional and is <code>//aegis.qq.com</code> by default. It is the host address that affects all reported data. After the following URL addresses are set, they will overwrite the original reporting addresses.
url	It is an optional string and is <code>//aegis.qq.com/collect</code> by default. It is the log reporting address.

	You can set it to an empty string to disable log reporting.
pvUrl	It is an optional string and is <code>//aegis.qq.com/collect/pv</code> by default. It is the PV reporting address. You can set it to an empty string to disable PV reporting.
whiteListUrl	It is an optional string and is <code>//aegis.qq.com/collect/whitelist</code> by default. It is the allowlist confirming API. You can set it to an empty string to disable allowlist API request.
offlineUrl	It is an optional string and is <code>//aegis.qq.com/collect/offline</code> by default. It is the offline log reporting address. You can set it to an empty string to disable offline log reporting.
eventUrl	It is an optional string and is <code>//aegis.qq.com/collect/events</code> by default. It is the custom event reporting address. You can set it to an empty string to disable custom event reporting.
speedUrl	It is an optional string and is <code>//aegis.qq.com/speed</code> by default. It is the speed test log reporting address.\\You can set it to an empty string to disable speed test data reporting.
customTimeUrl	It is an optional string and is <code>//aegis.qq.com/speed/custom</code> by default. It is the custom speed test reporting address. You can set it to an empty string to disable custom speed test reporting.
performanceUrl	It is an optional string and is <code>//aegis.qq.com/speed/performance</code> by default. It is the page performance reporting address. You can set it to an empty string to disable page performance reporting.
webVitalsUrl	It is an optional string and is <code>//aegis.qq.com/speed/webvitals</code> by default. It is the Web Vitals reporting address. You can set it to an empty string to disable Web Vitals reporting.
dbConfig	It is an optional object.
ext1	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.
ext2	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.
ext3	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.

Sample Code

api.retCodeHandler

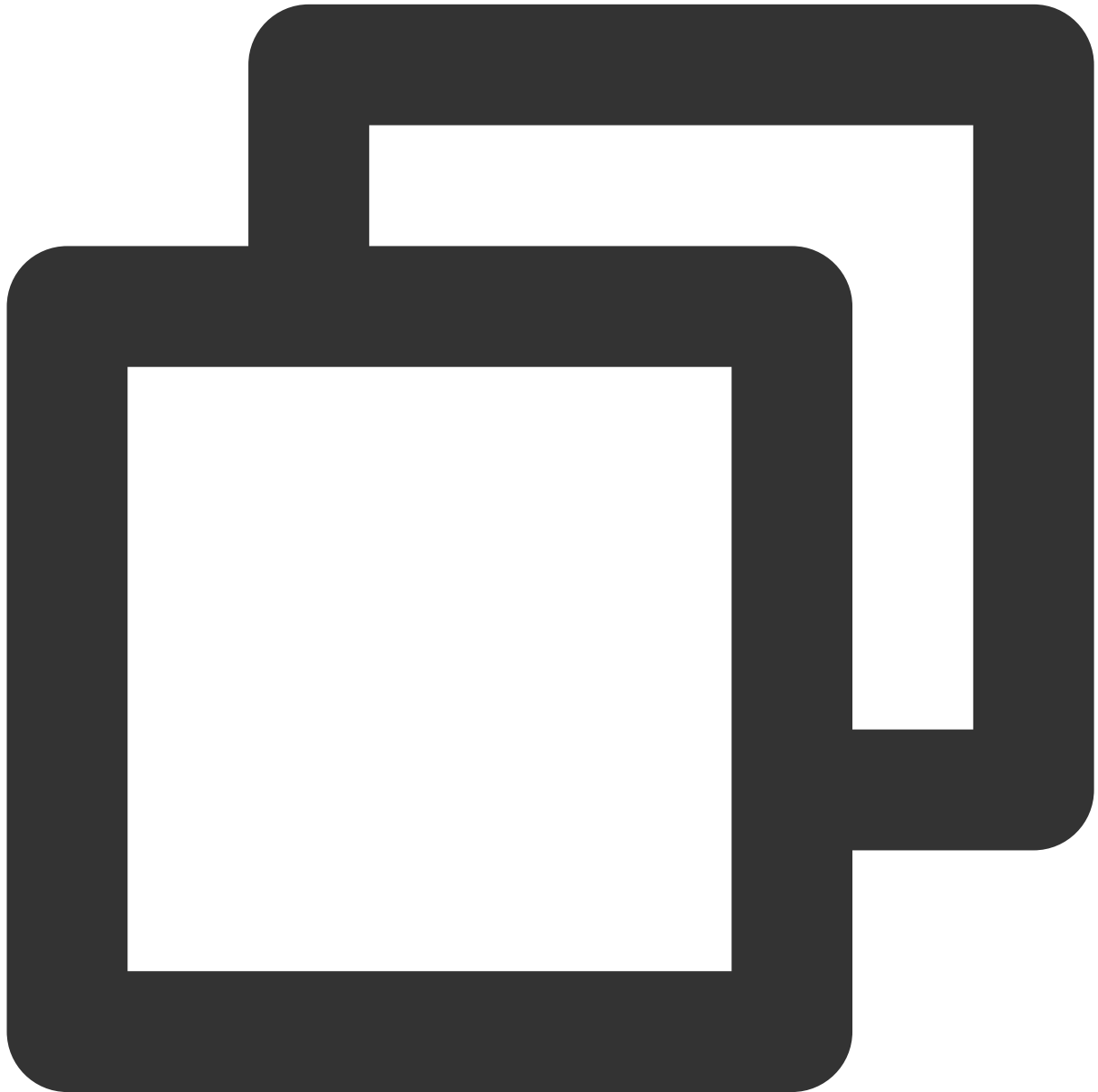
If the backend returns the following data:



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

```
        // xxx
    }
}
```

If your business requires that if the `code` is not 200 or `retCode` is not 0, the request is incorrect. To meet this requirement, you can simply configure as follows:

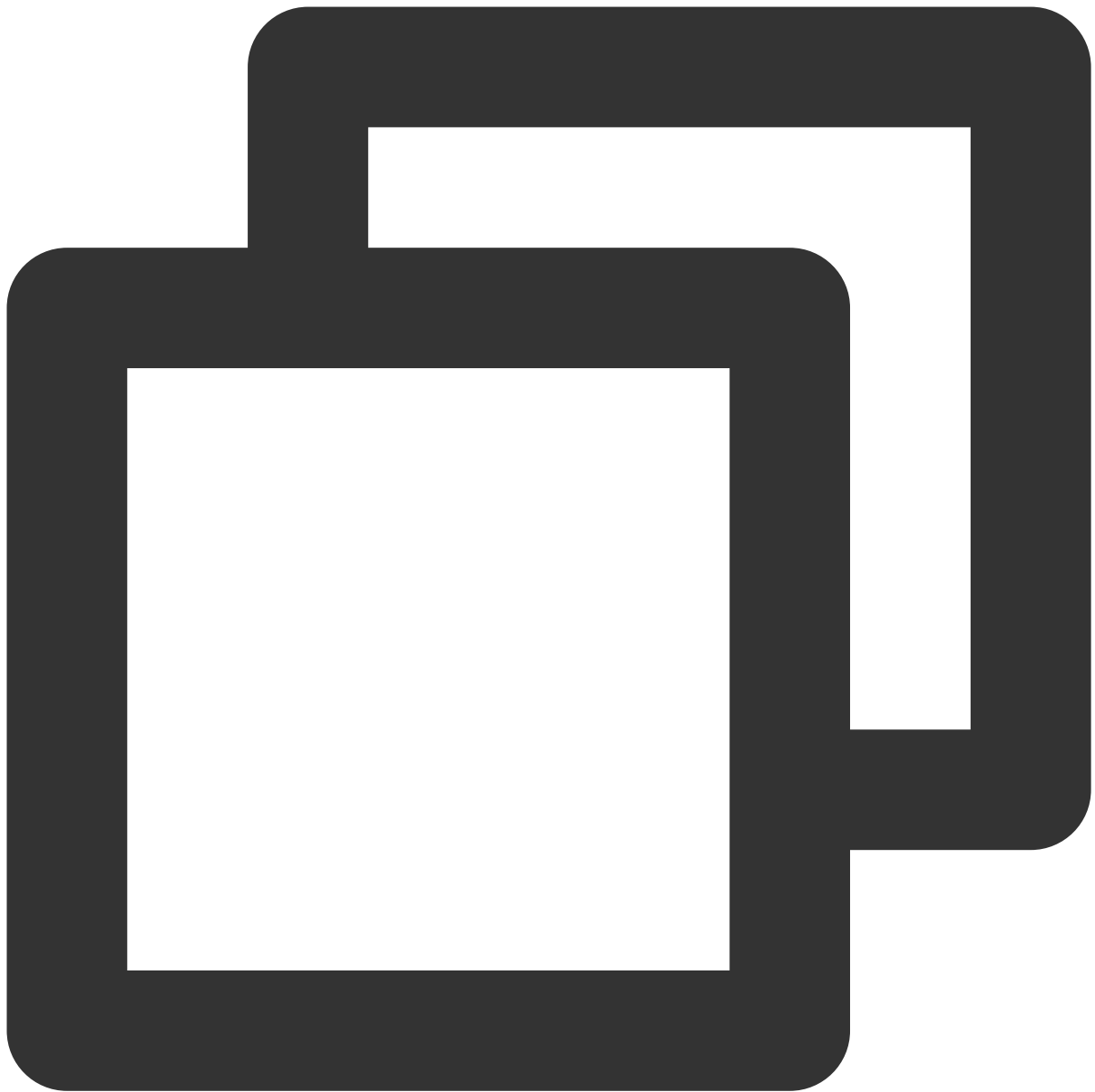


```
new Aegis({
  // xxx
  reportApiSpeed: true, // You need to enable two speed test APIs; otherwise, no
```

```
reportAssetSpeed: true,
api: {
  retCodeHandler(data, url, xhr) {
    // `data` is a string. If you want to get an object, you need to manual
    // `url` is the request URL
    // The complete backend `xhr` response can be obtained through `xhr.res
    try {
      data = JSON.parse(data)
    } catch(e) {}
    return {
      isErr: data.body.code !== 200 || data.body.retCode !== 0,
      code: data.body.code
    }
  }
}
})
```

api.resourceTypeHandler

If the API is `http://example.com/test-api` and the returned `Content-Type` is `text/html`, Aegis will consider that the content returned by the API is a static resource. In this case, you can correct the judgment as follows:

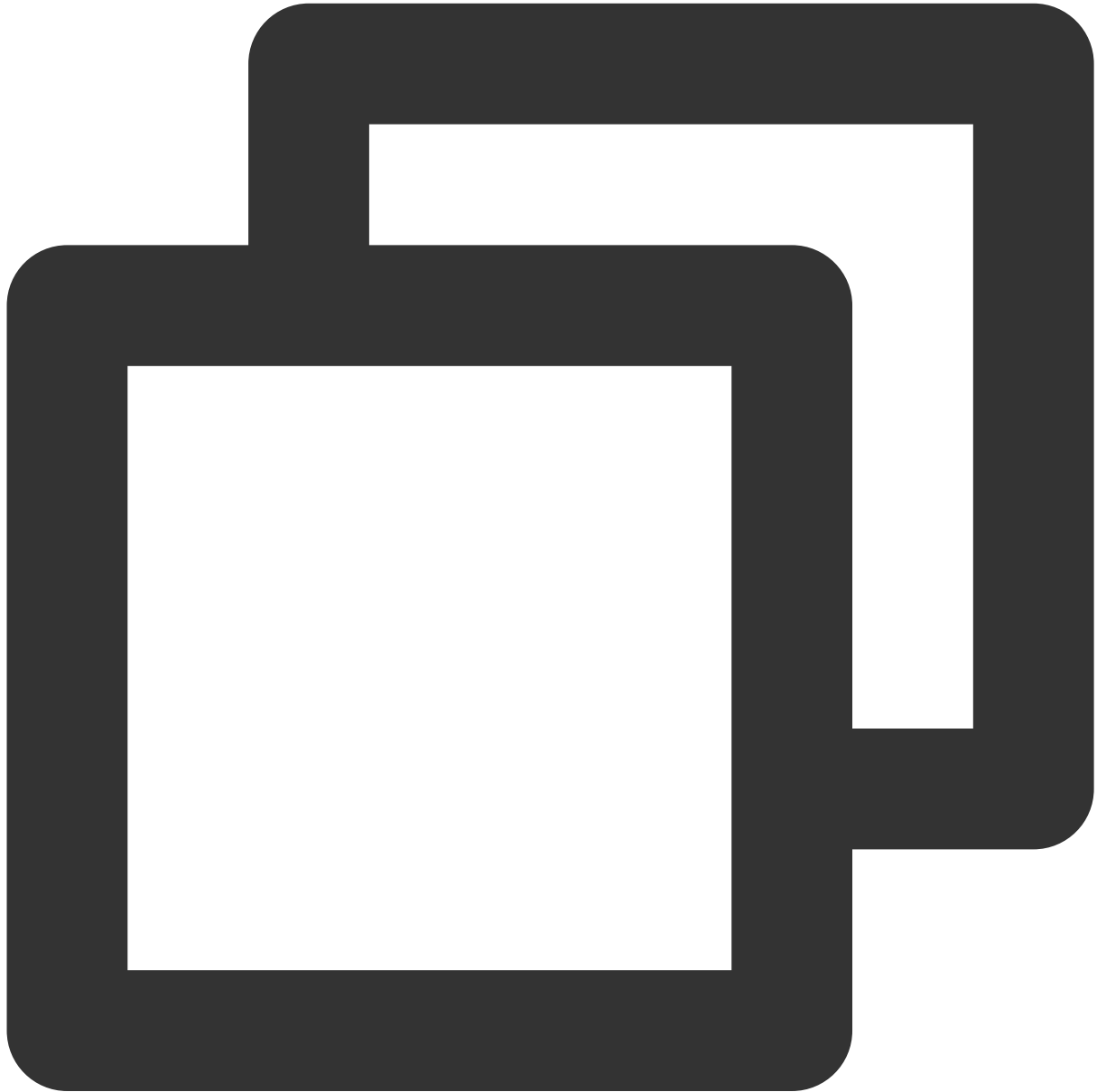


```
new Aegis({
  reportApiSpeed: true, // You need to enable two speed test APIs; otherwise, no
  reportAssetSpeed: true,
  api: {
    resourceTypeHandler(url) {
      if (url?.indexOf('http://example.com/test-api') !== -1) {
        return 'fetch';
      }
    }
  }
})
```


reportApiSpeed.urlHandler

If your page has RESTful APIs such as: `www.example.com/user/1000` `www.example.com/user/1001`

You need to aggregate the following APIs when reporting the speed test data:



```
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

If your page has RESTful APIs such as: `www.example.com/user/1000` `www.example.com/user/1001`

You need to aggregate the page addresses when reporting the page speed test data:



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


Mini Program Use Cases

Installation and Initialization

Last updated : 2024-01-22 19:25:42

Notes

You can only use npm to install the SDK for mini program.

The SDK supports WeChat Mini Program and QQ Mini Program.

To connect to a mini program in the production environment, you need to add the reporting domain as a secure domain.

Note:

`aegis-sdk` uses `https://aegis.qq.com` as the domain name to be reported by default. You can use the `hostUrl` parameter to configure the domain name to be reported.

For regions in the Chinese mainland, you can choose `https://tamaegis.com` as the domain name to be reported.

For Singapore region, you can choose `https://rumt-sg.com` as the domain name to be reported.

Installing SDK

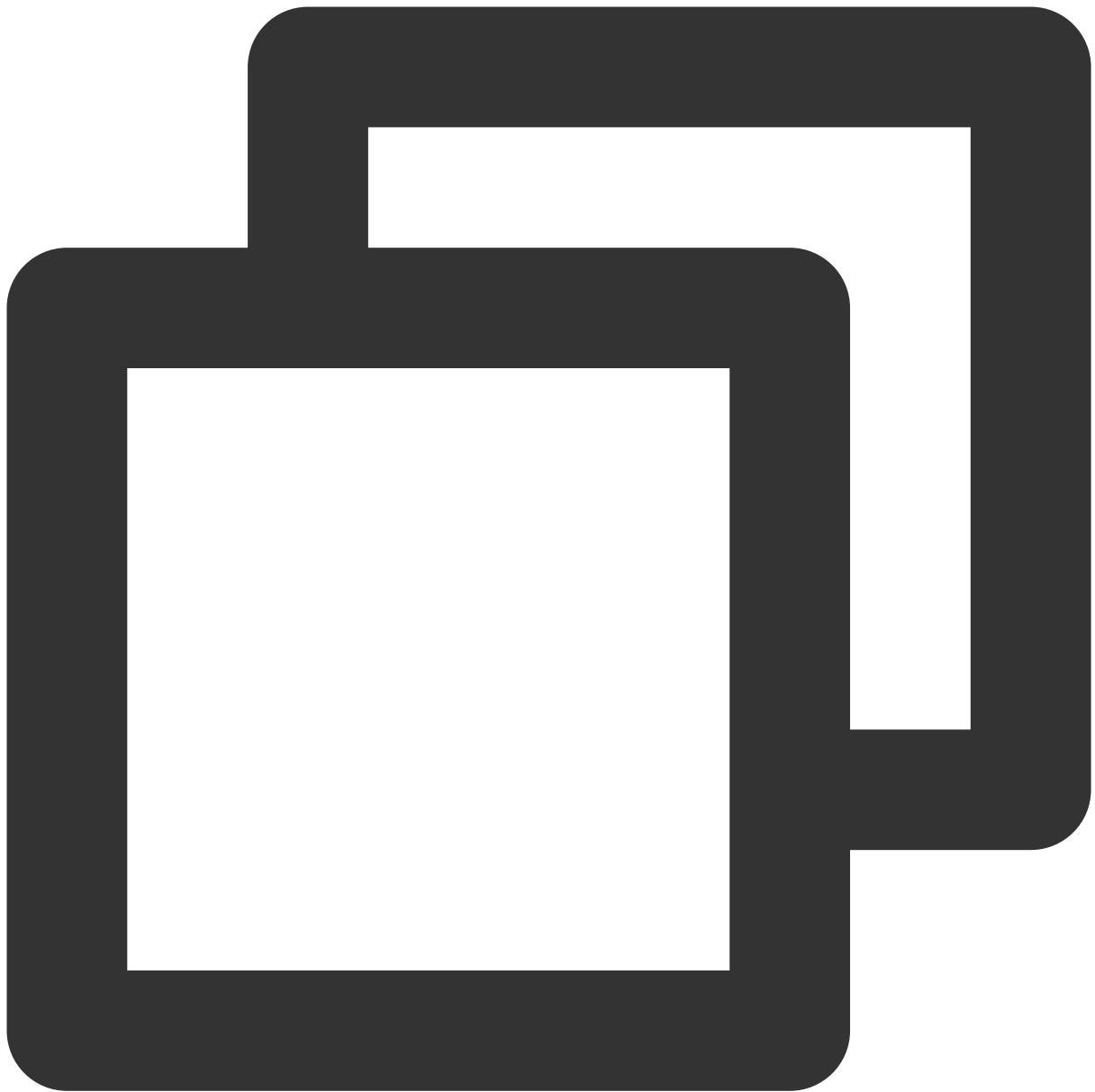
Run the following command to install `aegis-mp-sdk` in your npm repository:



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

Initialization

Create an Aegis instance and pass in the corresponding configuration in the following steps to initialize the SDK:



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

const aegis = new Aegis({
  id: "pGUVFTCZyewxxxxx", // Project key
  uin: 'xxx', // User UIN (optional)
  reportApiSpeed: true, // API speed test
  spa: true, // Report the PV during page switch
});
```

Note:

To avoid missing any data, you should initialize the SDK as early as possible. If you use `miniprogram-api-promise` to encapsulate the `wx.request` request in your mini program project, you should note that Aegis rewrites `wx.request` for API monitoring. Therefore, you must initialize Aegis before importing `miniprogram-api-promise`; otherwise, Aegis may fail to collect the complete API information.

After you install and initialize the SDK, you can use the following RUM features:

1. Error monitoring: JavaScript execution error monitoring.
2. Speed test: API speed test.
3. Statistics collection and analysis: you can analyze data in multiple dimensions on the [Data Overview](#) page in the RUM console.

Log Reporting

Last updated : 2024-01-22 19:25:42

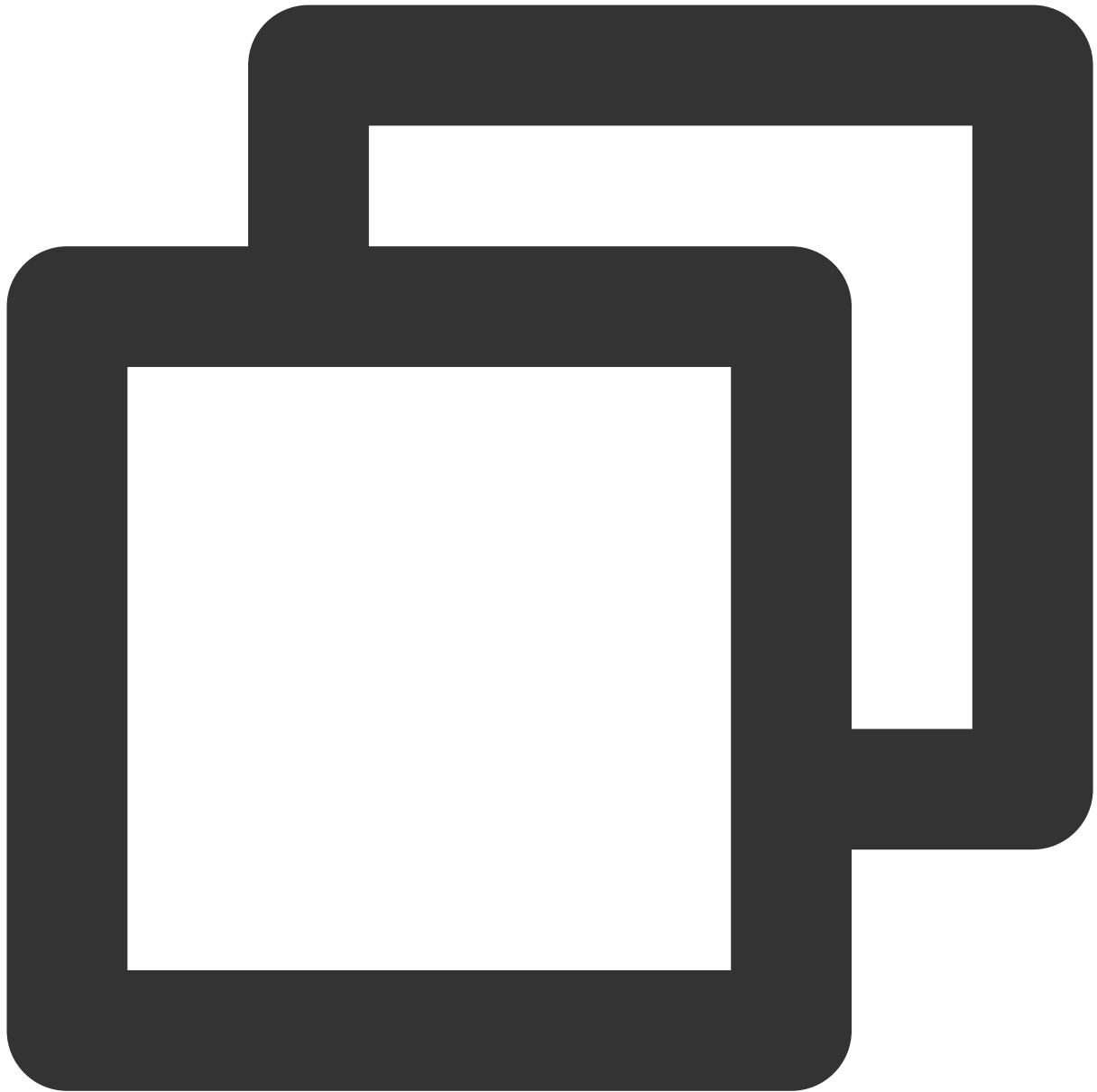
The log query feature can be used normally only after logs are reported. This document describes how to report logs to RUM.

Prerequisites

Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

Log Reporting

Pass in the following parameters to configure the SDK and report logs:



```
// `info` can report any string, number, array, and object, but it reports data only on the console
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// You can also report a specified object and pass in the `ext` and `trace` parameters
// You must pass in the `msg` field in this case
aegis.info({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
```

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

// Different from `info`, `infoAll` reports full data
aegis.infoAll({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// `error` indicates a JavaScript error log, whose full data will also be reported.
aegis.error({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
aegis.error(new Error('Actively report an error'));

// The default log type of `aegis.report` is `report`, but currently you can pass i
aegis.report({
  msg: 'This is an Ajax error log',
  level: Aegis.LogType.AJAX_ERROR,
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
```

aid

Last updated : 2024-01-22 19:25:42

`aid` is a unique ID assigned by the Aegis SDK to each user device. It is stored in the mini program storage and used to distinguish between users for UV calculation. It will be updated only when the user clears the mini program cache.

Prerequisites

Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

aid

You can use the following algorithm to customize the `aid` reporting rules:



```
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);  
});
```

Instance Methods

Last updated : 2024-01-22 19:25:42

RUM provides various instance methods for data reporting. You can use them to modify the instance configuration and customize the events and resources for speed test to be reported.

Currently, RUM provides the following Aegis instance methods:

Parameter	Description
setConfig	Passes in the configuration object, which contains information such as user ID and UIN.
info	Is a main reporting field to report allowlist logs. A log will be reported to the backend only in the following cases: 1. The user who opens the page is in the allowlist. 2. The page has an error.
infoAll	Is a main reporting field to report allowlist logs. The only difference between it and <code>info</code> is as follows: <code>info</code> reports the logs of only specified users, while <code>infoAll</code> reports the logs of all users.
error	Is a main reporting field to report the error information.
report	Reports the information of a log in any type.
reportEvent	Reports a custom event.
reportTime	Reports a custom resource for speed test.
time	Reports a custom resource for speed test and is used together with <code>timeEnd</code> to calculate and report data between two time points.
timeEnd	Reports a custom resource for speed test and is used together with <code>time</code> to calculate and report data between two time points.
destroy	Terminates the Aegis instance.

Prerequisites

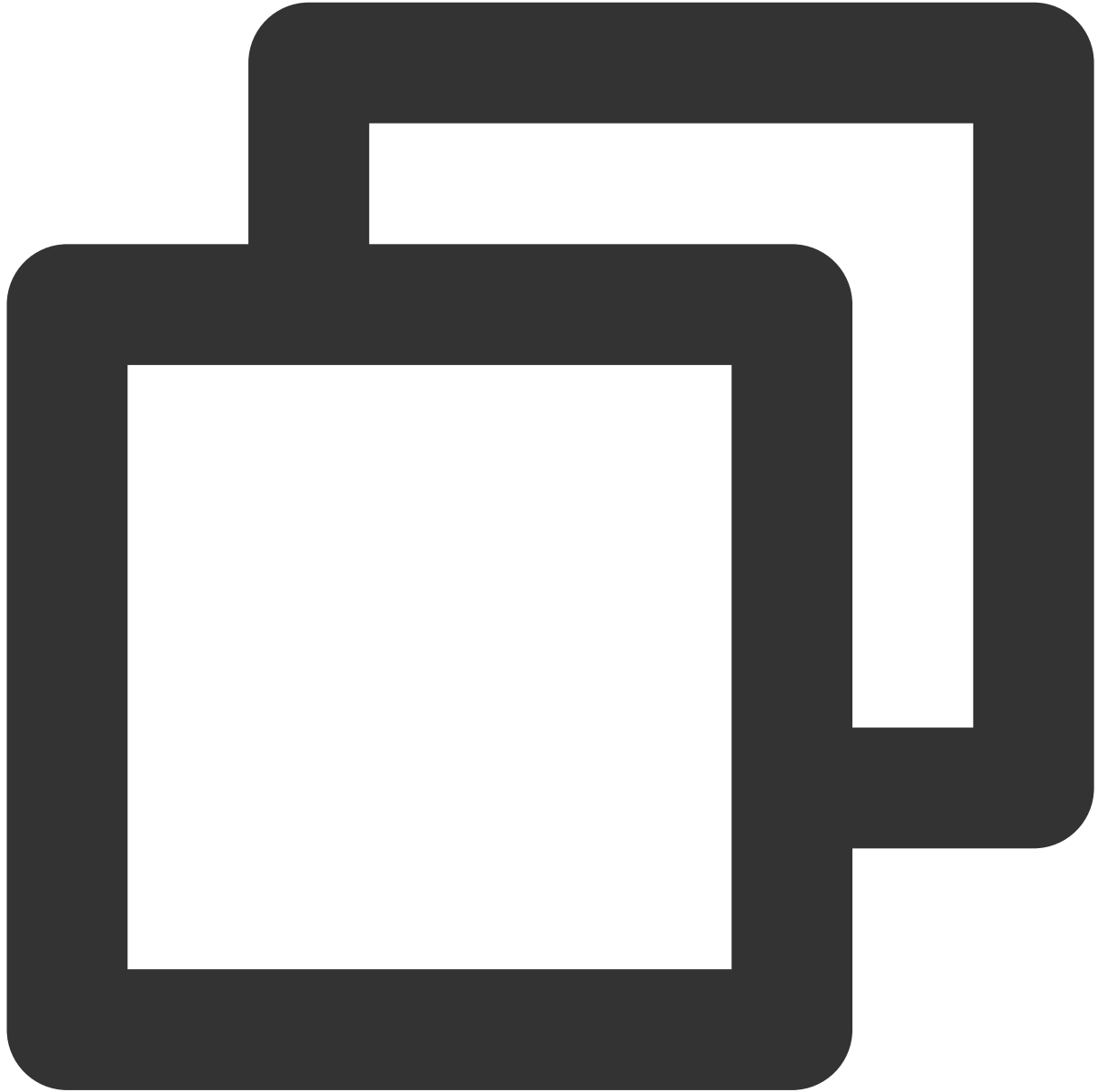
Install and initialize the Aegis SDK in any method as detailed in [Installation and Initialization](#).

Instance Methods

setConfig

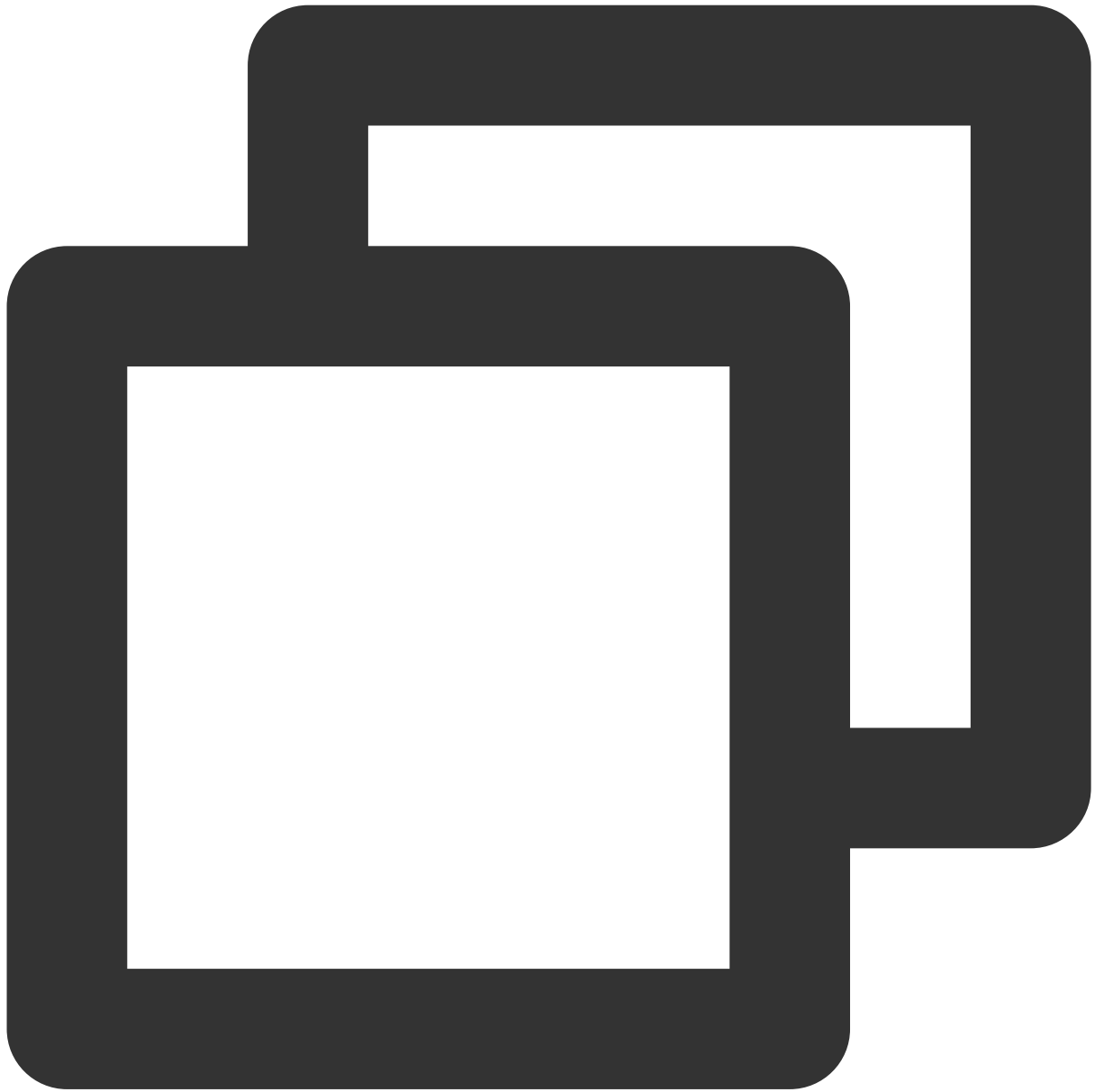
This method is used to modify the instance configuration in the following use cases:

1. You can get the user UIN and pass in two instance objects of user ID and UIN at the same time for instantiation:



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

2. Generally, the `uin` cannot be directly obtained in the beginning. If instantiation cannot be completed during the period when `uin` is obtained, RUM cannot listen on the errors occurring in this period. To solve this, you can pass in the ID first for instantiation and then import `setConfig` to pass in `uin` as follows:



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

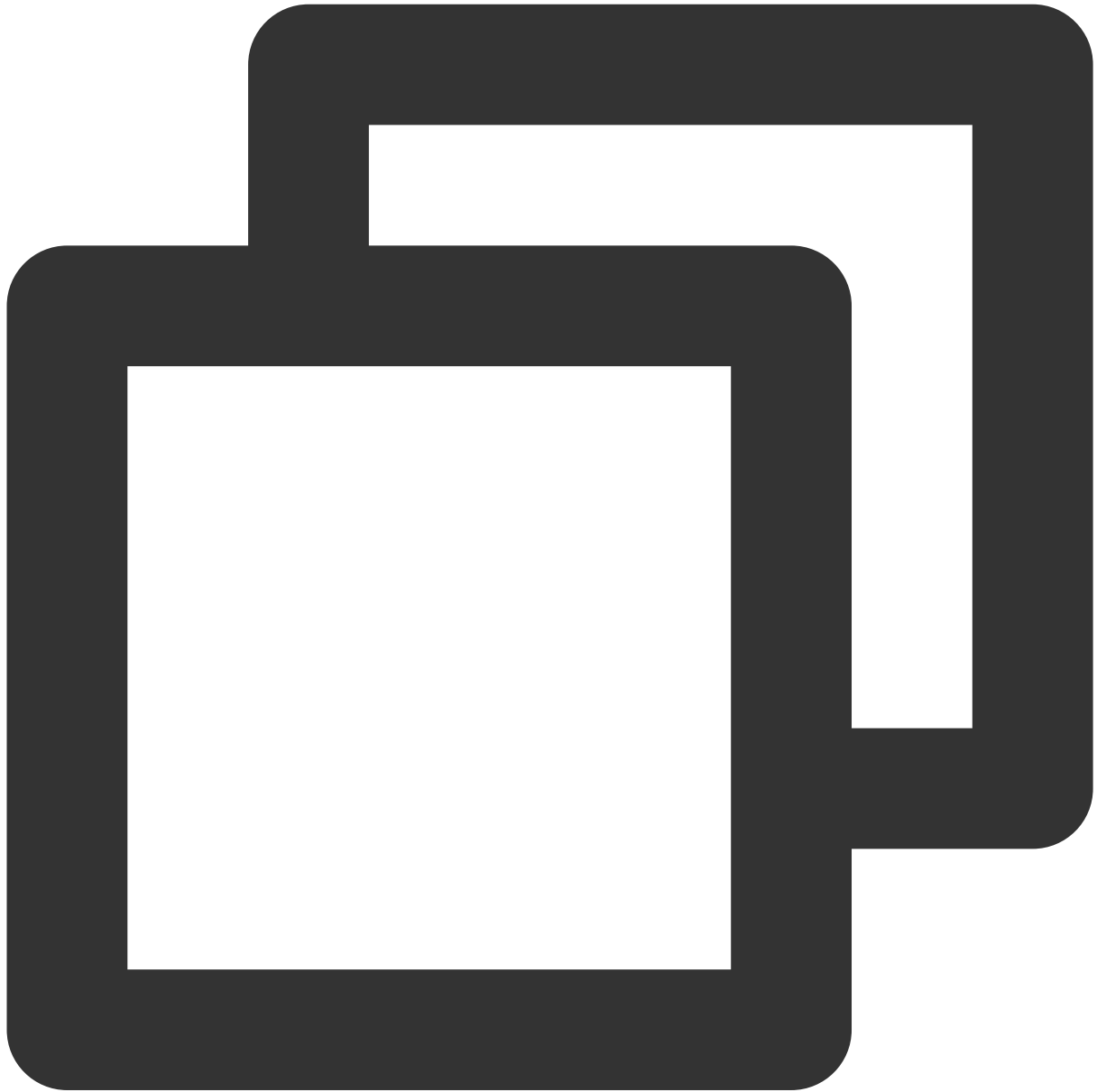
// After `uin` is obtained
aegis.setConfig({
  uin: '6666'
```



```
})
```

`info` , `infoAll` , `error` , and `report`

These are the main reporting methods provided by RUM.



```
// `info` can report any string, number, array, and object, but it reports data only on the client side
aegis.info('test');
aegis.info('test', 123, ['a', 'b', 'c', 1], {a: '123'});

// You can also report a specified object and pass in the `ext` and `trace` parameters
```

```
// You must pass in the `msg` field in this case
aegis.info({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// Different from `info`, `infoAll` reports full data
aegis.infoAll({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});

// `error` indicates a JavaScript error log, whose full data will also be reported.
aegis.error({
  msg: 'test',
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
aegis.error(new Error('Actively report an error'));

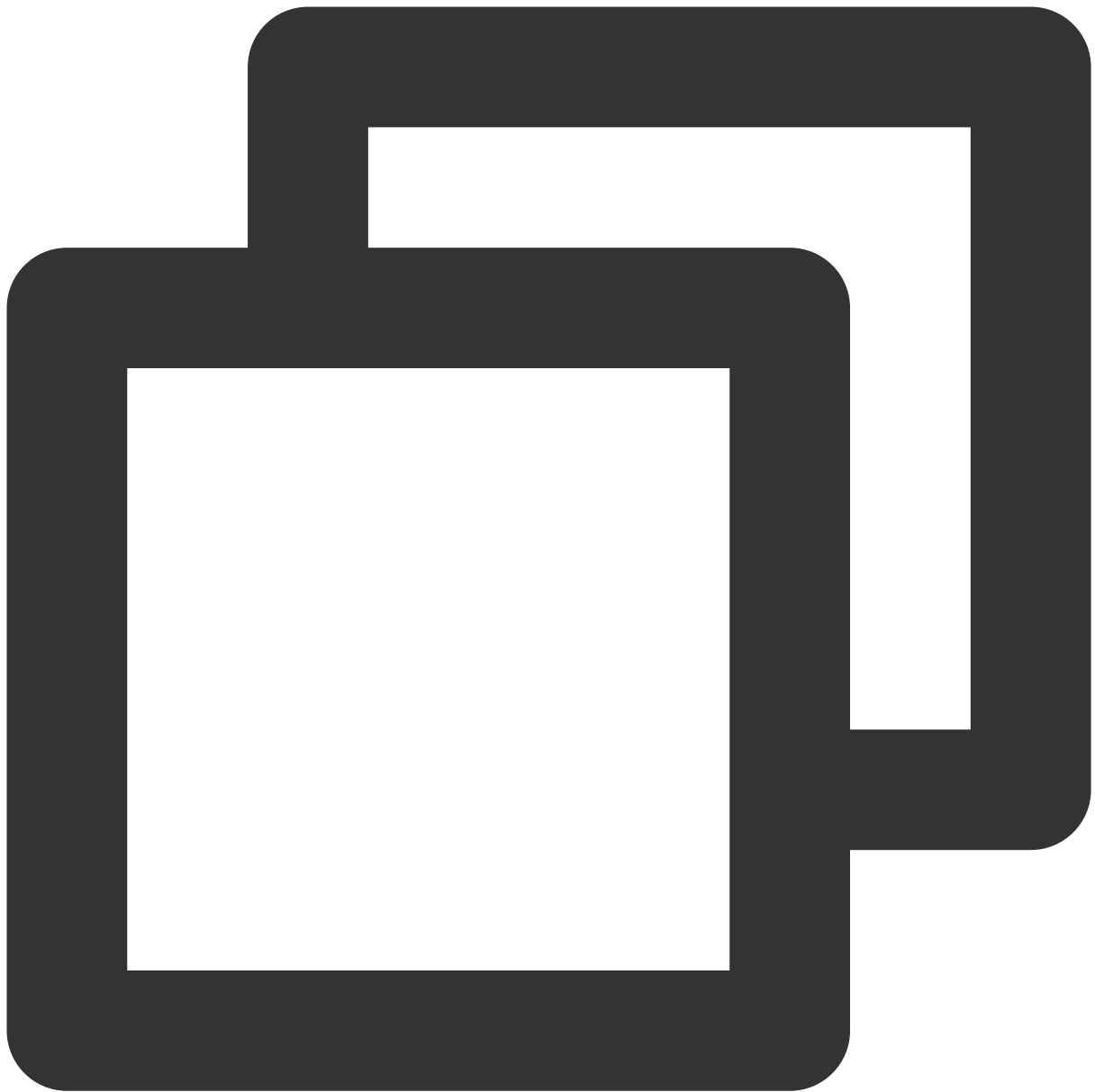
// The default log type of `aegis.report` is `report`, but currently you can pass i
aegis.report({
  msg: 'This is an Ajax error log',
  level: Aegis.LogType.AJAX_ERROR,
  ext1: 'ext1',
  ext2: 'ext2',
  ext3: 'ext3',
  trace: 'trace',
});
```

reportEvent

This method can be used to report a custom event, and the system will automatically collect the metrics of the reported event, such as PV and platform distribution.

`reportEvent` supports the string and object data types for the parameters to be reported.

String data type



```
aegis.reportEvent('The XXX request succeeded');
```

Object data type

`ext1` , `ext2` , and `ext3` use the parameters passed in when you use `new Aegis` . During custom event reporting, you can overwrite their default values.



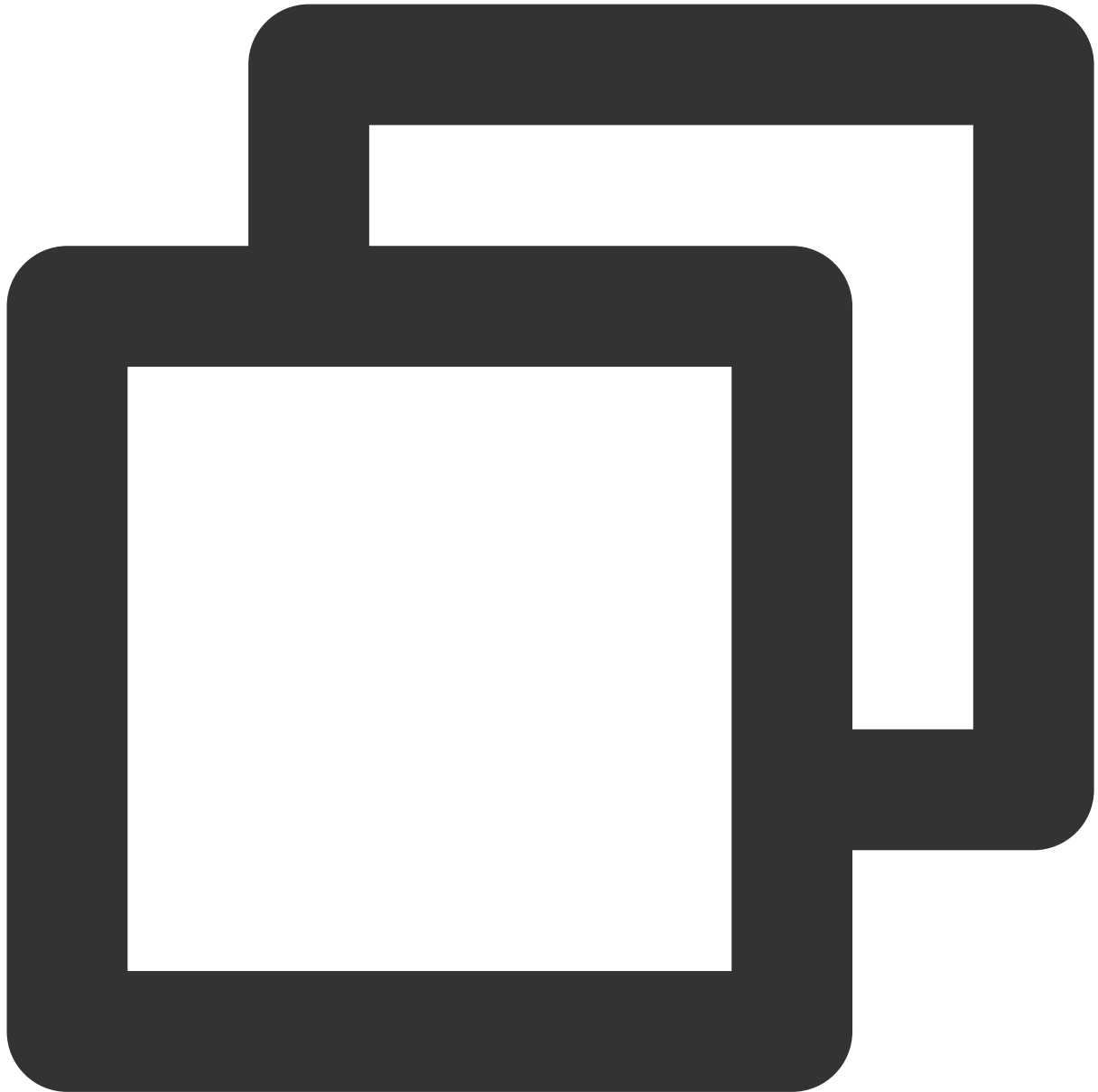
```
aegis.reportEvent({
  name: 'The XXX request succeeded', // Required
  ext1: 'Additional parameter 1',
  ext2: 'Additional parameter 2',
  ext3: 'Additional parameter 3',
})
```

Note:

The keys of the three additional parameters are fixed, which can only be `ext1` , `ext2` , and `ext3` currently.

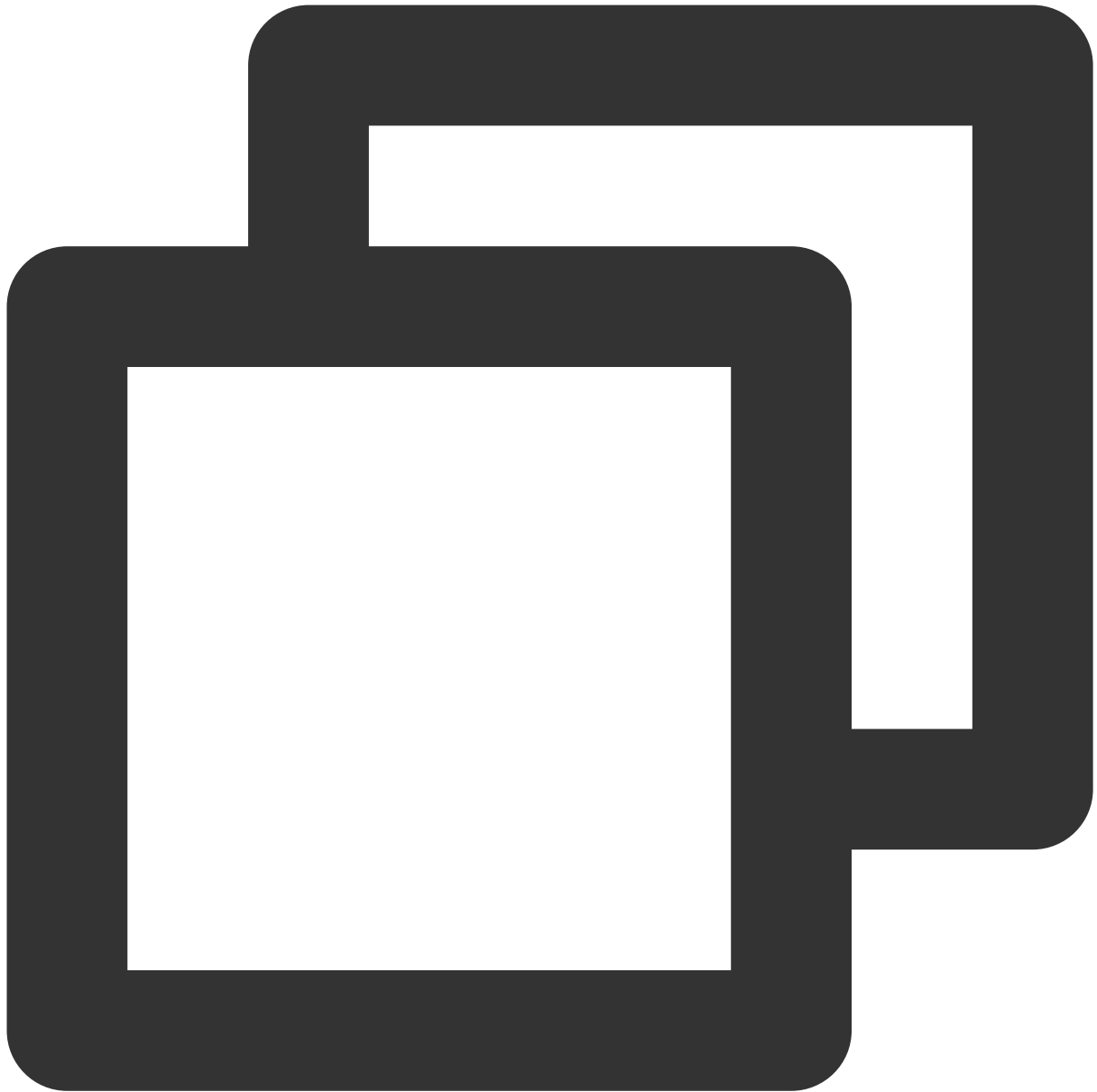
reportTime

This method can be used to report custom speed test as follows:



```
// Suppose the time of `onload` is 1 second  
aegis.reportTime('onload', 1000);
```

If you want to use additional parameters, you can pass them in by using the object type, and they will overwrite the default values of `ext1` , `ext2` , and `ext3` .



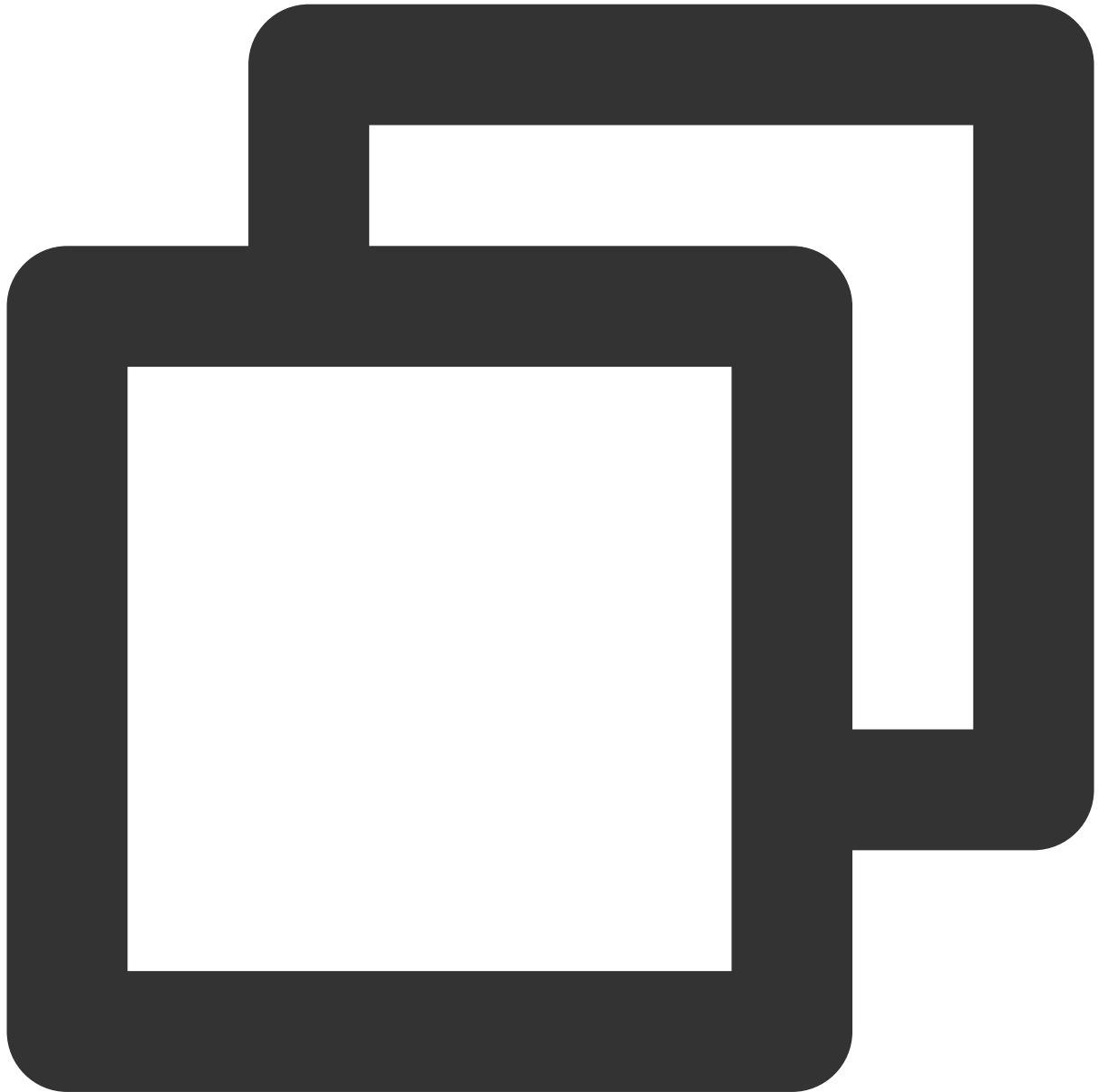
```
aegis.reportTime({  
  name: 'onload', // Custom speed test name  
  duration: 1000, // Custom speed test duration. Value range: 0-60000  
  ext1: 'test1',  
  ext2: 'test2',  
  ext3: 'test3',  
});
```

Note:

`onload` can be renamed.

time and **timeEnd**

These methods can be used to report a custom resource for speed test and are suitable for calculating and reporting a duration between two time points.



```
aegis.time('complexOperation');  
/**  
 * .  
 * .  
 * After complicated operations are performed for a long time  
 * .  
 * .  
 */
```

```
*/  
aegis.timeEnd('complexOperation'); /** At this point, the log has been reported**/
```

Note:

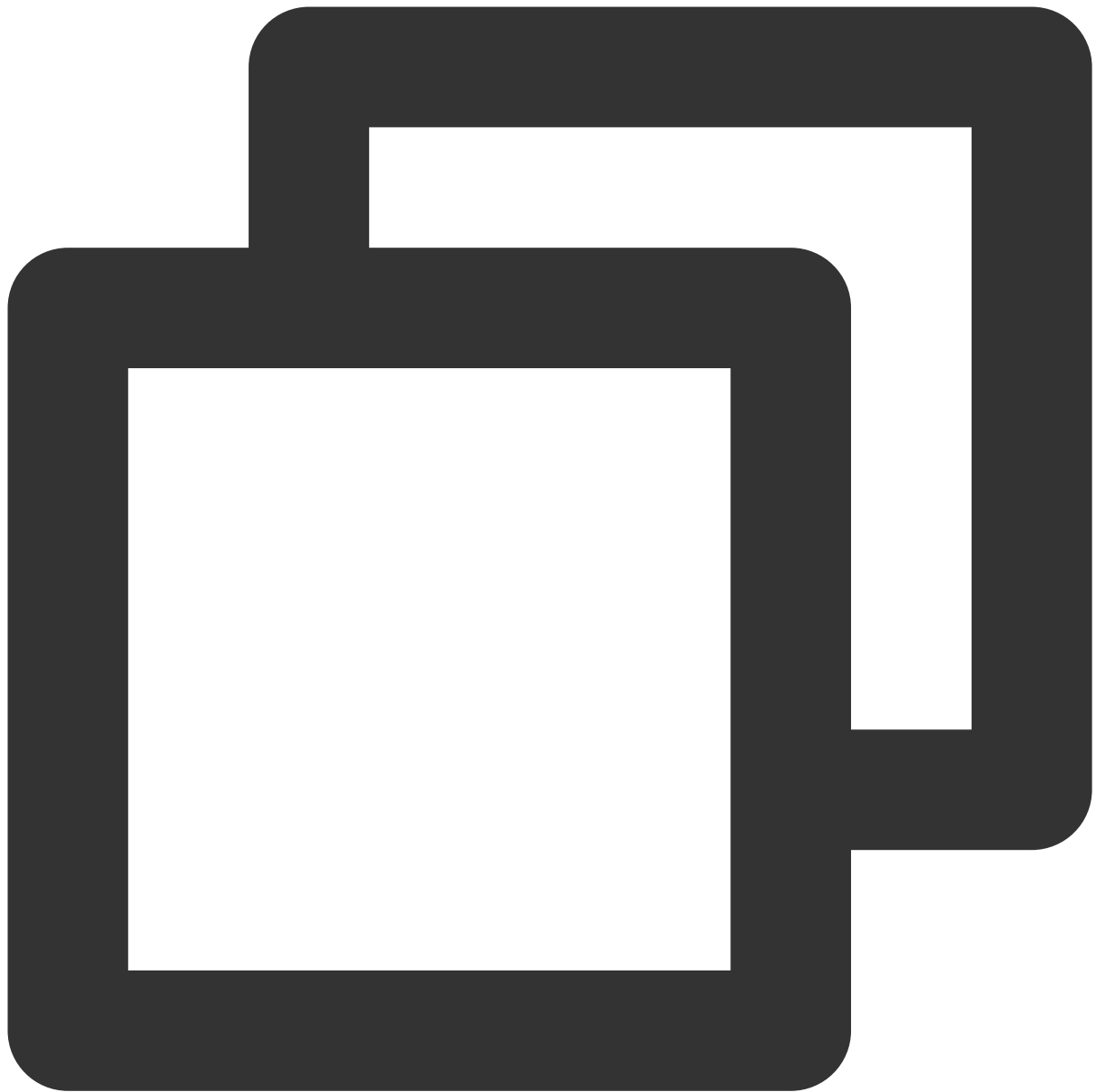
`complexOperation` can be renamed.

In custom speed test, you can report any values, and the server will collect and calculate them. As the server cannot process dirty data, we recommend you restrict the statistics values passed in to prevent the system from being affected by dirty data.

Currently, Aegis can calculate values only between 0 and 60000. If you use a greater value, we recommend you adjust it reasonably.

destroy

This method is used to terminate the instance process, after which no data will be reported, and Aegis will stop collecting user data.



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

Allowlist

Last updated : 2024-01-22 19:25:42

The allowlist feature is used to allow specified users to report more information. To filter out certain users and reduce the number of user API requests, RUM provides this feature and configures its logic.

The allowlist logic is as follows:

1. For users in the allowlist, all API information will be reported, including API request and response.
2. For users in the allowlist, the `info` API can be used to report data.
3. `info` and `infoAll` : in the actual development experience, for users in the allowlist, more logs can be added and reported with the `info` API. The `infoAll` API reports the data of all users without discrimination; therefore, it may result in a large volume of log data.
4. You can use the `whitelist` API to check whether the current user is in the allowlist, and if so, the response (`aegis.isWhiteList`) of the user will be bound to the Aegis instance for your use.
5. To reduce your workload, the allowlist takes effect at the business system level. You can go to **Application Management > Allowlist Management** to create an allowlist, and it will take effect for all applications in the current business system.

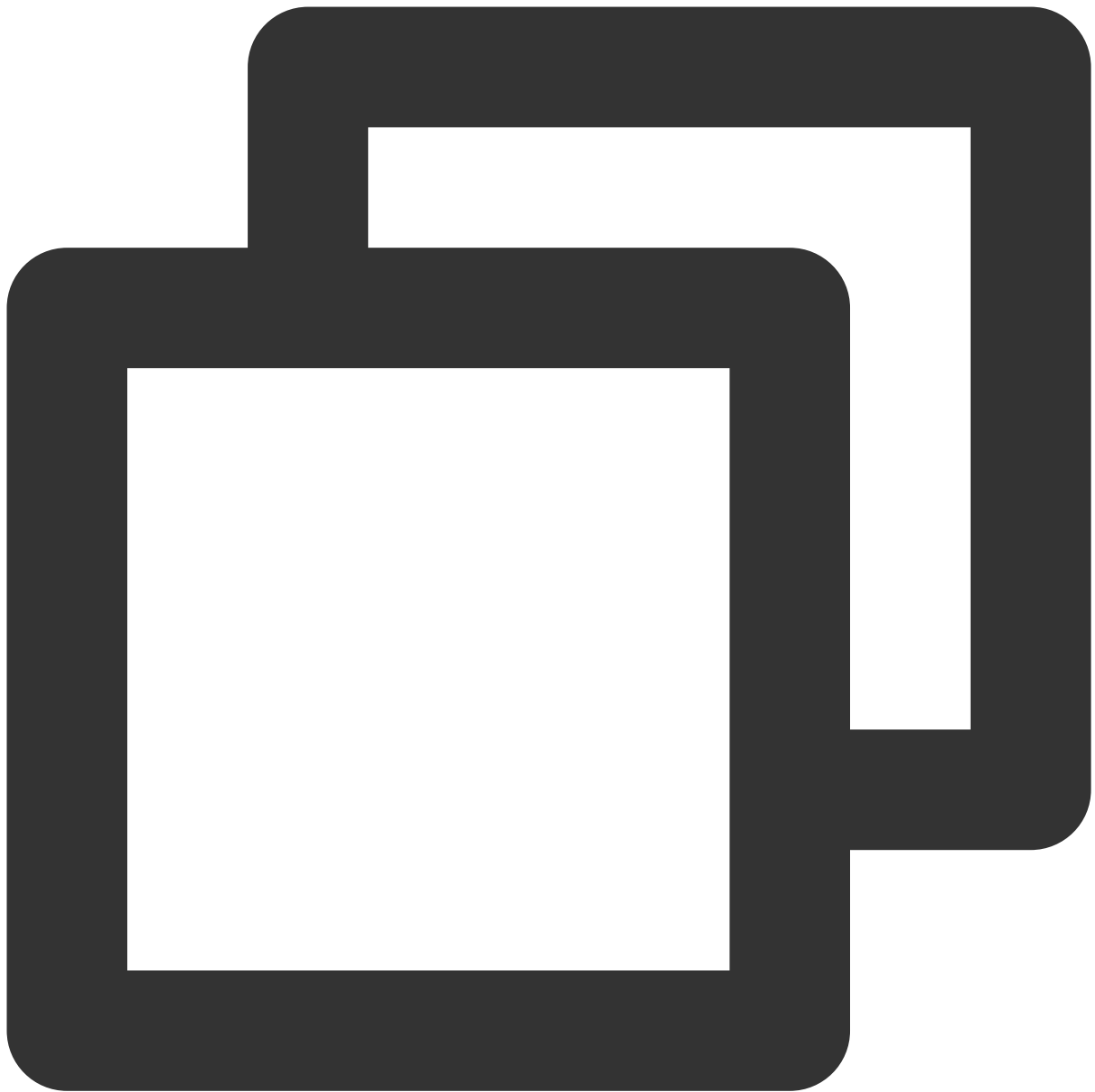
Hook Functions

Last updated : 2024-01-22 19:25:42

You can use a hook function to customize the configuration for resource speed test data reporting, and the system will calculate and collect the function execution duration. Then, you can use custom speed test to analyze the function execution duration in multiple dimensions.

beforeReport

1. This hook will be executed before log reporting (by the `/collect?` API); for example:



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    // Listen on the reported error below
    console.log(log); // {level: "4", msg: "An error occurred."}
    return log;
  }
});

throw new Error('An error occurred.');
```

Note:

The above `log` will have the following fields:

1. `level`: log level. For example, `'4'` indicates error log.
2. `msg``: log content.
3. Below are all log levels:

```
{ level: '1', name: 'API request log (allowed log)' }
{ level: '2', name: 'General log' }
{ level: '4', name: 'JavaScript execution error' }
{ level: '8', name: 'Promise error' }
{ level: '16', name: 'Ajax request exception' }
{ level: '32', name: 'JavaScript loading exception' }
{ level: '64', name: 'Image loading exception' }
{ level: '128', name: 'CSS loading exception' }
{ level: '256', name: 'console.error (reserved)' }
{ level: '512', name: 'Audio/Video resource exception' }
{ level: '1024', name: 'retcode exception' }
{ level: '2048', name: 'aegis report' }
```

2. If this hook returns `false`, the log won't be reported. This feature can be used to filter out errors that don't need to be reported; for example:



```
const aegis = new Aegis({
  id: 'pGUVFTCZyewxxxxx',
  beforeReport(log) {
    if (log.level === '4' && log.msg && log.msg.indexOf('An error that doesn't need
      return false
    }
  }
});
throw new Error('An error that doesn't need to be reported'); // This error will no
```

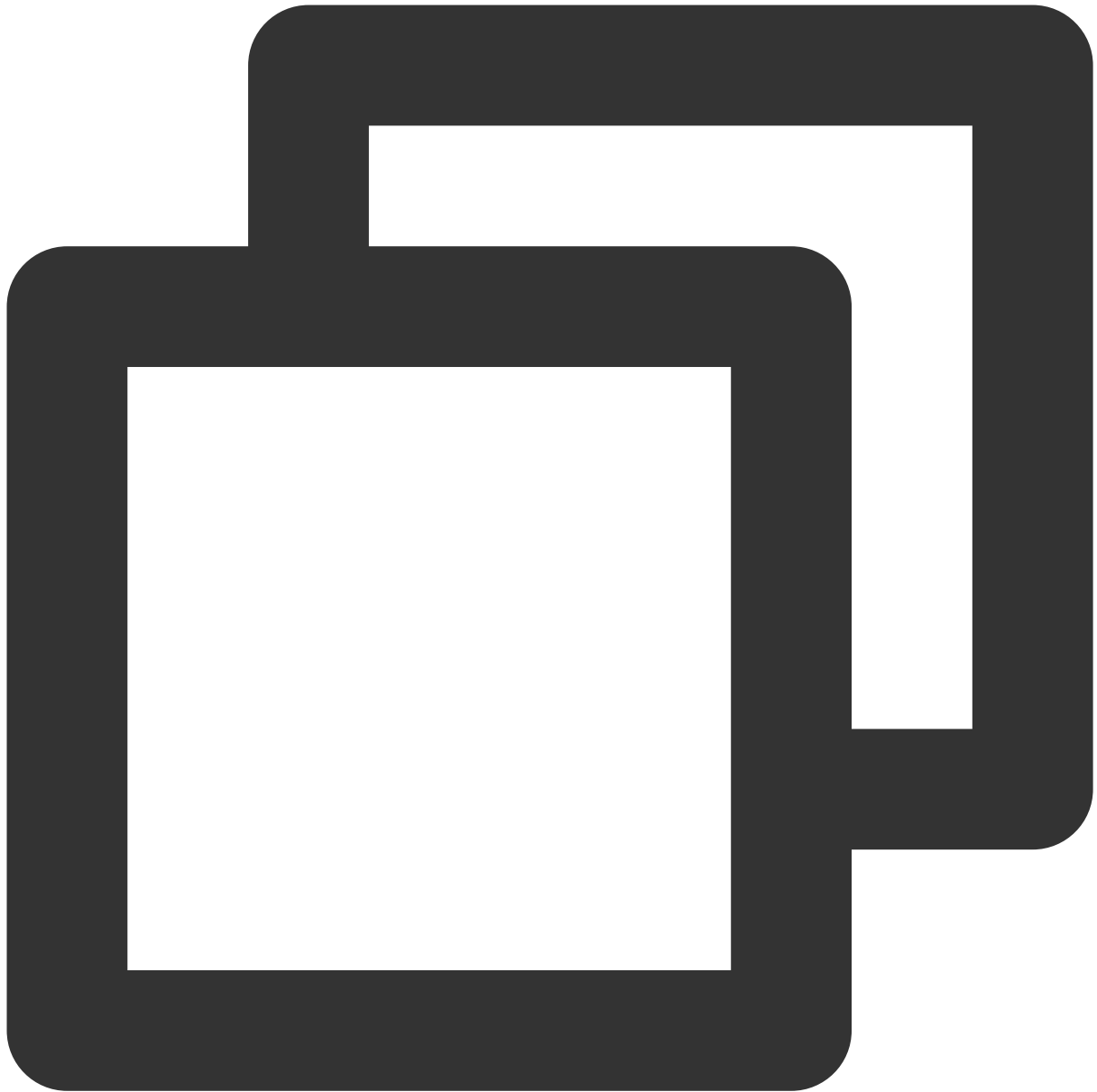
In the above sample code, if the error content contains the `An error that doesn't need to be reported` keywords, it won't be reported to the RUM backend.

onReport

This hook is executed after a log is successfully reported, and its usage is similar to that of `beforeReport`. Their only difference is that all parameters received by this hook are of an already reported log, but the parameters received by `beforeReport` are of a log to be reported.

beforeReportSpeed

1. This hook will be executed before the speed test data is reported; for example:



```
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
  }
});
```

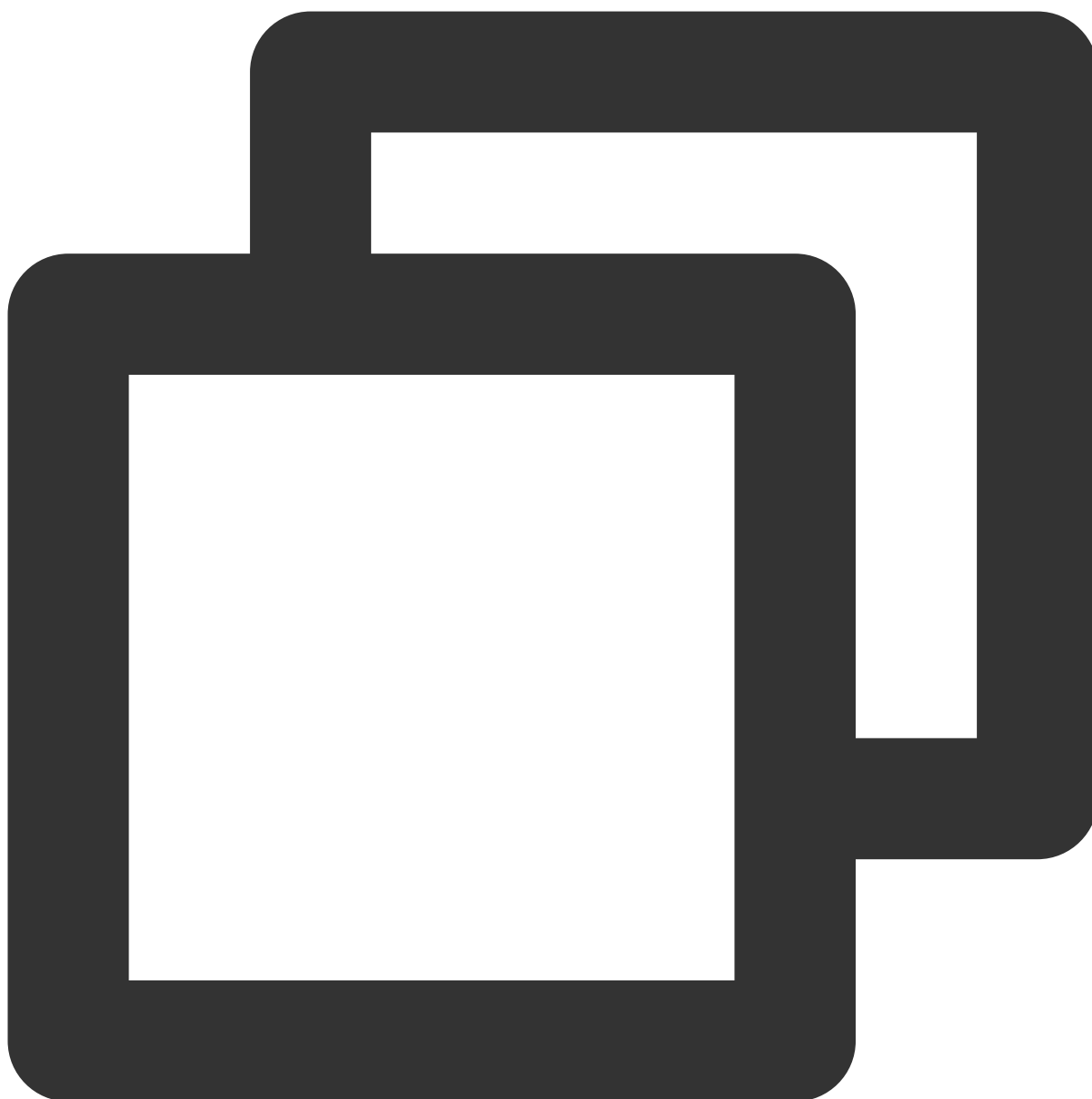
Note:

The above `msg` will have the following fields:

1. `url` : request address of this resource.
2. `type` : resource type. Valid values: `fetch` : Aegis will report the resource as an API request; `static` : Aegis will report the resource as a static resource.
3. `duration` : resource request duration.
4. `method` : `http method` used when the resource is requested.
5. `status` : status code returned by the server.

In the above sample code, every time after Aegis collects the loading details of a resource, it will use such details (i.e., `msg` returned above) as parameters to call the `beforeReportSpeed` hook.

2. If you have configured this hook, the final content reported by Aegis is subject to the execution result of the hook; for example:



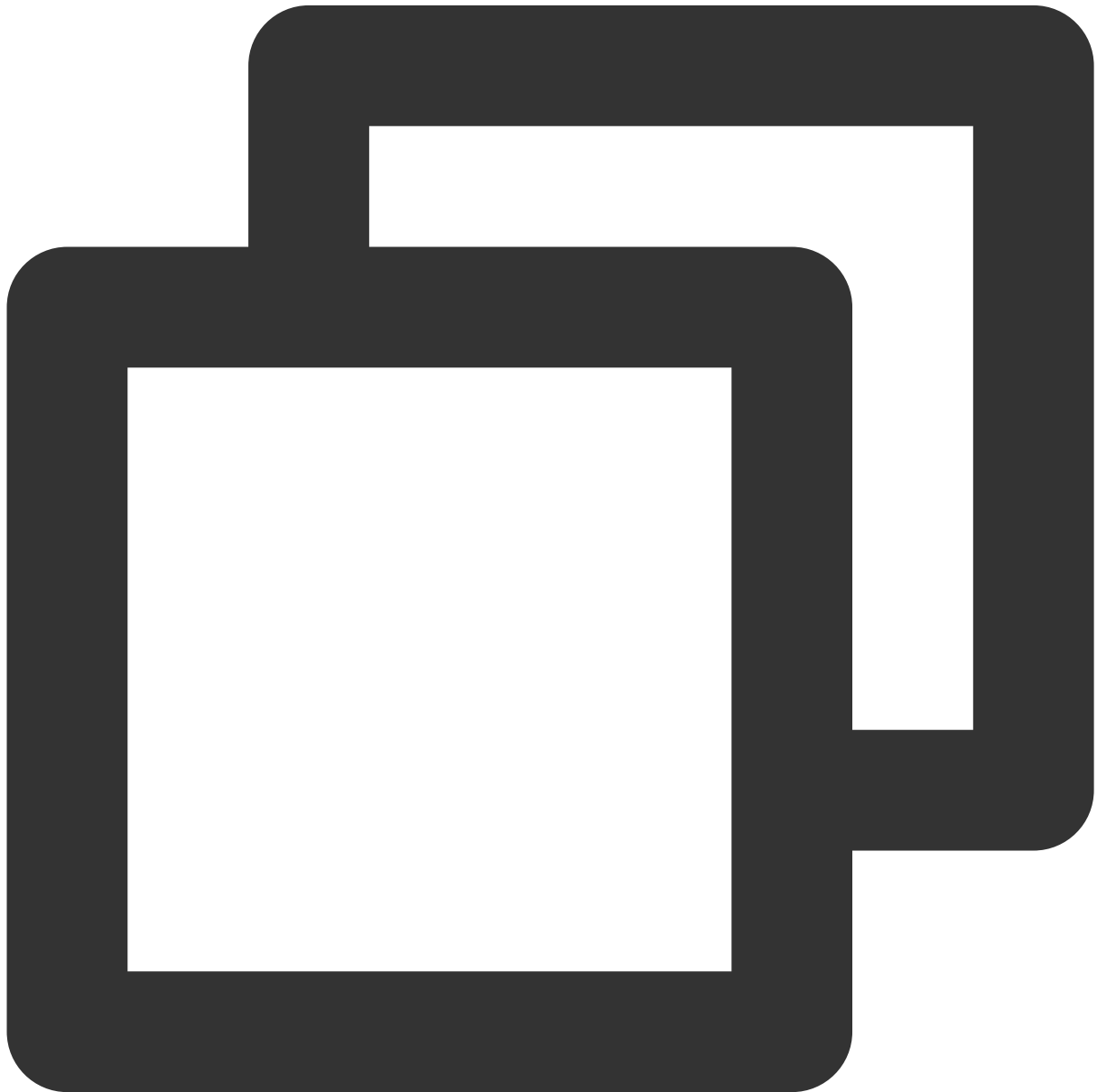
```
const aegis = new Aegis({  
  id: 'pGUVFTCZyewxxxxx',  
  reportApiSpeed: true,  
  reportAssetSpeed: true,  
  beforeReportSpeed(msg) {  
    msg.type = 'static';  
  }  
});
```

In the above sample code, all `msg.type` parameters are set to `static`, indicating that all resources, even API requests, will be reported as static resources.

3. You can use this hook to correct the Aegis type and judge incorrect requests.

Sample

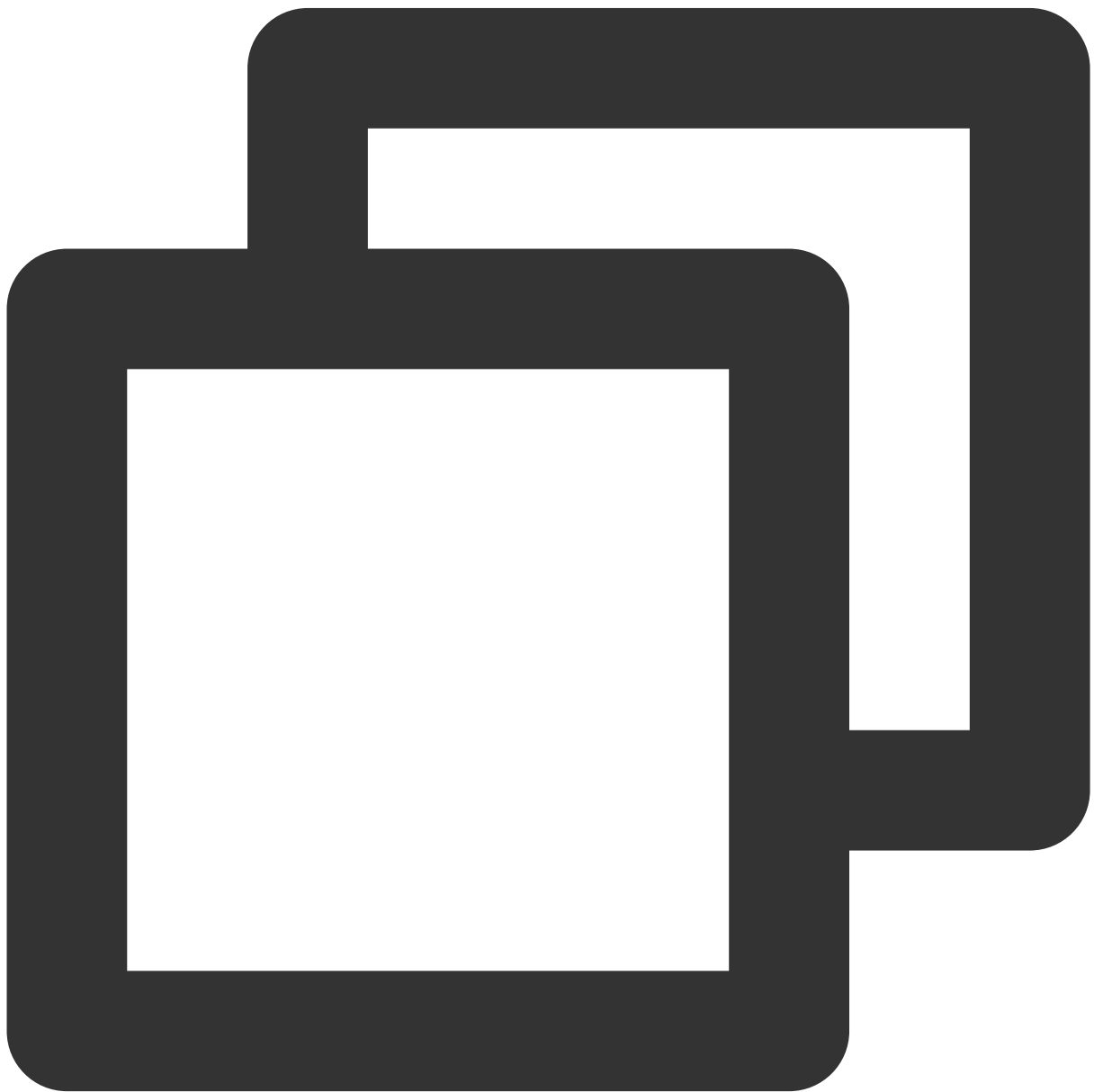
You have an API `https://example.com/api` whose response header `Content-Type` is `text/html`. In normal case, RUM will report this API as a static resource; however, it must be reported as an API request in your business. Then, you can configure Aegis with following hook for correction:



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

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

1. You can also block the speed test data reporting of a resource as follows:



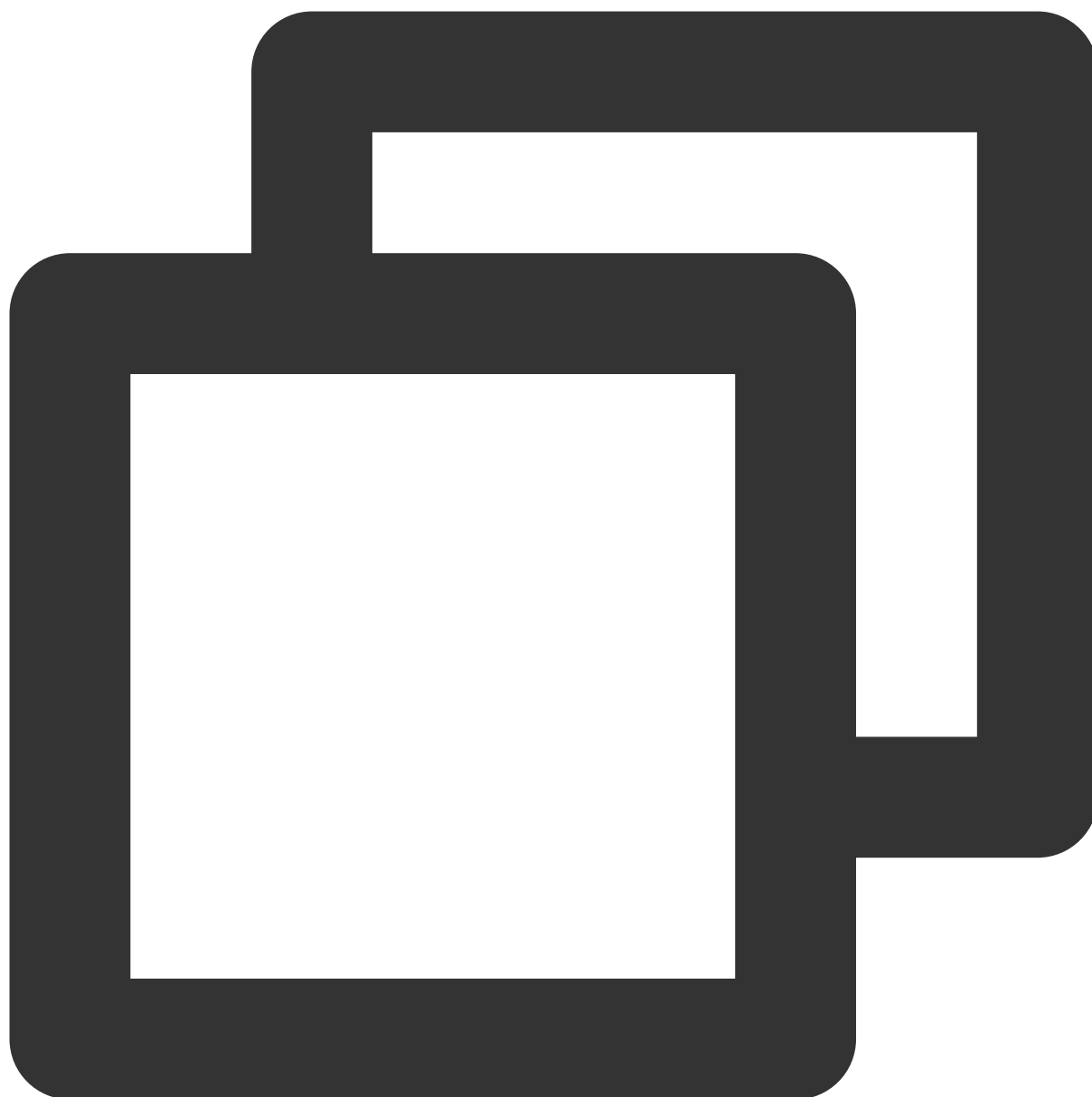
```
const aegis = new Aegis({
  id: 'pGUVFTCYewxxxxx',
  reportApiSpeed: true,
  reportAssetSpeed: true,
  beforeReportSpeed(msg) {
    // Speed test logs for resources whose address contains `https://example.com/ap
    if (msg.url.indexOf('https://example.com/api') !== -1) {
      // If `false` is returned, the reporting of the speed test log will be blocke
      return false
    }
  }
});
```

beforeRequest

This hook will be executed before log reporting; for example:

Note:

The SDK version must be 1.24.44 or above.



```
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;
  }
});
```

Here, `msg` will have the following fields:

1. `logType`: log type. Valid values:

`custom`: custom speed test.

`event`: custom event.

`log`: log.

`performance`: page speed test.

`pv`: PV.

`speed`: API and static resource speed test.

`vitals`: Web Vitals.

2. `logs`: reported log content:

If `logType` is `'custom'`, `logs` will be in the format of `{name: "white screen time", duration: 3015.7000000178814, ext1: '', ext2: '', ext3: ''}`.

If `logType` is `'event'`, `logs` will be in the format of `{name: "ios", ext1: "", ext2: "", ext3: ""}`.

If `logType` is `'performance'`, `logs` will be in the format of `{contentDownload: 2, dnsLookup: 0, domParse: 501, firstScreenTiming: 2315, resourceDownload: 260, ssl: 4, tcp: 4, ttfb: 5}`.

If `logType` is `'speed'`, `logs` will be in the format of `{connectTime: 0, domainLookup: 0, duration: 508.2, isHttps: true, method: "get", status: 200, type: "tatic", url: "https://xxxxxx", urlQuery: "max_age=1296000"}`.

If `logType` is `'vitals'`, `logs` will be in the format of `{delta: 1100, entries: [PerformancePaintTiming], id: "v1-1629344653118-4916457684758", name: "CP", value: 1100}`.

If `logType` is `'log'`, `logs` will be in the format of `{msg: "log details", level: '4', ext1: '', ext2: '', ext3: '', trace: ''}`.

Note:

Valid values of `level`:

```
{ level: '1', name: 'API request log (allowed log)' }
```

```
{ level: '2', name: 'General log' }
```

```
{ level: '4', name: 'JavaScript execution error' }
```

```
{ level: '8', name: 'Promise error' }
```

```
{ level: '16', name: 'Ajax request exception' }
```

```
{ level: '32', name: 'JavaScript loading exception' }
```

```
{ level: '64', name: 'Image loading exception' }
```

```
{ level: '128', name: 'CSS loading exception' }
```

```
{ level: '256', name: 'console.error (reserved)' }
```

```
{ level: '512', name: 'Audio/Video resource exception' }
```

```
{ level: '1024', name: 'retcode exception' }
```

```
{ level: '2048', name: 'aegis report' }
```

If this hook returns `false`, the log won't be reported. This feature can be used to filter out errors and logs that don't need to be reported.

afterRequest

This hook will be executed after speed test data reporting; for example:

Note:

The SDK version must be 1.24.44 or above.



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

Here, `msg` will have the following fields:

1. `isErr`: whether the request reporting API has an error.

- 2. result: response of the reporting API.
- 3. logs: reported log content.
- 4. logType: log type, which is the same as `logType` in `beforeRequest` .

Error Monitoring

Last updated : 2024-01-22 19:25:42

The Aegis instance of RUM automatically monitors exceptions such as JavaScript and Promise execution errors and Ajax (fetch) request exceptions. This document describes the monitoring logic of each error type and how to handle them.

Note:

The Aegis instance monitors such exceptions. It will not report any data if you merely import the Aegis SDK and don't instantiate it.

JavaScript Execution Error

Aegis listens on the `onerror` event in the `wx` object to capture project errors. Then, it parses errors and analyzes the heaps and stacks to automatically report the error information to the backend service. In this case, the report level is `error`; therefore, when the number of automatically reported errors reaches the threshold, Aegis will automatically trigger an alarm to help you find the exceptions promptly. As the report level is `error`, automatic reporting will also affect the project score.

Request Exception

Aegis will rewrite `wx.request` / `qq.request` and listen on each API request. If `statusCode` does not exist or is greater than 400, the request will be considered a failure, timeout, or abortion.

If you use a library that will also rewrite `wx.request` / `qq.request`, such as `miniprogram-api-promise`, in your project, or you also perform encapsulation, you must initialize Aegis before importing the library; otherwise, API failures cannot be listened on.

Note:

When an error occurs, the Aegis SDK doesn't actively collect API request and response parameters. You can configure the `apiDetail` parameter in the API calling to enable reporting of such information.



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

retcode Exception

Aegis rewrites `wx.request` / `qq.request` to get the API response. It also tries to get the `retcode` of the request in the response. If the `retcode` doesn't meet the expectation, the request will be considered exceptional and will be reported.

Note:

You can get `retcode` and judge which `retcode` values are normal as instructed in [Configuration Guide](#).

Performance Monitoring

Last updated : 2024-01-22 19:25:42

This document describes page and API speed tests.

Page Speed Test

The Aegis SDK for mini program automatically collects and reports page performance data, including:

1. Program start time.
2. Code injection time.
3. First contentful paint (FCP).
4. Route switch time.

Note:

Acquisition of the page data depends on the Performance API of the mini program. Make sure that the base library version is above 2.11.0.

QQ Mini Program does not yet implement the Performance API and therefore cannot report the page performance data.

API Speed Test

Note:

How to enable: pass in `reportApiSpeed: true` during initialization.

Aegis tests the API speed by hijacking `wx.request || qq.request`.

Configuration Guide

Last updated : 2024-01-22 19:25:42

Configuration Description

The configuration items in the configuration file are as detailed below:

Configuration Item	Description
id	It is a required number and is empty by default. It is the project key assigned by your platform.
uin	It is a recommended string and is the UIN field in the cookie by default. It is the unique ID of the current user. When a log is reported, it will be used to check whether the user is in the allowlist. Its value can contain only letters, digits, and <code>@=._-</code> , and its regular expression is <code>/^[@=._0-9a-zA-Z_-]{1,60}\$/</code> .
reportApiSpeed	It is an optional boolean value and is <code>false</code> by default. It specifies whether to enable API speed test.
version	It is an optional string and is the SDK version number by default. It is the version of the currently reported content. If the page uses PWA or an offline package, it can be used to judge the version of the code where the currently reported content is from. Its value can contain up to 60 letters, digits, and <code>.,:_-</code> , and its regular expression is <code>/^[0-9a-zA-Z.,:_-]{1,60}\$/</code> .
delay	It is an optional number and is 1000 ms by default. It is the time period for reducing reporting traffic, within which multiple reports will be merged into one reporting request.
repeat	It is an optional number and is 5 by default. It is the number of repeated reports. After it is exceeded, the same error will not be reported again.
env	It is an optional enum and is <code>Aegis.environment.production</code> by default. It is the current environment where the project runs.
spa	It is an optional boolean value and is <code>false</code> by default. It specifies whether to report the PV during page redirect in the mini program.
offlineLog	It is an optional boolean value and is <code>false</code> by default. It specifies whether to use offline log.
offlineLogExp	It is an optional number and is 3 by default.

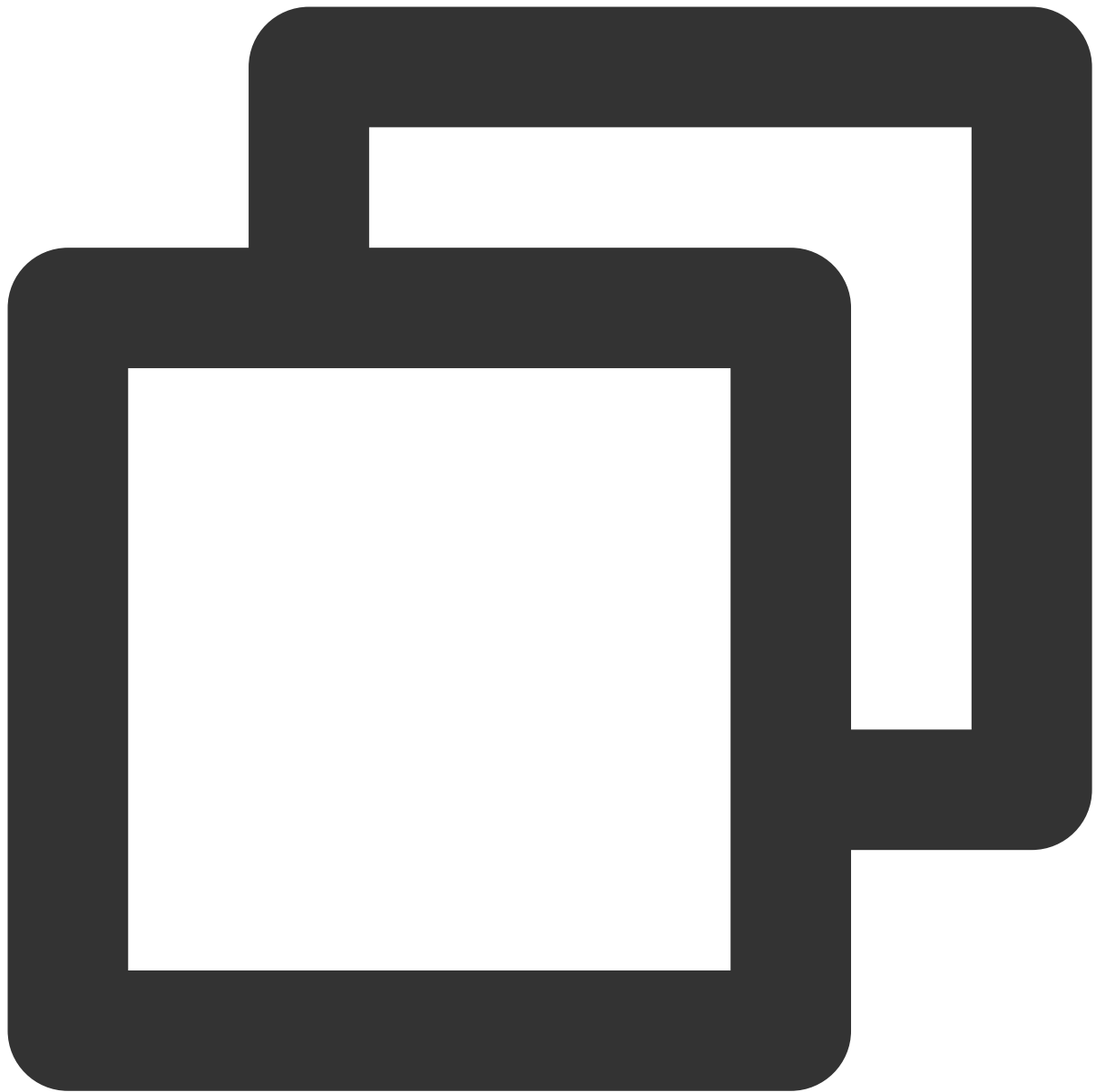
	It is the offline log validity period.
url	It is an optional string and is <code>//aegis.qq.com/collect</code> by default. It is the log reporting address. You can set it to an empty string to disable log reporting.
pvUrl	It is an optional string and is <code>//aegis.qq.com/collect/pv</code> by default. It is the PV reporting address. You can set it to an empty string to disable PV reporting.
whiteListUrl	It is an optional string and is <code>//aegis.qq.com/collect/whitelist</code> by default. It is the allowlist confirming API. You can set it to an empty string to disable allowlist API request.
offlineUrl	It is an optional string and is <code>//aegis.qq.com/collect/offline</code> by default. It is the offline log reporting address. You can set it to an empty string to disable offline log reporting.
eventUrl	It is an optional string and is <code>//aegis.qq.com/collect/events</code> by default. It is the custom event reporting address. You can set it to an empty string to disable custom event reporting.
speedUrl	It is an optional string and is <code>//aegis.qq.com/speed</code> by default. It is the speed test log reporting address. You can set it to an empty string to disable speed test data reporting.
customTimeUrl	It is an optional string and is <code>//aegis.qq.com/speed/custom</code> by default. It is the custom speed test reporting address. You can set it to an empty string to disable custom speed test reporting.
performanceUrl	It is an optional string and is <code>//aegis.qq.com/speed/performance</code> by default. It is the page performance reporting address. You can set it to an empty string to disable page performance reporting.
setDataReportConfig	It is an optional object and is <code>{}</code> by default. Fields: disabled: it specifies whether to disable <code>setData</code> data reporting. It is an optional boolean value and is <code>false</code> by default. timeThreshold: it is the reporting duration threshold. It is an optional number and is 30 by default. Only data whose update duration exceeds this threshold will be reported. withDataPaths: it specifies whether to report the field information updated currently. It is an optional boolean value and is <code>true</code> by default.
api	It is an optional object and is <code>{}</code> by default. Fields: apiDetail: it specifies whether to report the API request parameters and returned value if an API fails. It is an optional boolean value and is <code>false</code> by default.

	<p>retCodeHandler: it is the hook function for return code reporting and will pass in the API response data. The returned value is <code>{isErr: boolean, code: string}</code>. For more information, see api.retCodeHandler.</p> <p>reportRequest: it is a boolean value and is <code>false</code> by default. After it is enabled, <code>aegis.info</code> will report the full data with no need to configure the allowlist, and information of all APIs will be reported (you need to enable <code>reportApiSpeed</code> in the reporting API).</p>
ext1	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.
ext2	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.
ext3	It is the additional dimension in custom reporting, which can be overwritten during reporting. It is an optional string.

Sample Code

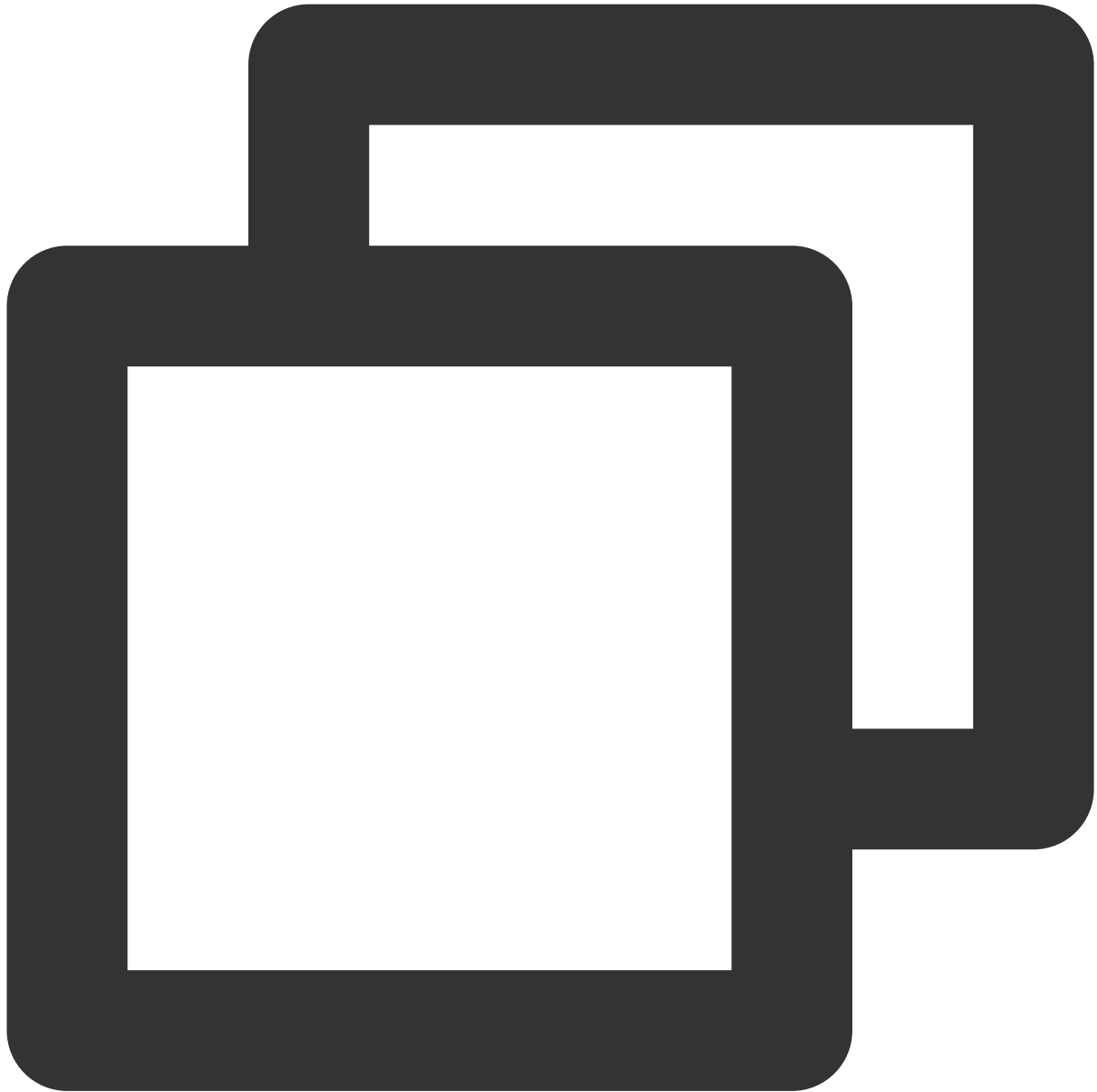
api.retCodeHandler

If the backend returns the following data:



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

If your business requires that if the `code` is not 200 or `retCode` is not 0, the request is incorrect. To meet this requirement, you can simply configure as follows:



```
new Aegis({  
  // xxx  
  reportApiSpeed: true, // You need to enable two speed test APIs; otherwise, no re  
  reportAssetSpeed: true,  
  api: {  
    retCodeHandler(data) {  
      // Note that the obtained `data` is a string. If you want to get an object, you need  
      try {
```

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

IOS Use Cases

Integration and Initialization

Last updated : 2024-04-25 11:42:15

This document describes how to integrate and initialize the SDK.

Directions

Step 1: Integrating the SDK

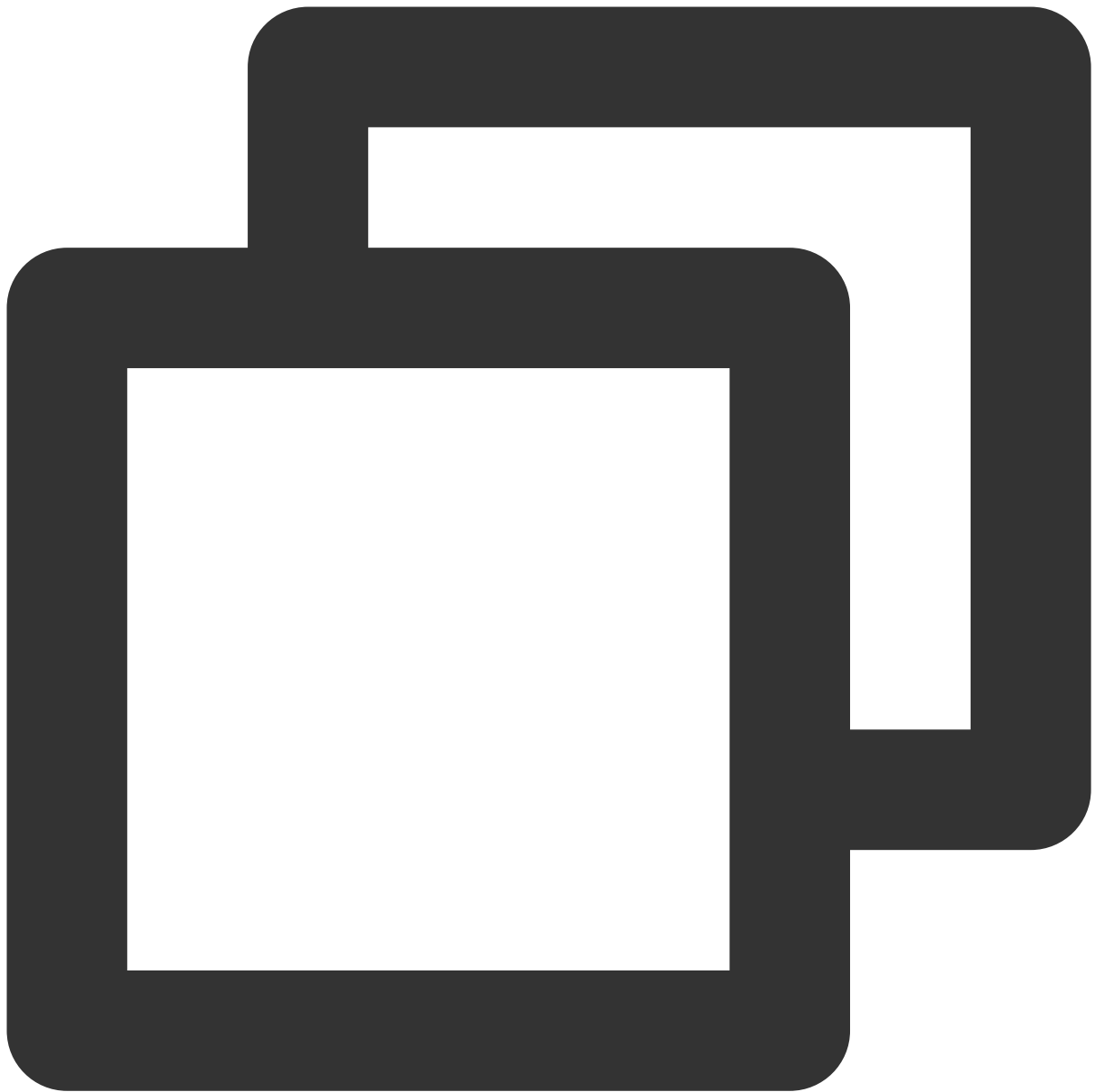
Choose any of the following methods to integrate the SDK.

Method 1: Integration via CocoaPods

Note:

Currently, only integration via local CocoaPods is supported.

1. Install [CocoaPods](#).
2. Update the CocoaPods version of QAPM via pod repo update.
3. In the corresponding target of Podfile, enter the following command and run pod install.



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

4. Use the .xcworkspace generated by CocoaPods to run the project, and set the project's bitcode parameter to NO.

Note:

If the following error occurs during the pulling process:

```
[!] Unable to add a source with url https://github.com/TencentCloud/QAPM-iOS-CocoaPods.git named tencentcloud-qapm-ios-cocoapods. You can try adding it manually in /Users/wxy/.cocoapods/repos or via pod repo add. You can run the following command in a terminal window:
```

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

Then run:

```
pod install
```

Method 2: Manual Integration

1. Copy the QAPM.framework under the [Demo project](#) QAPM_iOS_SDK_Demo to the business project.
2. Drag the QAPM.framework file into the Xcode project (and check the Copy items if needed).
3. In TARGETS -> Build Phases-Link Binary Libraries, add the dependencies libc++.tbd, libz.tbd, and CoreLocation.
4. Drag QAPM.bundle and js_sdk.js in framework into the business project.
5. Add -ObjC parameter in the project's Other Linker Flags.
6. Set the project's bitcode parameter to NO.

Step 2: Initializing the SDK and Web Environment

1. Log in to [TCOP](#) console. On the **Mobile Performance Monitoring** page, choose **Application Management** > **Application Settings** in the left sidebar, and access Appkey (reporting ID).

Application Management

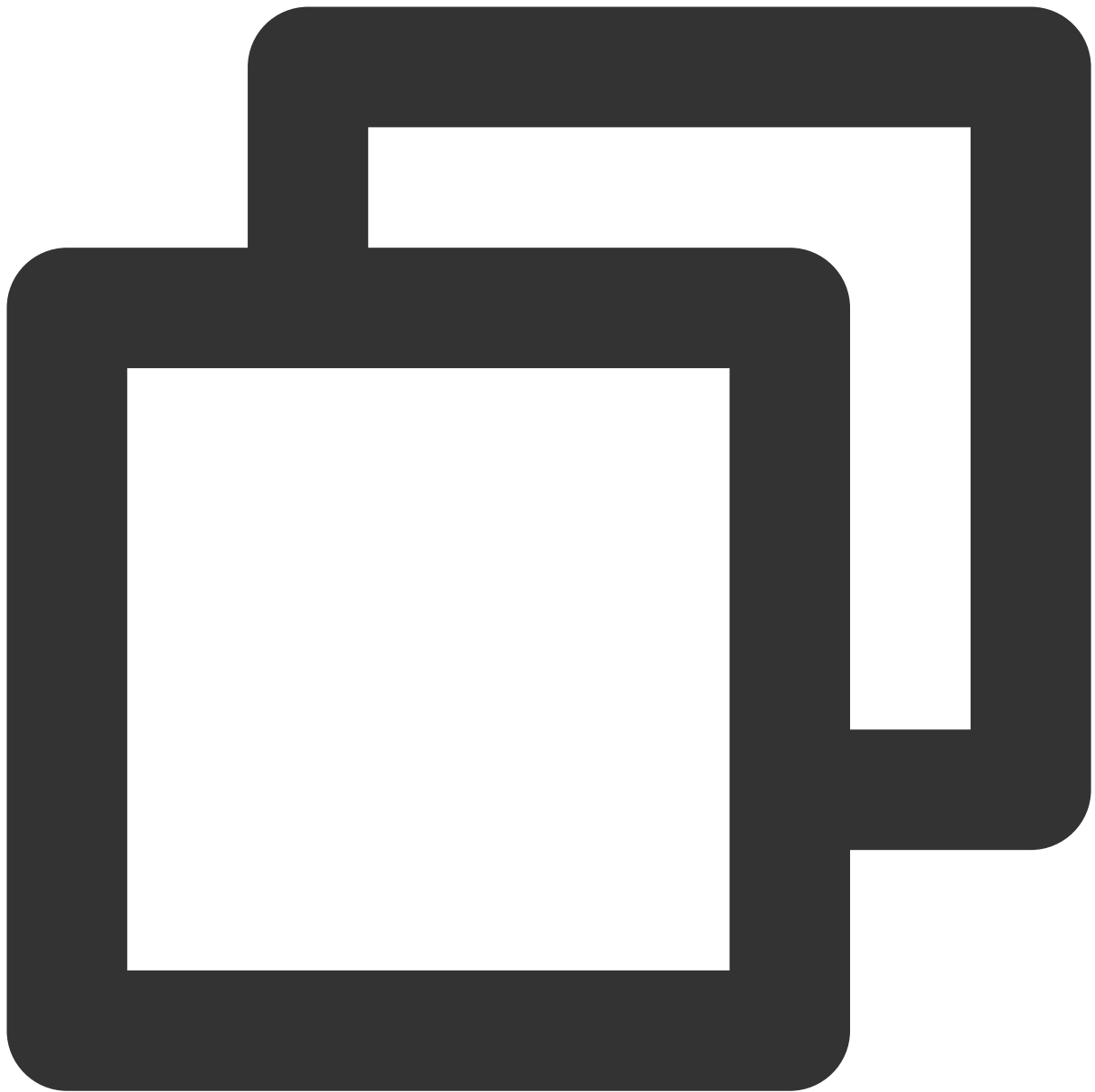
[Business System](#)[Application Settings](#)[Allowlist Management](#)

Business System: rum-36uJ0ycDx8qRVY.Rum-test ▼[Application Access](#)

Application Name	Report ID	Application ID
rum-android-demo	73129ae7-17456	500353
rum-ios-demo	3e3073c0-3712	500355

Total items: 2

2. In `AppDelegate.m` file, import the header files and add the following configuration:



```
#import <QAPM/QAPM.h>
#import <QAPM/QAPMLaunchProfile.h>
If it is a Swift project, import files to the corresponding bridging-header.h.
Initialize QAPM with the method of application:didFinishLaunchingWithOptions: in pr

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

#ifdef RELEASE
    if (level <= QAPMLogLevel_Event) { ///Log of release version
        NSLog(@"%@", [NSString stringWithUTF8String:log]);
    }
}
```



```
#endif

#ifdef GRAY
    if (level <= QAPMLogLevel_Info) { ///Log of grayscale version and release version
        NSLog(@"%@", [NSString stringWithUTF8String:log]);
    }
#endif

#ifdef DEBUG
    if (level <= QAPMLogLevel_Debug) { ///Log of internal version, grayscale version
        NSLog(@"%@", [NSString stringWithUTF8String:log]);
    }
#endif
}

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    //Initialize QAPM after agreeing to the corresponding privacy policy.
    [self setupQapm];
}

- (void)setupQapm {

    //Start logging begin timestamp, business logging on its own.
    [QAPMLaunchProfile setBeginTimestampForScene:@"finish"];

    [QAPM registerLogCallback:loggerFunc];
#ifdef DEBUG
    //Enable all QAPM monitoring features.
    [[QAPMModelStableConfig getInstance] setupModelAll:1];
#else
    [[QAPMModelStableConfig getInstance] setupModelStable:0.5];
#endif

    //Used to view the current SDK version information.
    NSLog(@"qapm sdk version : %@", [QAPM sdkVersion]);

    //Reporting address, fixed value. https://app.rumt-zh.com
    [QAPMConfig getInstance].host = @"https://app.rumt-zh.com";

    // Set user tags. The default value is 10,000. Provide userID to calculate user tags.
    [QAPMConfig getInstance].userId = @"Fill in the unique identifier of the user code";

    // Set device unique identifier. The default value is 10,000. Provide deviceID to calculate device tags.
    [QAPMConfig getInstance].deviceId = @"Fill in the unique identifier of the device";
    // Set App version number.
    [QAPMConfig getInstance].customerAppVersion = @"Fill in the business APP version";
    [QAPM startWithAppKey:@"Fill in the applied appkey correctly."];
}
```

```
//Logging begin timestamp completed, business logging on its own.  
[QAPMLaunchProfile setEndTimestampForScene:@"finish"];  
}
```

Note

1. The allowlist is divided into a user allowlist and a device allowlist. The userId or deviceId in the initial configuration code needs to be added to the allowlist through the **Mobile Performance Monitoring > Application Management** page, in order for mobile webviews, lag and stutter monitoring, frame drops, and network monitoring data reporting to be enabled.
2. For configuration of advanced features, see the [Demo project](#), as well as documents in the shell script and related document folders.

Note:

1. For version 5.2.8 and later ones, if the SDK does not detect an embedded.mobileprovision file, it will not output SDK logs for optimization purposes. During access testing, you can manually create a blank embedded.mobileprovision file (which needs to be introduced or created via XCode and is accessed by the SDK via pathForResource) to bypass the SDK check.
2. Monitoring for lag, stutter and frame drops is not available on emulators, therefore, real device testing is recommended.

Android Use Cases

Integration and Initialization

Last updated : 2024-04-25 11:42:15

Overview

This document describes how to integrate and initialize the Android SDK.

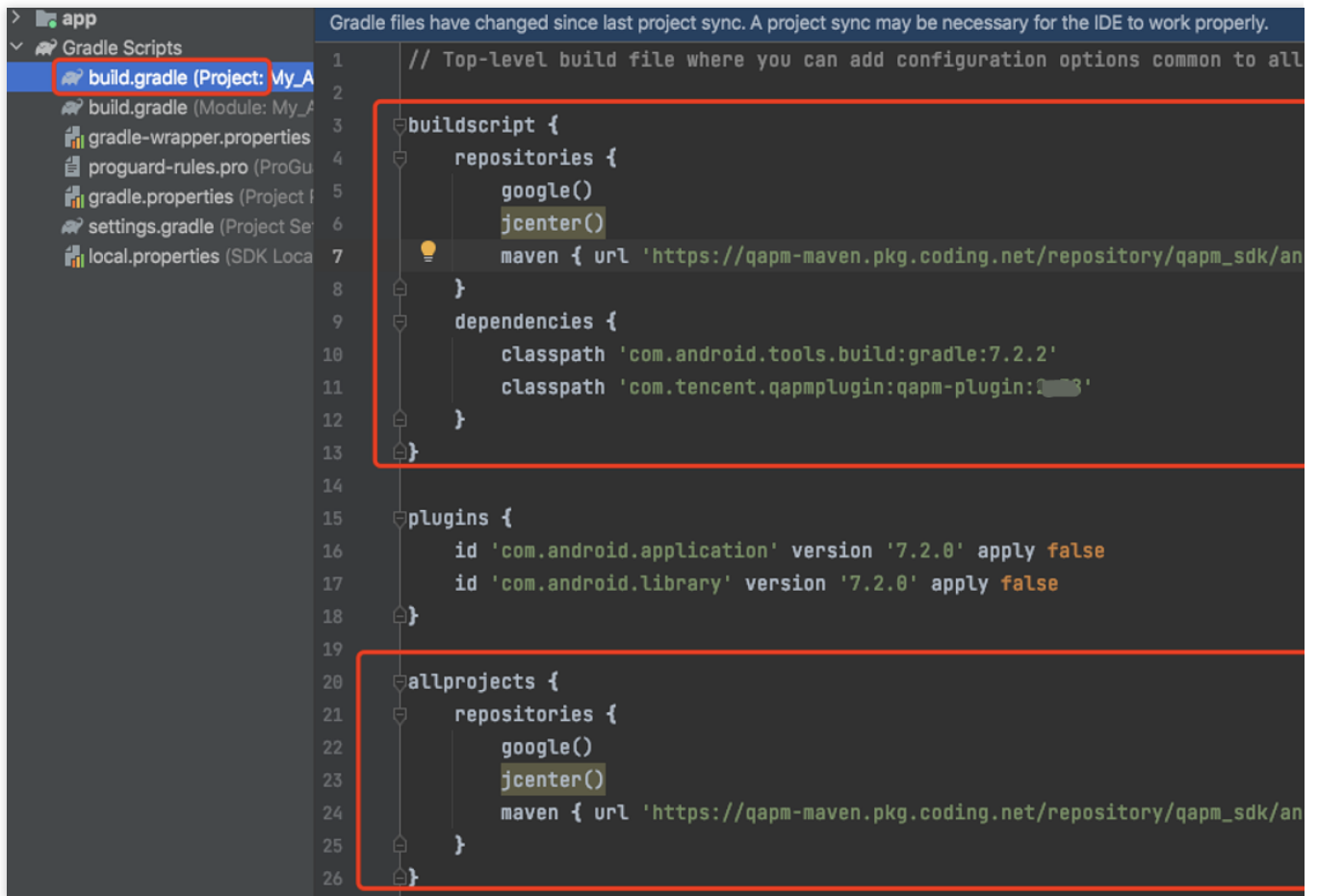
Directions

Step 1: Integration via Gradle

1. Add maven dependency to the project-level build.gradle.

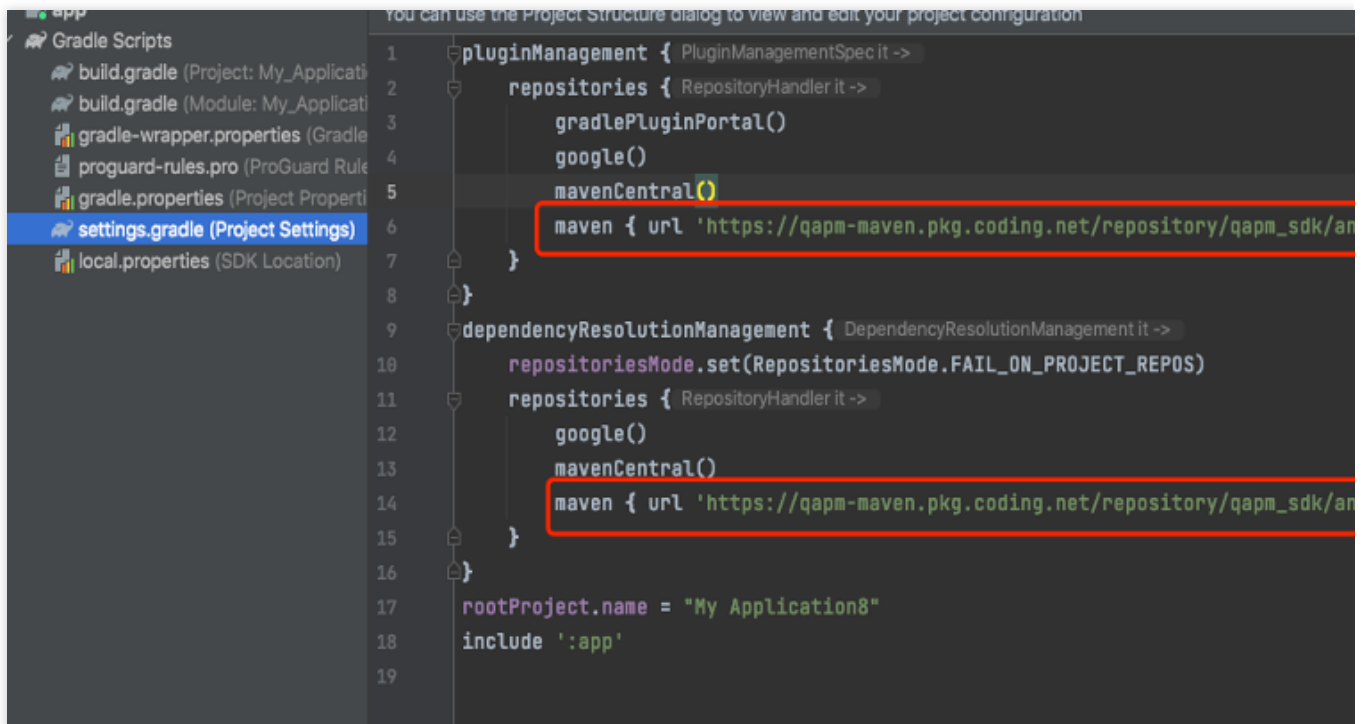
- i. Add buildscript and allprojects (not required for gradle 7.0 and later) .

For versions below gradle 7.0, apply the following configuration:



For gradle 7.0 and later, apply the following configuration:

For gradle 7.0 and later, allprojects are not supported. Maven dependency of allproject should be configured in settings.gradle, as shown below:



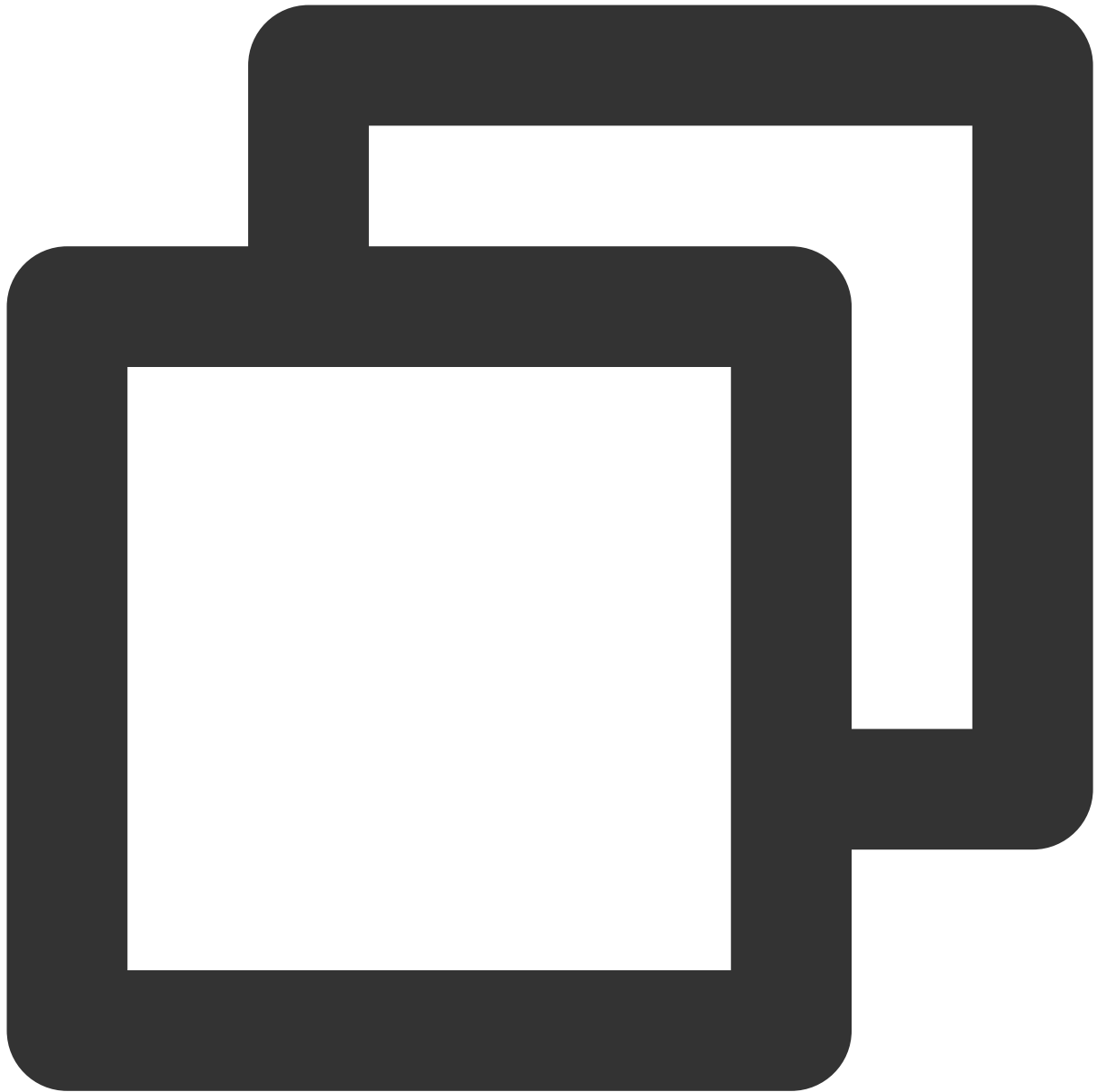
The screenshot shows an IDE window with the 'settings.gradle' file selected in the left sidebar. The main editor displays the following Kotlin code:

```
1  pluginManagement { PluginManagementSpec it ->
2      repositories { RepositoryHandler it ->
3          gradlePluginPortal()
4          google()
5          mavenCentral()
6          maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an
7      }
8  }
9  dependencyResolutionManagement { DependencyResolutionManagement it ->
10     repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
11     repositories { RepositoryHandler it ->
12         google()
13         mavenCentral()
14         maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an
15     }
16 }
17 rootProject.name = "My Application8"
18 include ':app'
19
```

Two red rectangular boxes highlight the Maven repository URLs in the code:

- Line 6: `maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an`
- Line 14: `maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/an`

Code:

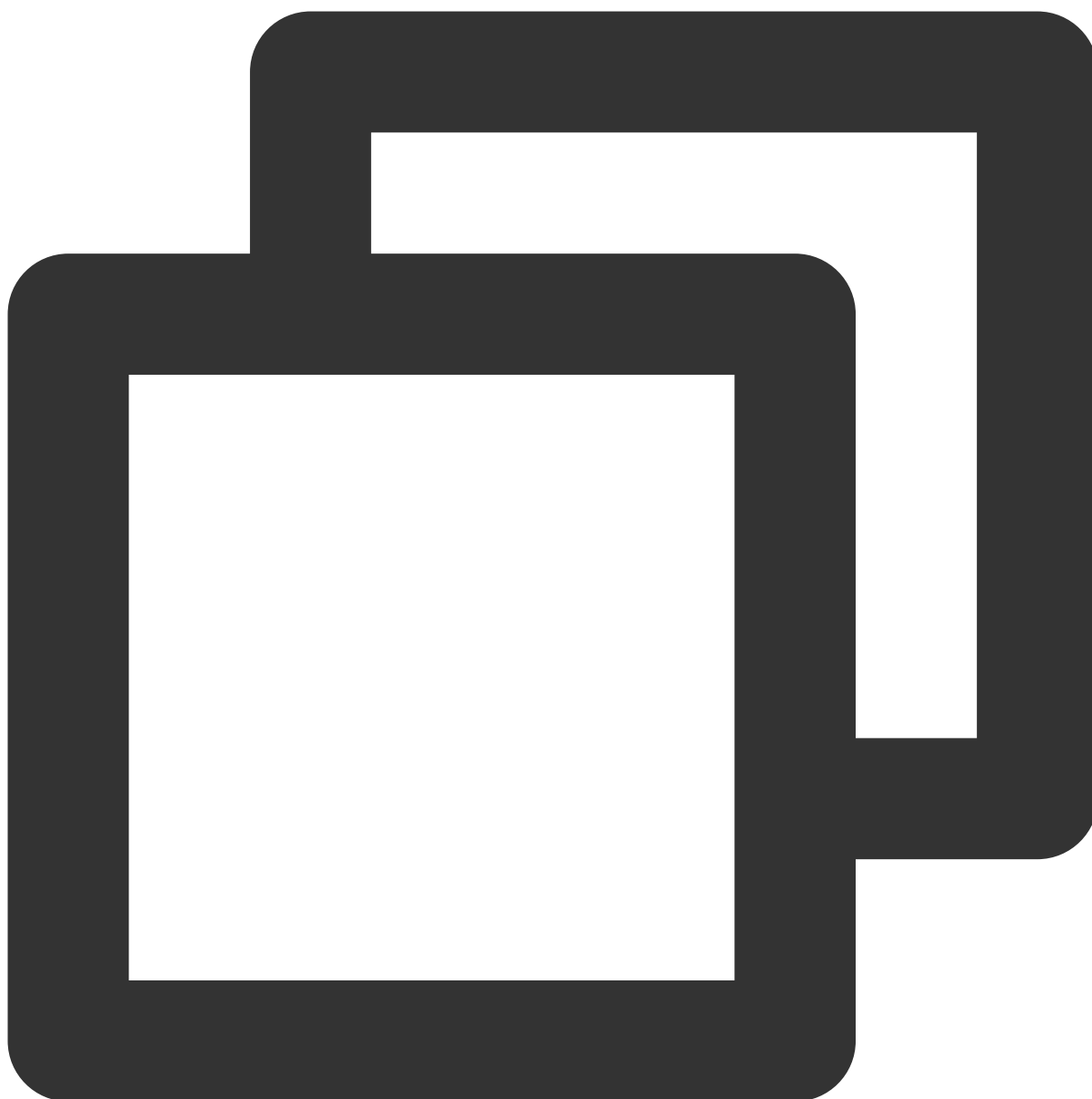


```
maven {url'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_release/'}
```

ii. For `com.android.tools.build:gradle:*.*.*` in the buildscript, fill in your gradle plugin version, as shown below:

```
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:3.0.3'  
    }  
}
```

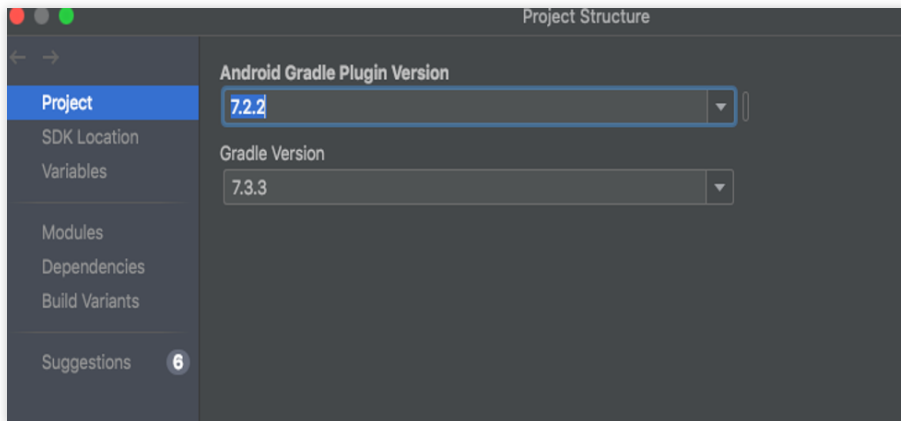
gradle plu



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

Note:

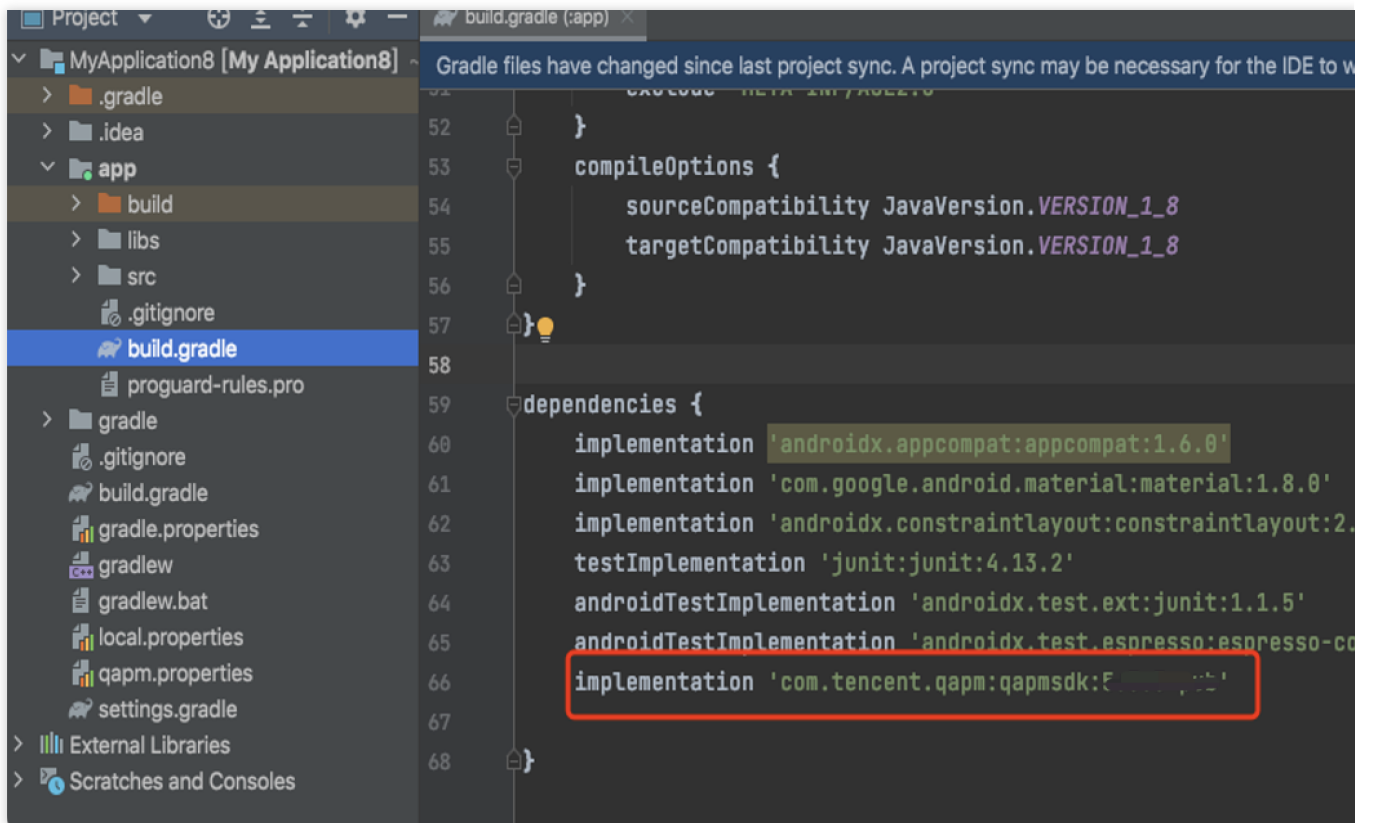
1. Check Gradle version and gradle plugin version in the menu file > project structure, as shown below:



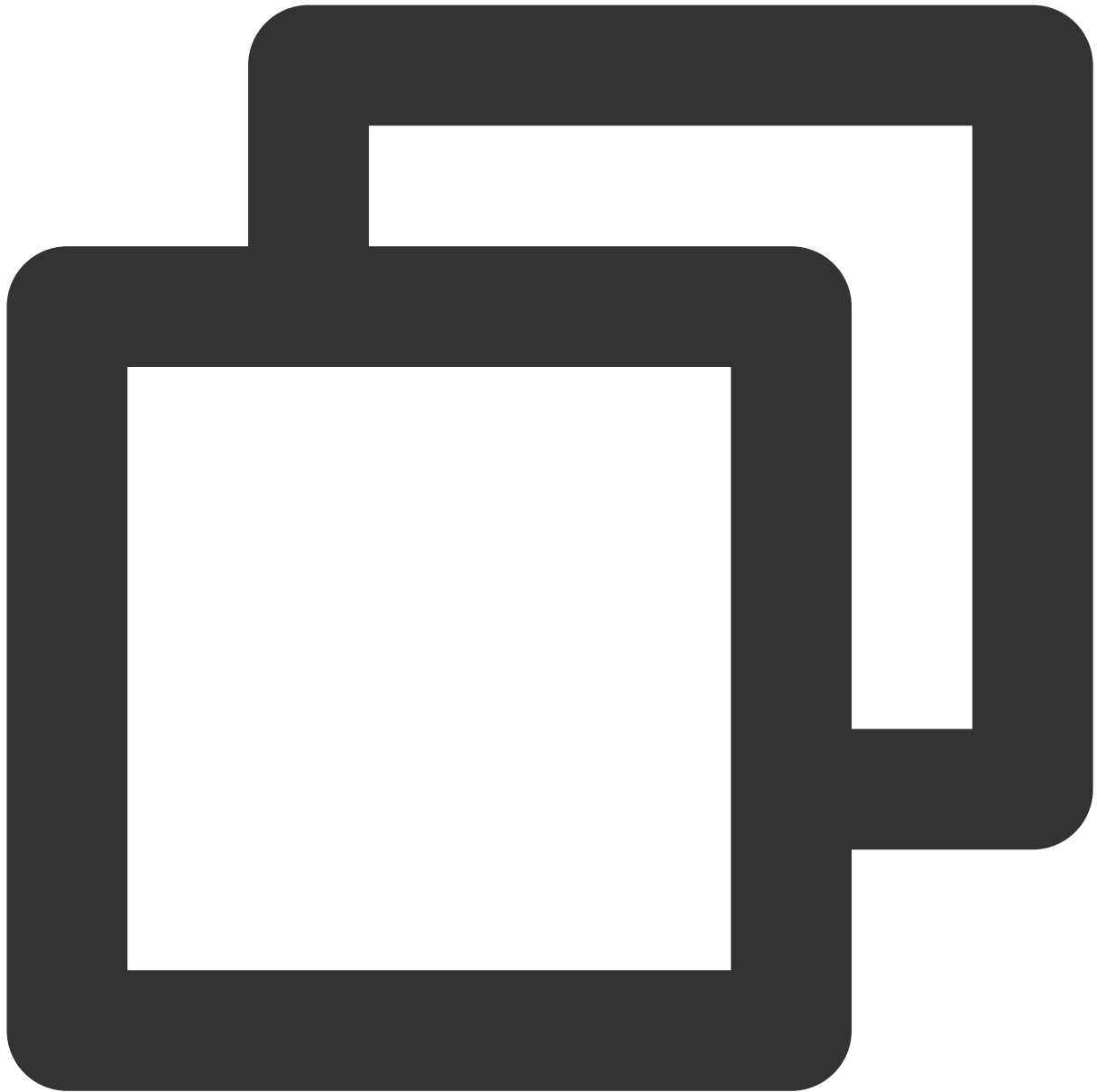
2. The corresponding relationship between Gradle and gradle plugin.

plugin version	minimum required Gradle version
8.1	8
8	8
7.4	7.5
7.3	7.4
7.2	7.3.3
7.1	7.2
7	7
4.20+	6.7.1

2. **Introduce the module to the App's build.gradle (for studio versions below 3.0, use the compile reference header)**.



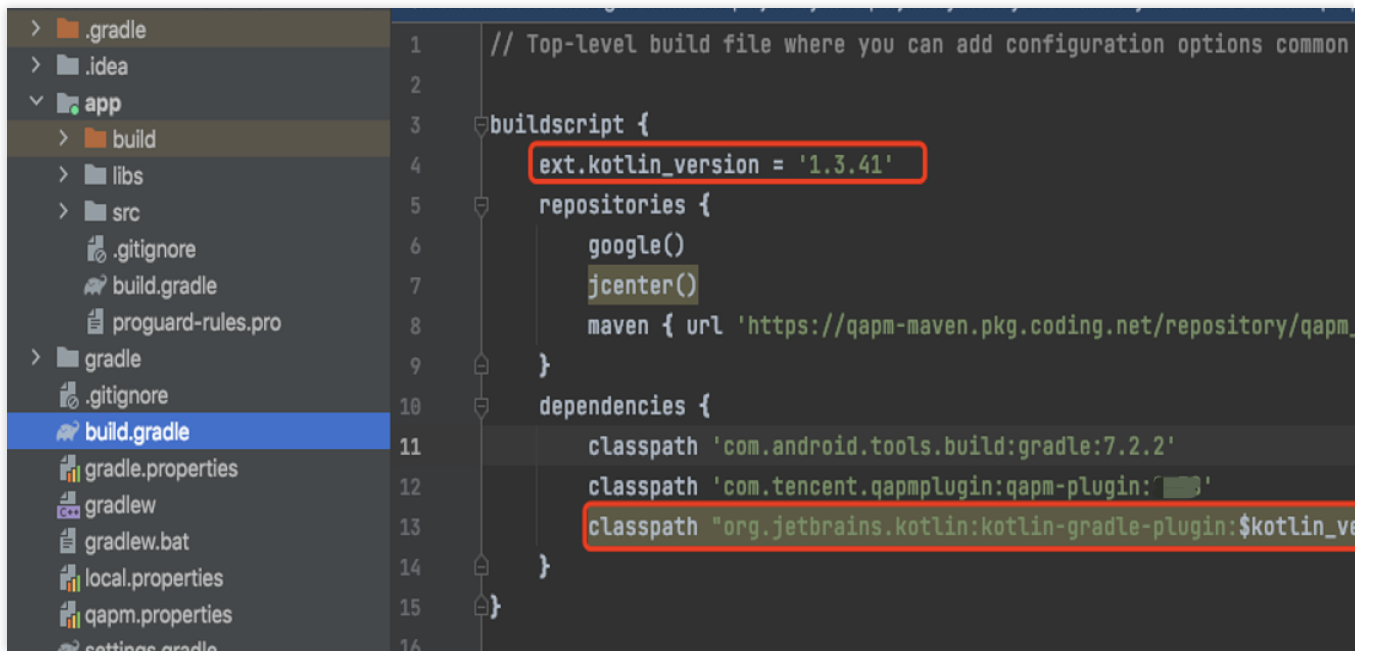
Code:



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

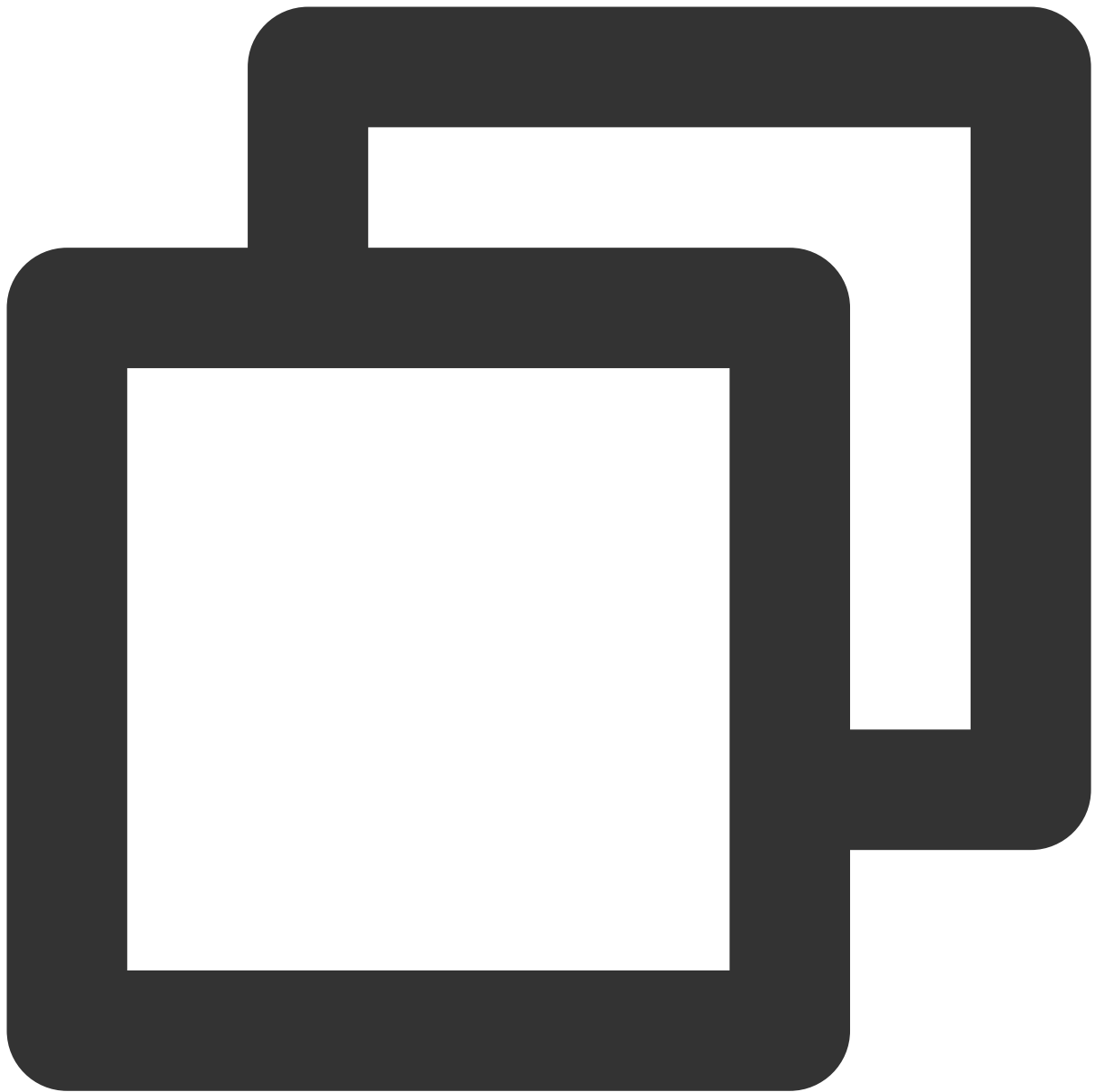
3. Introduce the kotlin dependency.

- i. Add the following code to the project-level build.gradle:



```
1 // Top-level build file where you can add configuration options common
2
3 buildscript {
4     ext.kotlin_version = '1.3.41'
5     repositories {
6         google()
7         jcenter()
8         maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm'
9     }
10 }
11 dependencies {
12     classpath 'com.android.tools.build:gradle:7.2.2'
13     classpath 'com.tencent.qapmplugin:qapm-plugin:1.3.41'
14     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
15 }
16
```

Code:



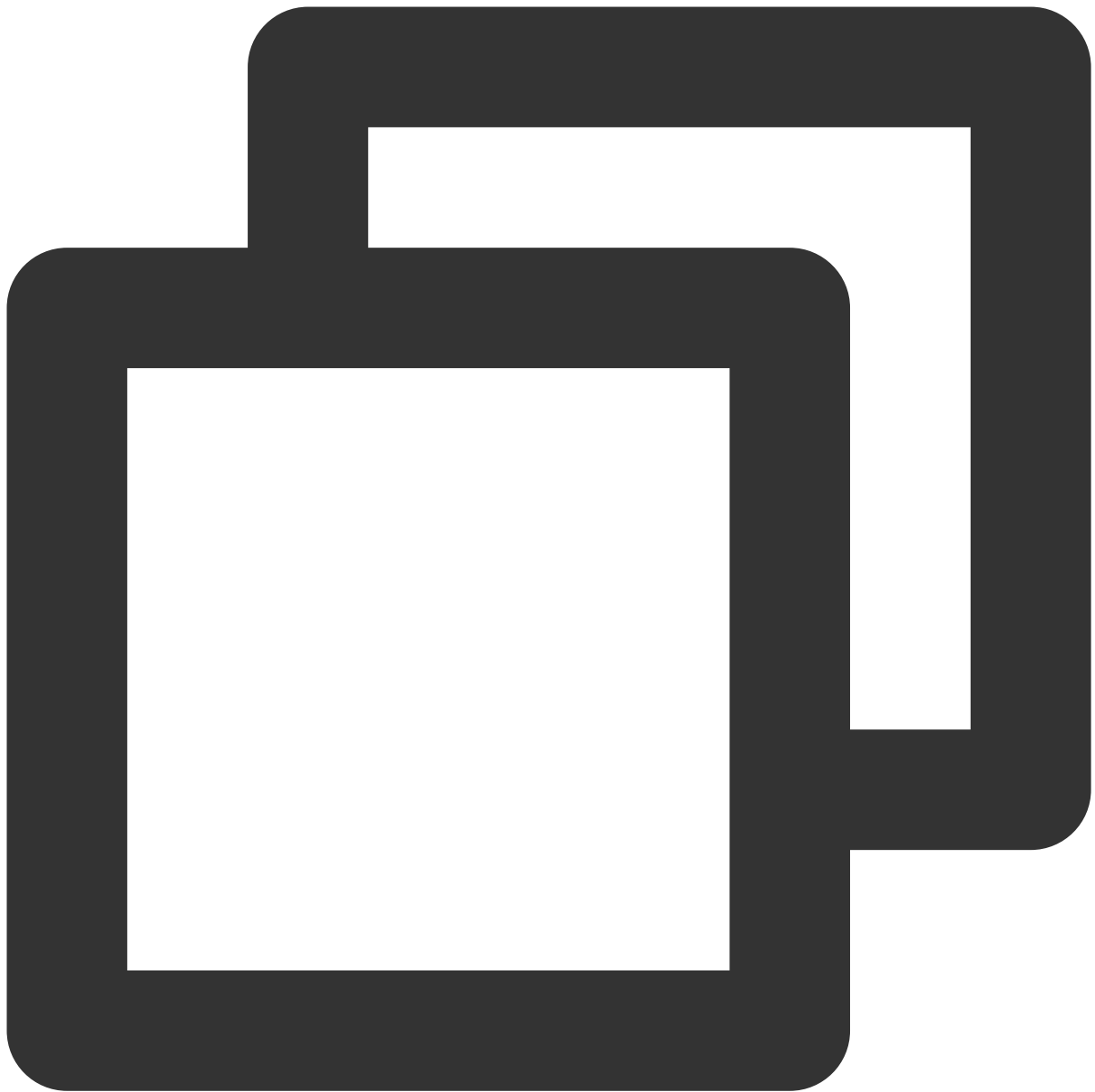
```
ext.kotlin_version = '1.3.41'  
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
```

ii. Add the following code to the App's build.gradle:

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

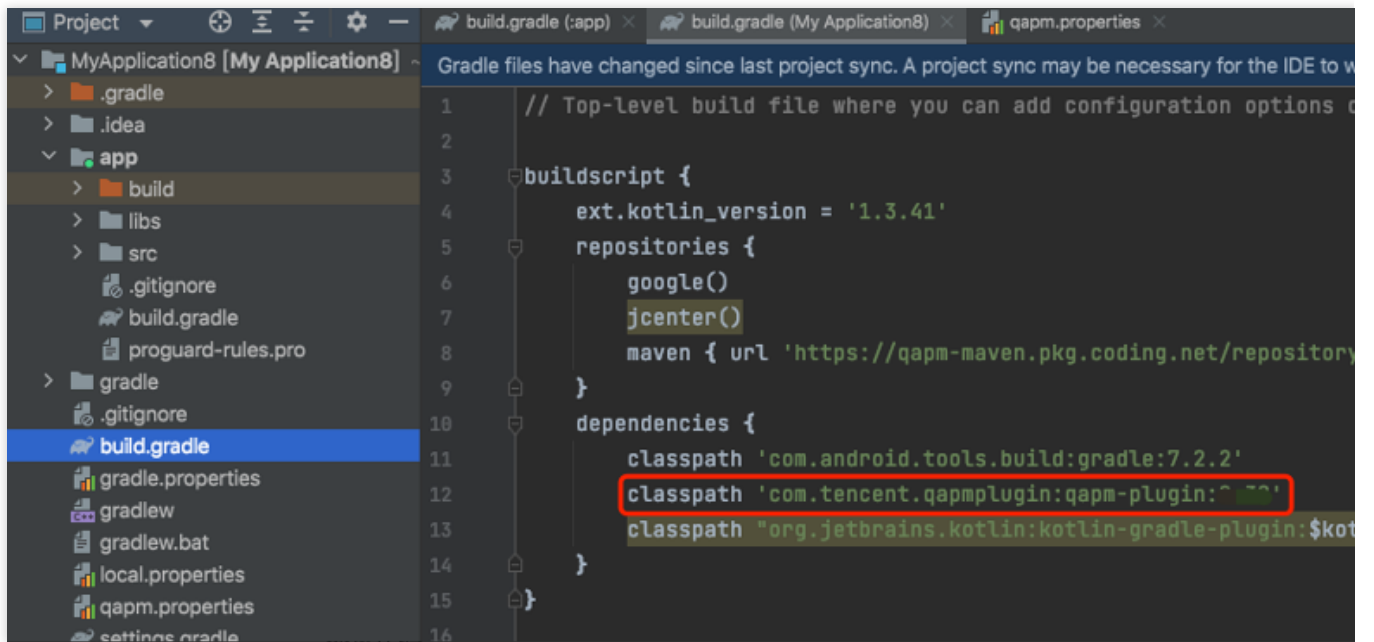
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.sdkapp"
        minSdkVersion 16
        targetSdkVersion 26
    }
}
```

Code:

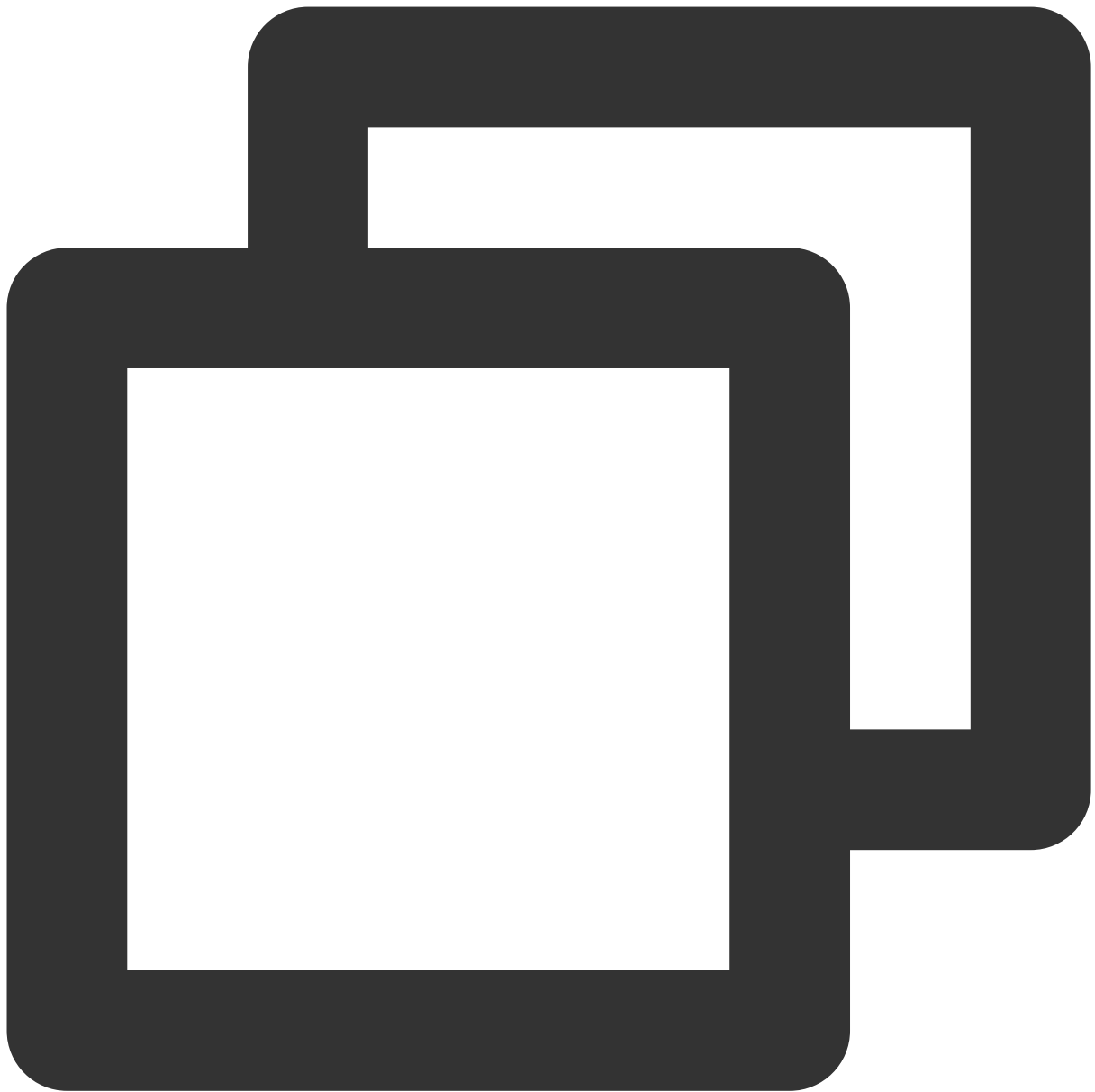


```
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-android-extensions'
```

4. Add the following configuration to the project-level build.gradle file.



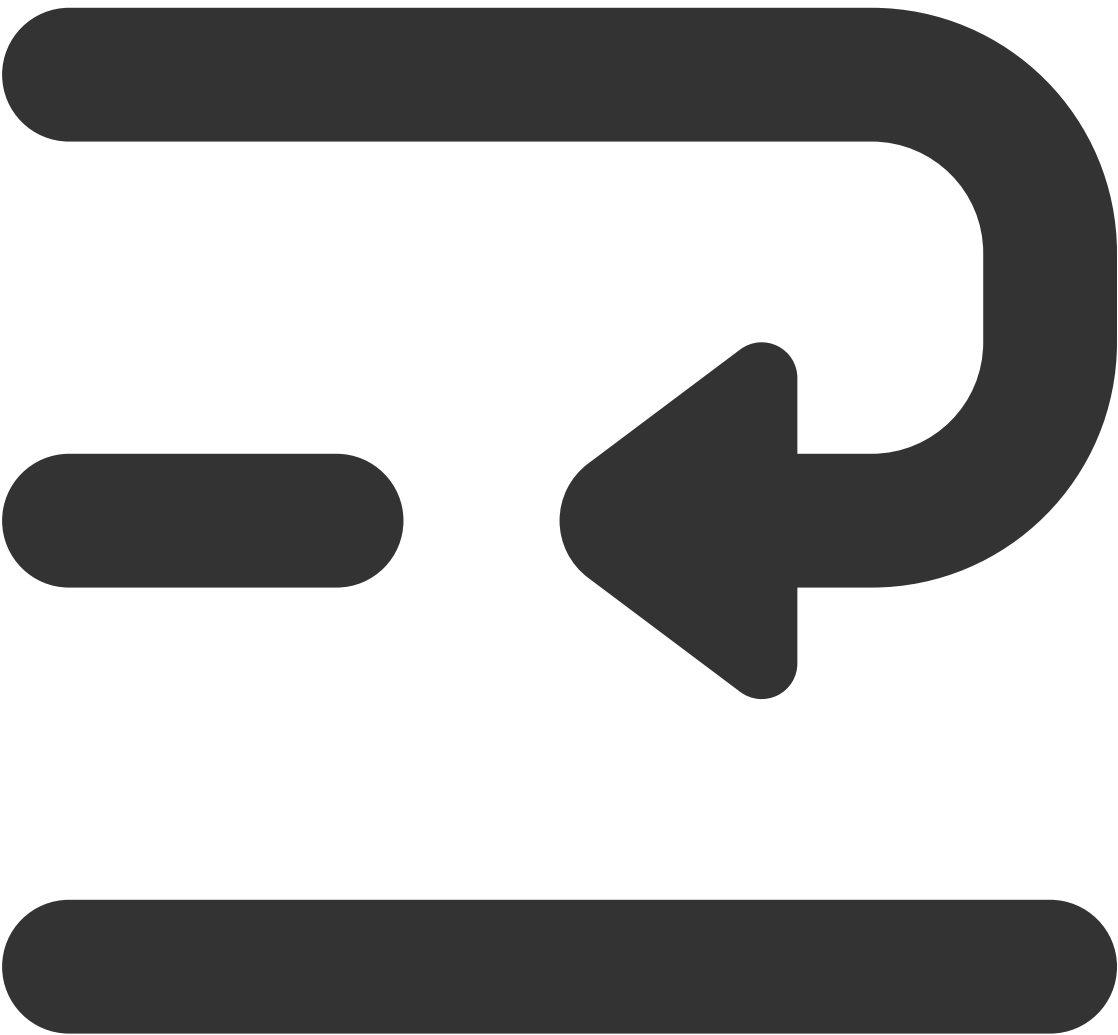
Code:

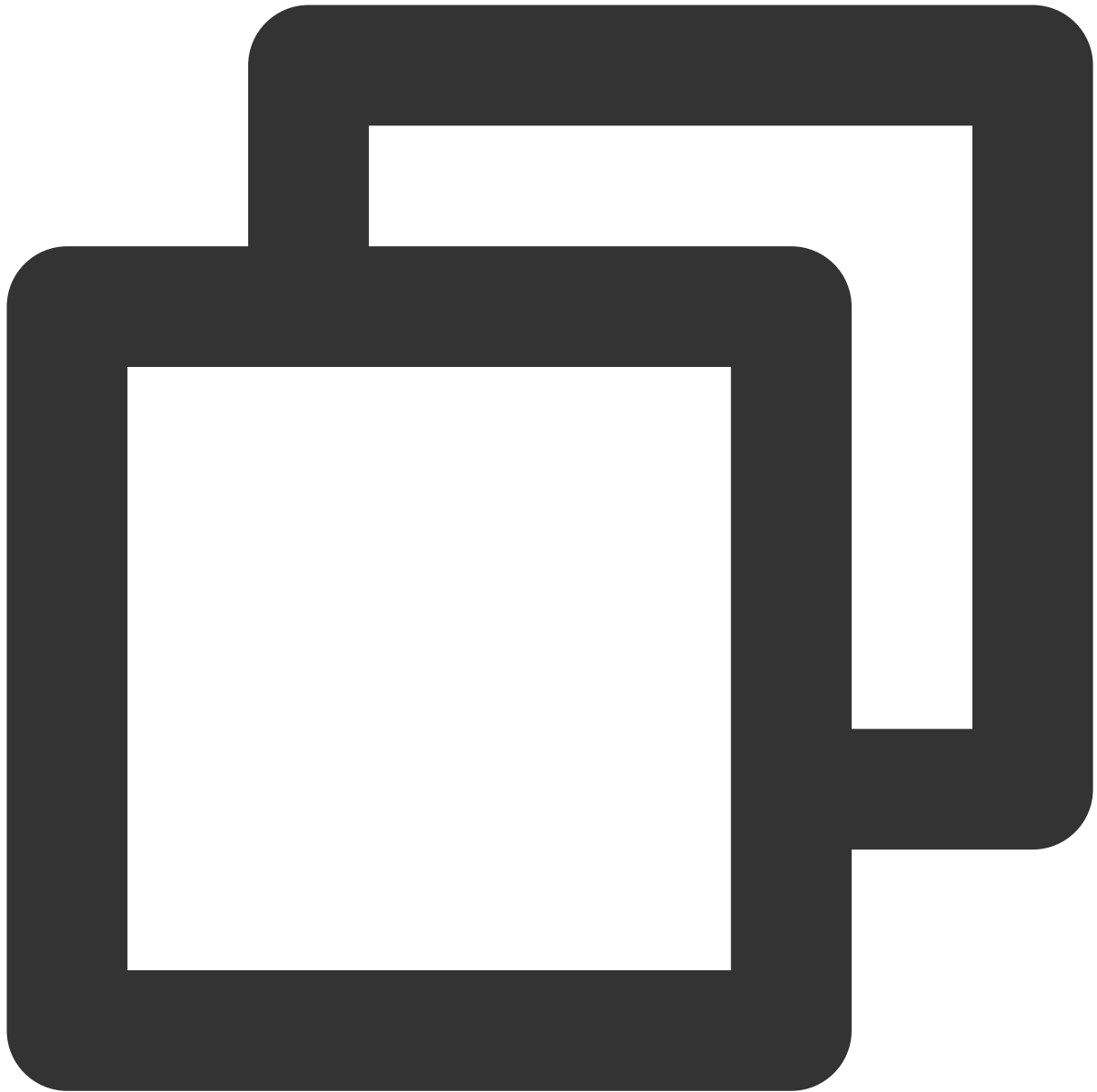


```
classpath 'com.tencent.qapmplugin:qapm-plugin:2.39'
```

5. Add the following configuration to the App's build.gradle file.

Code:





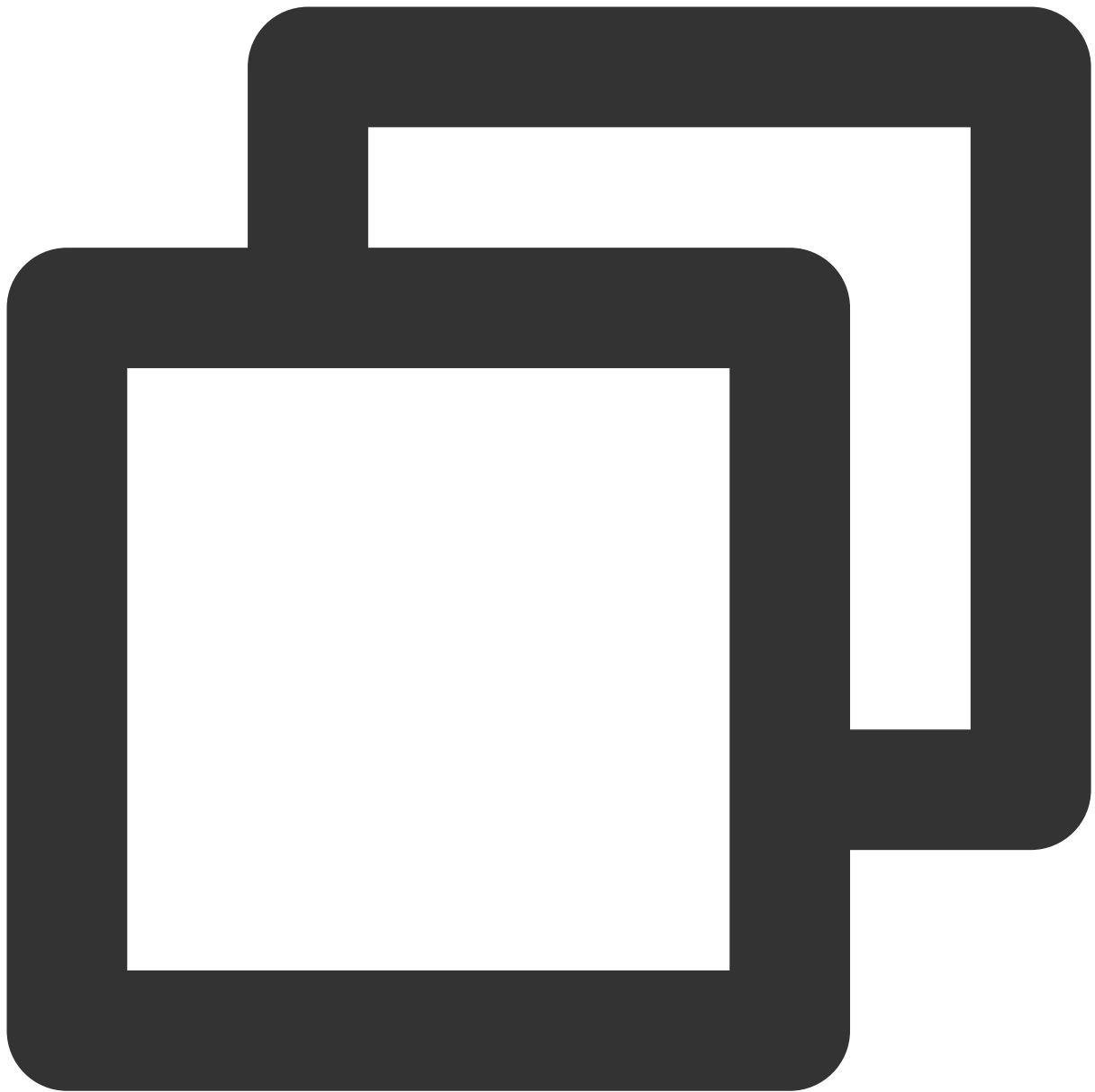
```
apply plugin: 'qapm-plugin'
QAPMPluginConfig{
    // Optional, default is null. Enter attachBaseContext in the class where Applicat
}
```

Note:

If items 4 and 5 are not configured, you may fail in startup, network monitoring, and automatic symbol table uploading.

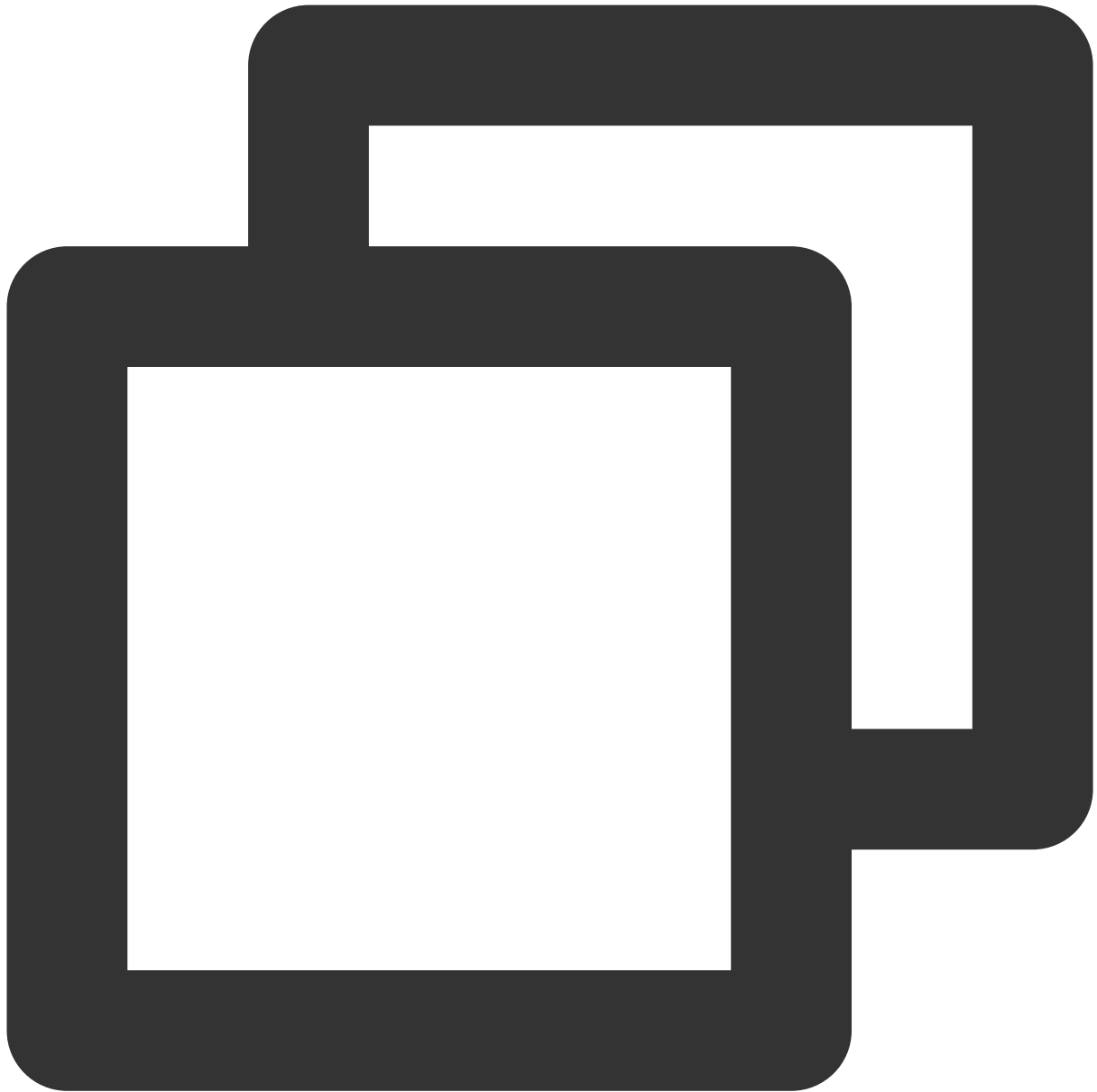
Step 2: Parameter Configuration

1. Add the following permissions in AndroidManifest.xml.



```
<!--Required for information submission -->
<uses-permission android:name="android.permission.INTERNET" />
<!--Required for information collection-->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

2. To avoid the SDK obfuscation, add the following configuration in the App's proguard-rules.pro file:



```
-keep class com.tencent.qapmsdk.**{*;}  
# If you need network monitoring, ensure okhttp3 is not obfuscated.  
-keep class okhttp3.**{*};
```

Step 3: Initializing the SDK



1. Log in to [TCOP](#) console, choose the **Mobile Performance Monitoring** page from the left sidebar, and click **Application Management > Application Settings**, to obtain Appkey (reporting ID).

Application Management

[Business System](#)[Application Settings](#)[Allowlist Management](#)

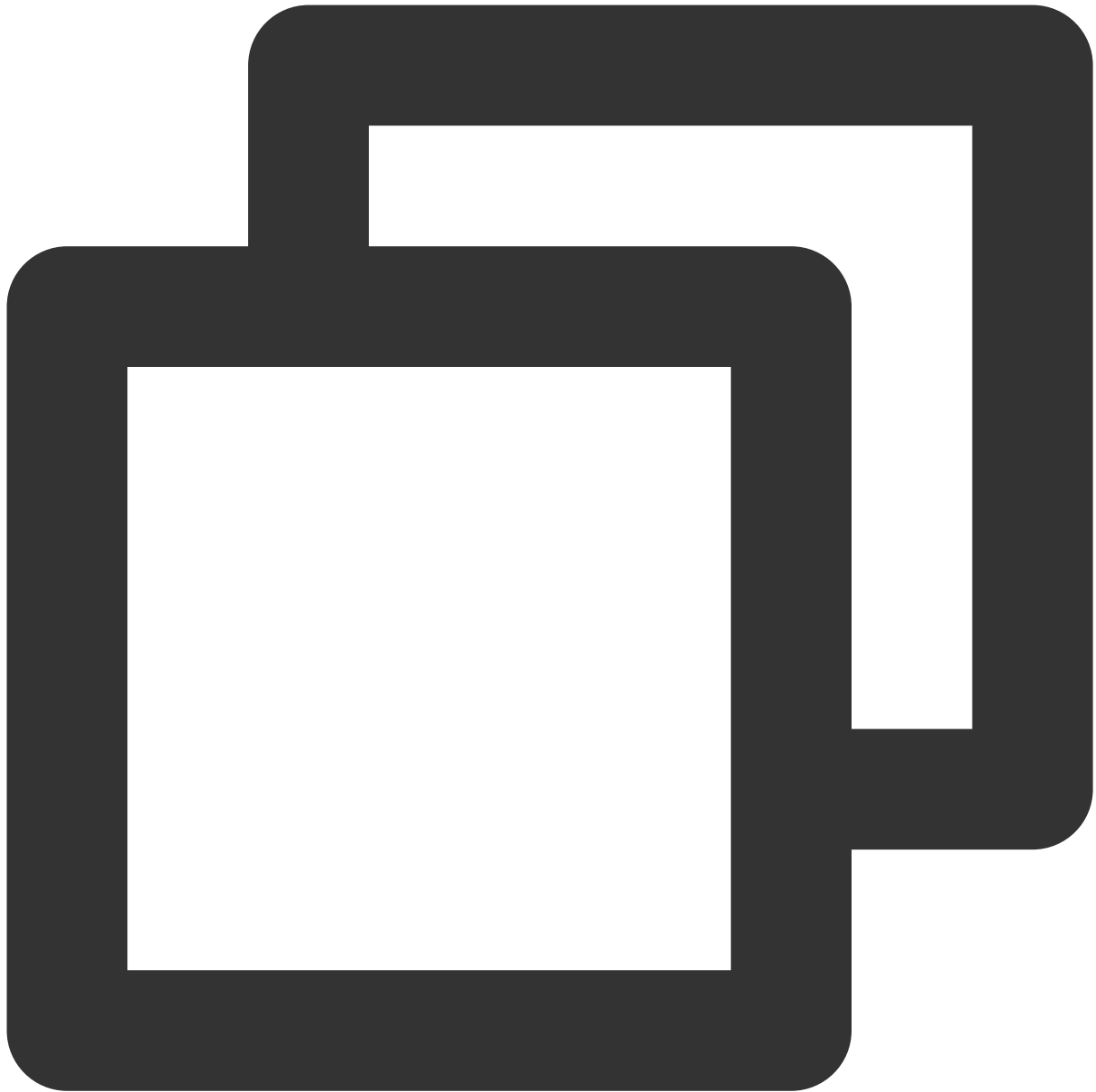
Business System: rum-36uJ0ycDx8qRVY.Rum-test ▼

[Application Access](#)

Application Name	Report ID	Application ID
rum-android-demo	73129ae7-17456 	500353
rum-ios-demo	3e3073c0-3712 	500355

Total items: 2

2. Copy the code below and modify some fields. The following items are required API settings. For configurations of other APIs, see the initialization API analysis. (It is recommended to initialize QAPM in the Application.)



```
// Set the mobile phone model and device ID.  
// Provide the device's identifier. Any character string can be used. DeviceId (req  
// deviceId can be used to enable the allowlist, preventing data from being sampled  
QAPM.setProperty(QAPM.PropertyKeyDeviceId, "device identifier");  
// Provide the mobile phone model (required).  
QAPM.setProperty(QAPM.PropertyKeyModel, "Fill in the mobile phone model");  
  
// Set Application (required).  
QAPM.setProperty(QAPM.PropertyKeyAppInstance, getApplication());  
// Set AppKey (required, to distinguish the reported product, this value is obtaine  
QAPM.setProperty(QAPM.PropertyKeyAppId, "YourAppKey");
```

```
// Set Product Version, for backend search fields (required).
QAPM.setProperty(QAPM.PropertyKeyAppVersion, "YourApp Version");
// Set UUID, to pull the mapping of obfuscated stack traces (required, if you use t
// This variable will be generated during build process. Ignore when it is red.
QAPM.setProperty(QAPM.PropertyKeySymbolId, BuildConfig.QAPM_UUID);
// Set User ID (userId), for backend to search fields (required).
// userId can be used to enable the allowlist, preventing data from being sampled a
QAPM.setProperty(QAPM.PropertyKeyUserId, "123456");
// Set Log level (optional). For online version, set to QAPM.LevelOff or QAPM.Level
QAPM.setProperty(QAPM.PropertyKeyLogLevel, QAPM.LevelInfo);
// Set QAPM's reporting domain on public network (required). Fixed value: https://a
QAPM.setProperty(QAPM.PropertyKeyHost, "https://app.rumt-zh.com");
// Enable QAPM.
QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeStable);
```

Note:

For AppKey, see the previous step, check it on the **Mobile Performance Monitoring > Application Management** page.

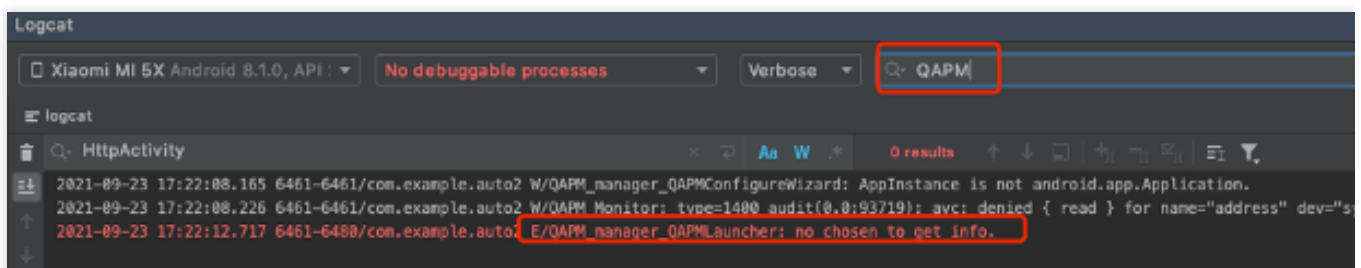
Crash and launch data are fully reported. Other data, due to the large amount, are sampled and reported at a rate of 0.1% (one-thousandth). To fully report the data, enable the allowlist, and the app will change the sampling rate upon the next launch.

Add the configured userId or deviceId to the allowlist through the **Mobile Performance Monitoring > Application Management** page, and enable the allowlist.

For multiple processes, initialize QAPM them individually.

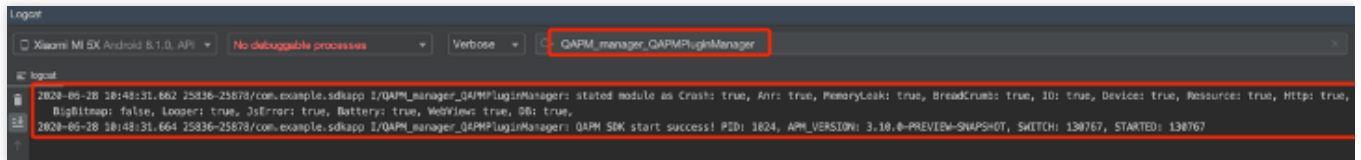
Step 4: Access Verification

1. If the following log is printed, it indicates that the user was not sampled, and sampling rate should be reconfigured.



TAG: QAPM_manager_QAPMLauncher.

2. If the following log is printed, it indicates preliminary access is successful, and you can verify data reporting/try enabling advanced features.



TAG: QAPM_manager_QAPMPluginManager.

Initialization API Analysis

1. API name: public static QAPM setProperty(int key, String value)

Purpose: Setting related parameters for QAPM

Parameter: key - The Key to set

Optional:

QAPM.PropertyKeyLogLevel: to enable log level (It is recommended to enable QAPM.LevelDebug for Debug versions and QAPM.LevelWarn for release versions).

QAPM.PropertyNeedTranslate: whether to translate the stack, defaulted to yes. If the apk has no obfuscation, set it to false; otherwise the front end may display everything as unTranslated.

2. public static boolean beginScene(String sceneName, int mode)

Purpose: To enable monitoring

Parameters: sceneName-Scene name, mode-Enabled feature.

Optional:

QAPM.ModeAll: To enable all features (It is recommended during the development process. Features include interval performance, crash, ANR, webview (page loading, JsError, network), and HTTP network)

QAPM.ModeStable: To enable some features (it is recommended for release versions. Features include interval performance, crash, ANR, webview (page loading, JsError, network), and HTTP network)

QAPM.ModeWebView: To enable WebView Page Load monitoring

QAPM.ModeJsError: To enable WebView JS Error monitoring

QAPM.ModeHTTPInWeb: To enable WebView network monitoring

QAPM.ModeHTTP: Enable network monitoring

Note:

1. It is recommended to enable QAPM.ModeStable for official versions and QAPM.ModeAll for development versions.

2. By default, ModeStable feature is enabled. You can use the OR operation based on ModeStable to add features you need. For example, to enable Stable and memory ceiling: beginScene("Stable&Ceiling", QAPM.ModeStable | QAPM.ModeCeiling). You can use the XOR operator to exclude unwanted features. For example, disable the network: QAPM.ModeStable^QAPM.ModeHTTP.

3. public static boolean endScene(String sceneName, long mode)

Purpose: To end monitoring (only effective for frame drops and interval performance collection)

Parameters: sceneName-Scene name need to be closed (it should correspond to the one used in beginScene)

Optional:

QAPM.ModeDropFrame: Turn off frame drop monitoring

QAPM.ModeResource: Turn off interval performance monitoring

Others

Note:

When compiling and packaging the App through the qapm plugin, the APP needs a uuid as the build ID. If a qapm.properties file exists in the project directory and the value of the qapm_uuid attribute in the file exists, that value will be used as the build ID. Otherwise, the plugin will generate a build ID at random.

For qapm-plugin version 2.39 and earlier ones, it will report an IO Error during the App compilation process:

java.io.FileNotFoundException, qapm.properties (No such file or directory).

```
java.io.FileNotFoundException Create breakpoint : , /qapm.properties (No su
  at java.base/java.io.FileInputStream.open0(Native Method)
  at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
  at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
  at com.tencent.qapm.QAPMTransformerTask.loadBuildId(QAPMTransformerTask.java:201)
  at com.tencent.qapm.QAPMTransformerTask.beforeTransform(QAPMTransformerTask.java:147)
  at com.tencent.qapm.QAPMTransformerTask.transform(QAPMTransformerTask.java:91)
  at com.android.build.gradle.internal.pipeline.TransformTask$2.call(TransformTask.java:281)
  at com.android.build.gradle.internal.profile.NoOpAnalyticsService.recordBlock(NoOpAnalyticsService.kt:72)
  at com.android.build.gradle.internal.pipeline.TransformTask.transform(TransformTask.java:239) <61 internal l
  at java.base/java.util.Optional.orElseGet(Optional.java:369) <13 internal lines>
  at java.base/java.util.Optional.orElseGet(Optional.java:369) <49 internal lines>
```

This error only occurs during the compilation process and can be ignored, which will not affect the App's runtime.