云开发 CloudBase

开发指南

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

开发指南
身份认证
概述

最近更新时间：2024-06-11 15:03:31

CloudBase 提供跨平台的登录鉴权功能，您可以基于此为自己的应用构建用户体系，包括但不限于：

- 为用户分配全局唯一的身份标识 uid。
- 储存和管理用户个人信息。
- 关联多种登录方式。
- 管理用户对数据、资源的访问权限。
- 用户行为的收集和分析。

同时，CloudBase 登录鉴权是**保护您的服务资源**的重要手段，CloudBase 对用户端发来的每一个请求，都会进行身份和权限的检查，避免您的资源被恶意攻击者消耗或者盗用。

登录鉴权方式

云开发 CloudBase 提供以下登录鉴权方式供不同的用户场景使用：

登录类型	场景
匿名登录	用户以临时的匿名身份登录云开发，无需注册。
邮箱登录	用户使用自己的邮箱+密码登录。
微信授权登录	• 经**微信公众平台**授权的**公众号网页**。 • 经**微信开放平台**授权的**网站**。
自定义登录	开发者可以完全接管登录流程，例如与自有的账号体系打通、自定义登录逻辑等。
用户名密码登录	用户使用自己的用户名+密码登录。
微信小程序登录	已开通云开发的微信小程序初始化后便同步完成登录鉴权，无需额外操作。
短信验证码登录	用户使用自己的手机号+验证码登录。

CloudBase 用户

每个登录 CloudBase 的用户，都有一个对应的 CloudBase 账号，用户通过此账号访问调用 CloudBase 的数据与资源。

信息项	说明
UID	每个账号都有全局唯一的 UID，即账号 ID，作为用户的唯一身份标识。
用户信息	每个账号可以添加、修改用户信息，请参见 管理用户 。
登录方式	每个账号除了最初的登录方式之外，还可以关联其它登录方式，请参见 账户关联 。

登录状态的持久化

您可以指定登录状态如何持久保留。默认为 `local`，相关选项包括：

值	说明
`session`	在 SessionStorage 中保留登录状态，当前页面关闭后会被清除。
`local`	在本地存储中长期地保留登录状态。
`none`	在内存中保留登录状态，当前页面刷新、重定向之后会被清除。

例如，对于网页应用，建议选择是 `local`，即在用户关闭浏览器之后仍保留该用户的会话。这样，用户不需要每次访问该网页时重复登录，避免给用户带来诸多不便体验。

> **说明：**
> 匿名登录 的持久化类型强制设为 `local`，外部设置的值会被忽略。

访问令牌与刷新令牌

- **访问令牌：**用户登录 CloudBase 之后，会获得**访问令牌（Access Token）**作为访问 CloudBase 的凭证，访问令牌**默认具有两小时有效期**。
- **刷新令牌：**登录时会获得**刷新令牌（Refresh Token）**，**默认有效期 30 天**，用于访问令牌过期后，获取新的访问令牌。

> **说明：**
> - CloudBase 用户端 SDK 会自动维护令牌的刷新和有效期，开发者无需特别关注此流程。
> - 匿名登录 的刷新令牌（Refresh Token）会在到期后自动续期，以实现长期的匿名登录状态。

匿名登录

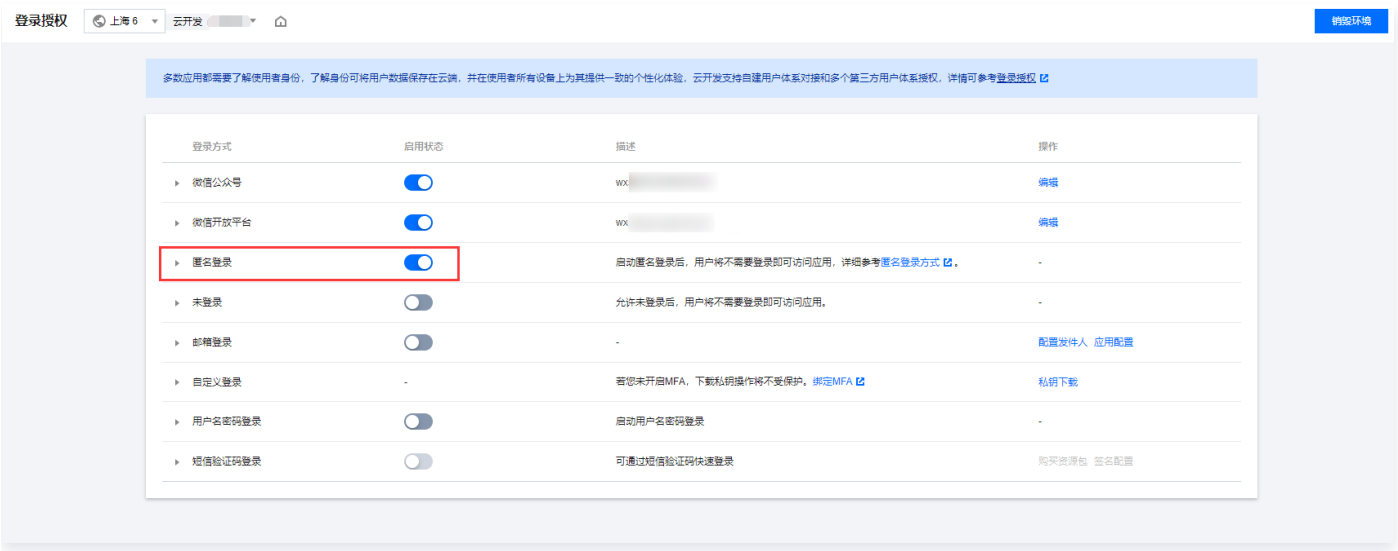
最近更新时间：2024-08-13 16:24:11

CloudBase 匿名登录的所有逻辑均可由客户端代码主动执行，无需用户手动操作。在匿名登录状态下可正常的调用 CloudBase 的资源，开发者同时可以配合安全规则针对匿名用户制定对应的访问限制。

开通流程

步骤1：开启匿名登录

登录 腾讯云 CloudBase 控制台，在 登录授权 页面中，将**匿名登录**一栏打开或关闭。



步骤2：添加安全域名（可选）

Web 应用需要将域名添加到 CloudBase 控制台的 Web 安全域名 列表中，否则将被识别为非法来源。

登录流程

在项目中引入 SDK 并初始化，调用 匿名登录 接口进行登录认证，示例代码如下：

```
import cloudbase from '@cloudbase/js-sdk';

// 初始化 SDK
const app = cloudbase.init({
  env: 'xxxx-yyy'; // 填入您的环境 ID
});

const auth = app.auth();

async function login(){
  // 调用匿名登录接口
  await auth.anonymousAuthProvider().signIn();
  // 匿名登录成功后，登录状态isAnonymous字段值为true
  const loginState = await auth.getLoginState();
  console.log(loginState.isAnonymousAuth); // true
```

```
}

login();
```

数量限制

每个 CloudBase 环境的匿名用户数量**不超过 1000 万个**。

安全规则

匿名用户在安全规则中的 `auth.loginType` 值为 `ANONYMOUS`，配合安全规则可以限制匿名用户的 云数据库 和 云存储 的访问权限。例如下述代码展示的安全规则：

云数据库

云数据库匿名用户不可读写。

```
{
  "read": "auth.loginType != 'ANONYMOUS'",
  "write": "auth.loginType != 'ANONYMOUS'"
}
```

云存储

云存储所有用户可读，匿名用户不可写。

```
{
  "read": "auth != null",
  "write": "auth.loginType != 'ANONYMOUS'"
}
```

> **说明**
> 详细请参见 安全规则-用户身份认证 。

转化为正式用户

如果用户在匿名状态下产生了一些私有数据（例如游戏中获取了个人成就和装备），想将此匿名账号转化为正式账号长久持有。针对这种需求，您可以 将匿名账号与任意一种登录方式关联，关联后，便可以永久使用该种登录方式登录 CloudBase，达成**匿名账号转正**的效果。详情请参见 账户关联 。

常见问题

匿名登录与未登录有什么区别?

- 从 C 端用户的角度来讲：
  - 匿名登录和未登录在使用上没有任何区别，都无需注册。
  - 匿名登录用户有独立的用户标识，在同设备有效期内，用户可以产生独立的私有数据。
  - 与未登录相比，匿名登录可以转为正式用户，匿名登录期间的私有数据会自动继承到正式用户名下。
- 从应用开发者的角度来讲：
  - CloudBase 匿名登录产生的匿名用户本质上是一个有效用户，拥有唯一的用户 ID。从而可以为其创建私有的 云数据库 和 云存储 数据，以及配合 安全规则 制定个性化的访问策略。
  - 未登录模式是纯粹的无登录态访问，该模式下的访问都不会进入用户的追踪统计。
  - 未登录的用户默认权限下无法使用任何 CloudBase 的服务和资源，而匿名登录在基础权限下也可以进行对应的资源读写，也可以结合安全规则实现更细粒度的管控。

匿名登录的用户达到上限后怎么办？

CloudBase 限制每个环境的匿名用户数量不超过 1000 万个，如果达到上限可以在 CloudBase 控制台 的“用户管理”页面查看匿名用户的活跃情况，针对长期不登录的匿名用户可以考虑将其删除以释放空间。

匿名用户是否会过期？

CloudBase 对匿名用户的有效期限策略是：每个设备同时只存在一个匿名用户，并且此用户永不过期。当然，如果用户手动清除了设备或浏览器的本地数据，那么匿名用户的数据便会被同步清除，再次调用 CloudBase 匿名登录 API 会产生一个新的匿名用户。

邮箱登录

最近更新时间：2023-05-29 10:21:21

使用邮箱登录，您可以让您的用户**使用自己的邮箱和密码注册、登录 CloudBase，并且还可以更新登录使用的邮箱和密码。**

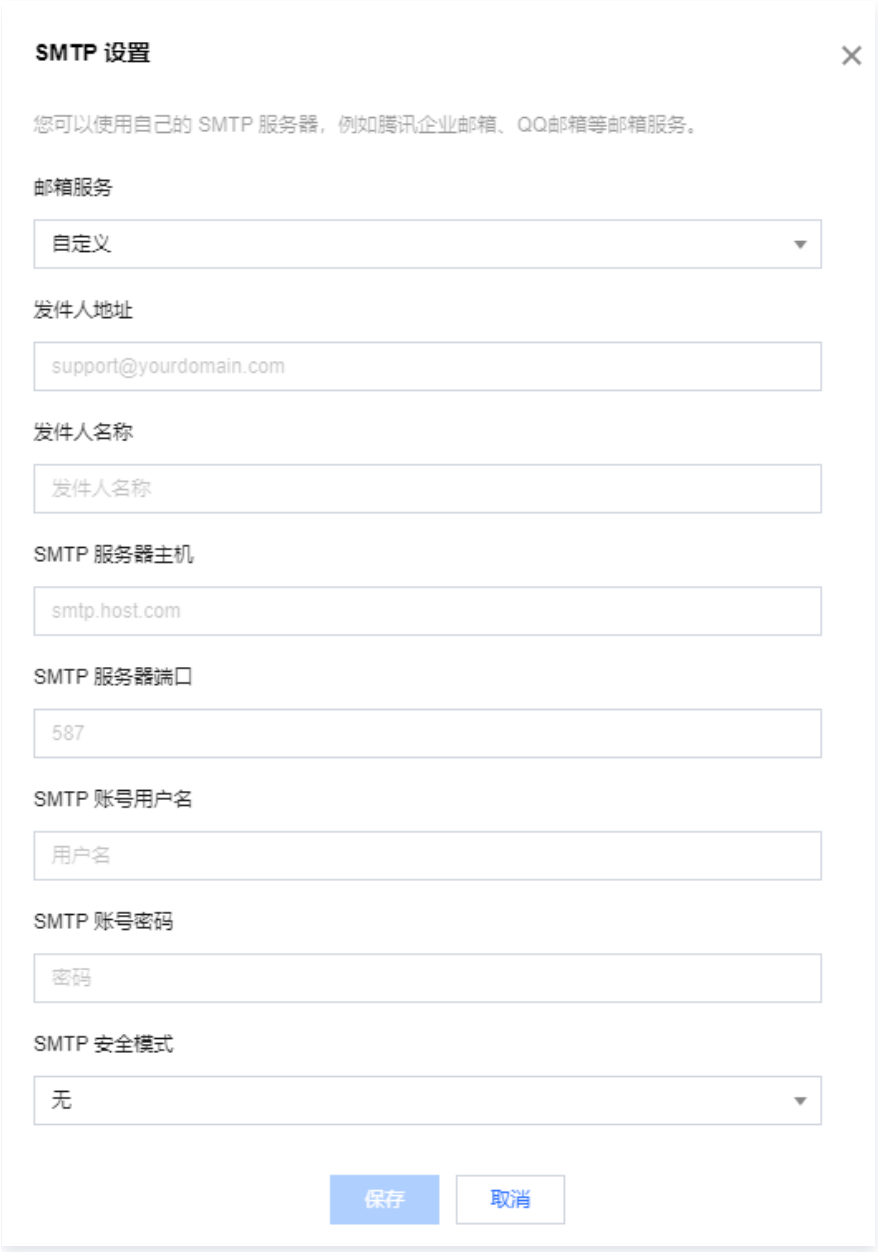
开通邮箱登录

步骤1：开启邮箱登录

进入 云开发 CloudBase 控制台，在 登录授权 设置页面中，开启邮箱登录：

步骤2：配置发件邮箱

打开右侧**配置发件人**页面，填入您邮箱的 **SMTP 账号信息**。



步骤3：设置应用名称及自动跳转链接

打开右侧**应用配置**页面，设置您的**应用名称**和**自动跳转链接**。

> **说明**
> - 您设置的**应用名称**将会出现在验证邮件的内容中。
> - CloudBase 发送的邮件中会包含一个 URL，用户打开邮件中的 URL 后，会自动跳转到您设置的**自动跳转链接**。

登录流程

步骤1：初始化 SDK

```
import cloudbase from "@cloudbase/js-sdk";

const app = cloudbase.init({
  env: "your-env-id"
});
```

步骤2：使用邮箱注册账号

首先需要用户填入自己的邮箱和密码，然后调用 SDK 的注册接口：

```
app
  .auth()
  .signUpWithEmailAndPassword(email, password)
  .then(() => {
    // 发送验证邮件成功
  });
```

调用注册接口之后，CloudBase 会使用您**预先设置的邮箱**，发送一封**验证邮件**到用户的邮箱。邮件中包含一个**激活链接**，用户在单击激活链接后，账号才会正式注册成功。

> **注意**
> 密码长度不小于 8 位，不大于 32 位，需要包含字母和数字。

步骤3：使用邮箱和密码登录 CloudBase

```
app
  .auth()
  .signInWithEmailAndPassword(email, password)
  .then((loginState) => {
    // 登录成功
```

```
});
```

使用 QQ 邮箱配置邮箱登录

步骤1：登录 QQ 邮箱

进入 QQ 邮箱首页，登录您的 QQ 邮箱。

步骤2：开启 IMAP/SMTP 服务

登录邮箱后，进入**设置** > **账户**：

然后，在**账户**设置中，找到**开启服务**设置项，开启 IMAP/SMTP 服务：

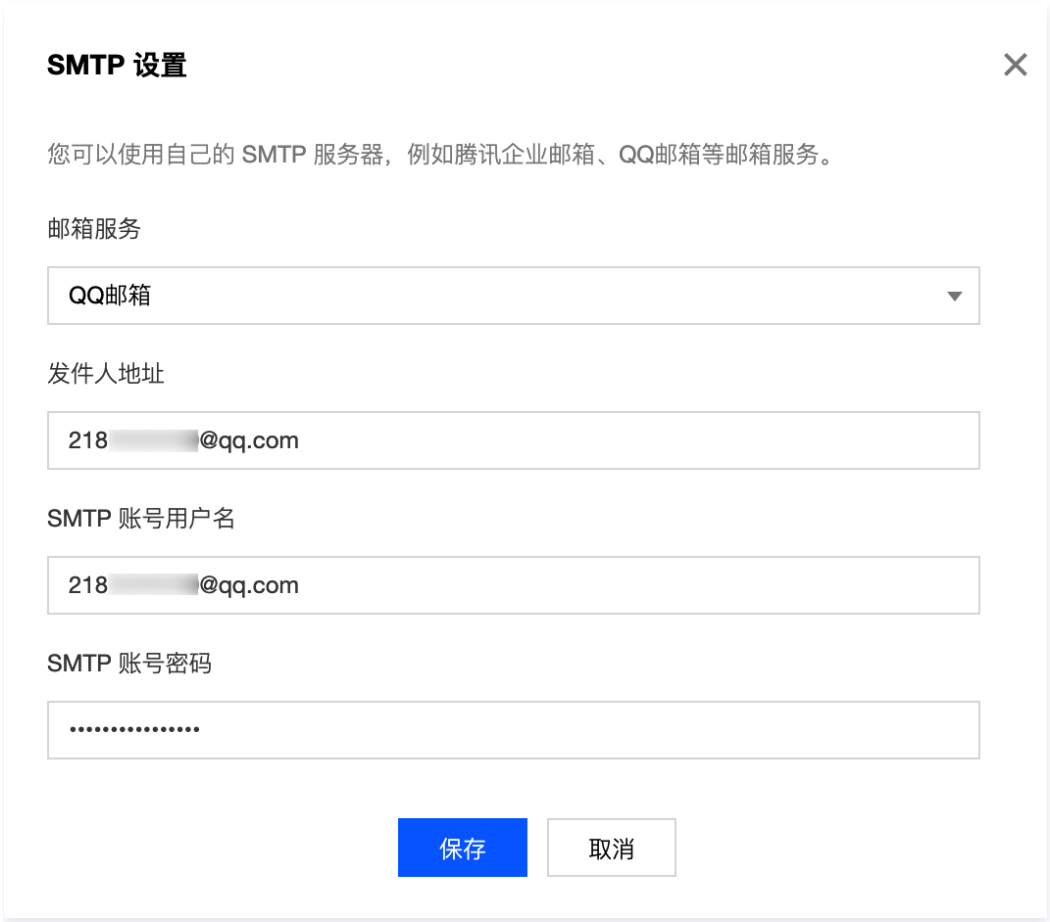
开启成功后，请保存您的邮箱登录授权码：

说明

您也可以开启 POP3/SMTP 服务，两种服务的授权码都可以作为第 3 步的 SMTP 账号密码。

步骤3：配置 QQ 邮箱作为发件人

进入 云开发 CloudBase 控制台，在 登录授权 页面中，打开右侧**配置发件人**页面，选择 QQ 邮箱作为邮箱服务，使用 QQ 邮箱作为发件人地址和 SMTP 账号用户名，使用第 2 步的授权码作为 SMTP 账号密码。


SMTP 设置
您可以使用自己的 SMTP 服务器，例如腾讯企业邮箱、QQ邮箱等邮箱服务。
邮箱服务
QQ邮箱
发件人地址
218@qq.com
SMTP 账号用户名
218@qq.com
SMTP 账号密码
保存
取消

自定义登录

最近更新时间：2024-10-09 14:33:51

开发者可以使用**自定义登录**，在自己的服务器或者云函数内，为用户签发带有**自定义身份 ID** 的自定义登录凭证 Ticket，随后用户端 SDK 便可以使用 Ticket 登录 CloudBase。

适用场景

自定义登录一般用于以下几种场景：

- 开发者希望将自有的账号体系与云开发 CloudBase 账号进行一对一关联。
- 开发者希望自行接管鉴权流程。

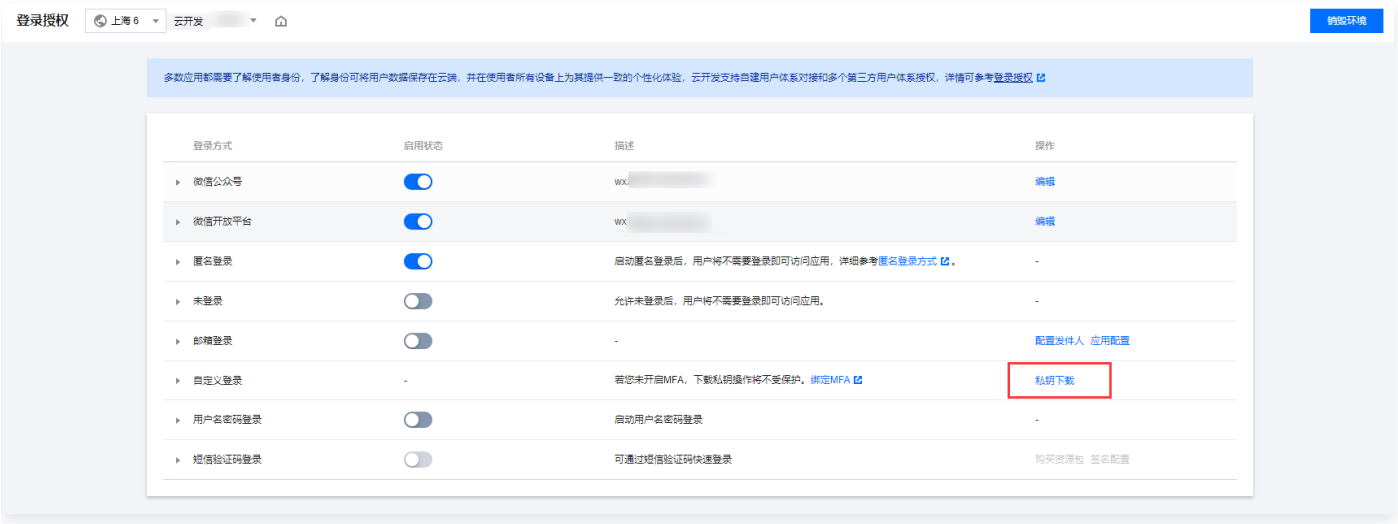
步骤概览

自定义登录需要以下几个步骤：

1. 获取 CloudBase 自定义登录私钥。
2. 使用 CloudBase 服务端 SDK，通过私钥签发出 Ticket，并返回至用户端。
3. 用户端 SDK 使用 Ticket 登录 CloudBase。

步骤1：获取自定义登录私钥

登录 CloudBase 控制台，在**环境** > **登录授权** 下的**自定义登录**栏中，单击**私钥下载**：



私钥是一份携带有 JSON 数据的文件，请将下载或复制的私钥文件保存到您的服务器或者云函数中，假设路径为 `/path/to/your/tcb_custom_login.json` 。

> **注意**
>
> - 私钥文件是证明管理员身份的重要凭证，请务必妥善保存，避免泄露。
> - 每次生成私钥文件都会**使之前生成的私钥文件在 2 小时后失效**。

步骤2：签发 Ticket

调用 CloudBase 服务端 SDK，在初始化时传入自定义登录私钥，随后便可以签发出 Ticket，并返回至用户端。

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

// 1. 初始化 SDK
const app = cloudbase.init({
  env: "your-env-id",
  // 传入自定义登录私钥
  credentials: require("/path/to/your/tcb_custom_login.json")
});

// 2. 开发者自定义的用户唯一身份标识
const customUserId = "your-customUserId";

// 3. 创建ticket
const ticket = app.auth().createTicket(customUserId);

// 4. 将ticket返回至客户端
return ticket;
```

说明

开发者也可以编写一个**云函数**用于生成 Ticket，并为其设置 HTTP 访问服务，随后用户端便可以通过 HTTP 请求的形式获取 Ticket，详细的方案请参见 使用 HTTP 访问云函数 。

注意

customUserId 必须满足以下需求：

- 4-32 位字符；
- 字符只能是大小写英文字母、数字、以及 `_-#@(){}<>[]:.,<>+#~` 中的字符。

步骤3：使用 Ticket 登录 CloudBase

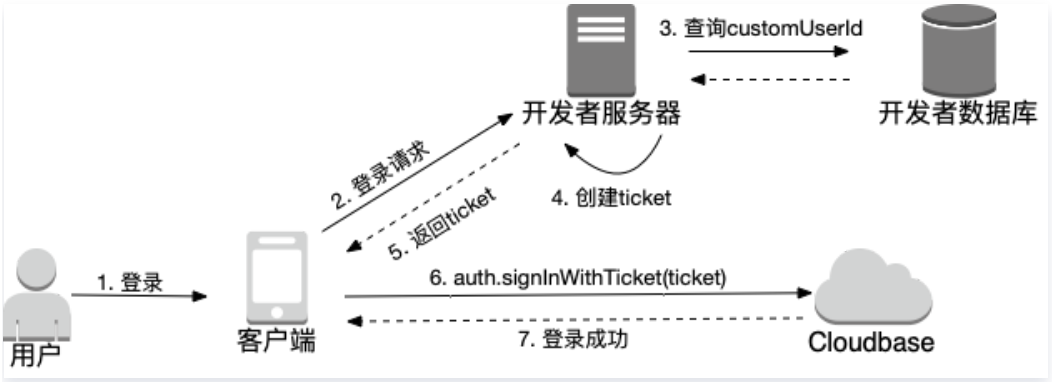
用户端应用获取到 Ticket 之后，便可以调用客户端 SDK 提供的 `auth.signInWithTicket()` 登录 CloudBase：

Web：

```
import cloudbase from '@cloudbase/js-sdk';

const app = cloudbase.init({
  env: 'your-env-id'
});

const auth = app.auth({ persistence: 'none' });

async function login(){
  const loginState = await auth.getLoginState();
  // 1. 建议登录前检查当前是否已经登录
  if(!loginState){
    // 2. 请求开发者自有服务接口获取ticket
    const ticket = await fetch('...');
```

```
    // 3. 登录 CloudBase
    await auth.customAuthProvider().signIn(ticket);
  }
}

login();
```

整体流程示意如下：



常见问题

自定义登录一定需要自己建设用于创建 Ticket 的服务器吗?

自定义登录必须有一个创建 Ticket 的服务，但是开发者并非一定要自己搭建服务器。

开发者还可以编写一个云函数来创建 Ticket，然后客户端使用 HTTP 请求调用这个云函数获取 Ticket，详细请参见 使用 HTTP 访问云函数 。

用户名密码登录

最近更新时间：2023-09-13 15:19:31

使用用户名密码登录，您可以让您的用户绑定用户名，并使用用户名密码登录 CloudBase。您可以更改用户名和密码，还可以查询用户名是否绑定过。

> **注意**
> **如果用户名未被绑定过，需要先使用其他登录方式完成登录后，才可以绑定用户名。绑定成功后，可以使用用户名和密码完成登录。**

开通用户名密码登录

进入 云开发 CloudBase 控制台，在 登录授权 设置页面中，开启用户名密码登录。

绑定用户名流程

步骤1：初始化 SDK

```
import cloudbase from "@cloudbase/js-sdk";

const app = cloudbase.init({
  env: "your-env-id"
});
```

步骤2：使用其他方式进行登录

绑定用户名之前，用户需要先使用其他方式进行登录，例如邮箱登录、微信公众号登录等，但不包括匿名登录。
下面以 邮箱登录 为例：

```
const auth = app.auth();
await auth.signInWithEmailAndPassword(email, password); // 邮箱登录
```

> **说明**

> 用户名可以是符合规则的任意字符串，为了避免您的应用被恶意者注册过多无效的用户名，CloudBase 目前不允许直接使用用户名 + 密码的形式注册用户。

步骤3：绑定用户名

绑定用户名时，可以检查在当前云开发环境下，此用户名是否存在。然后再调用绑定用户名的接口。

```
const auth = app.auth();
if (!(await auth.isUsernameRegistered(username))) {
  // 检查用户名是否绑定过
  await auth.currentUser.updateUsername(username); // 绑定用户名
}
```

> **注意**
>
> - 可以包含数字和字母，但是不允许是纯数字。
> - 符号只允许出现 `-` 和 `_`，不允许这两个符号出现在开头和结尾。
> - 长度范围是 `[1, 32]` 。

登录流程

步骤1：初始化 SDK

```
import cloudbase from "@cloudbase/js-sdk";

const app = cloudbase.init({
  env: "your-env-id"
});
```

步骤2：使用用户名和密码登录 CloudBase

```
const auth = app.auth();
const loginState = await auth.signInWithUsernameAndPassword(username, password); // 用户名密码登录
```

> **注意**
>
> 用户名登录和邮箱登录的密码是相同的。

短信验证码登录

最近更新时间：2023-09-13 16:55:03

使用短信验证码登录，您可以让用户使用自己的手机号，结合短信验证码或密码注册、登录 CloudBase，并且还可以更新或者解绑登录使用的手机号。

使用限制及费用

- 新开通的按量计费环境，或者 2021 年 4 月 9 日前开通的按量计费环境，享有首月 100 条的免费额度。
- 超出免费额度的需求，开发者可以前往云开发控制台 购买资源包 。
- 短信下发存在频率限制：
  - 同一号码 30 秒内最多发送 1 条。
  - 同一手机号一个自然日最多发送 100 条。

开通短信验证码登录

进入 云开发 CloudBase 控制台，在 登录授权 设置页面中，开启短信验证码登录。

登录流程

步骤1：初始化 SDK

```
import cloudbase from "@cloudbase/js-sdk";
```

```
const app = cloudbase.init({
  env: "your-env-id"
});
```

步骤2：使用手机号注册账号

首先需要用户填入自己的手机号，然后调用 SDK 的发送短信验证码接口：

```
app
  .auth()
  .sendPhoneCode(phoneNumber)
  .then(() => {
    // 发送短信验证码
  });
```

调用发送短信接口后，手机将会收到云开发的短信验证码。用户填入短信验证码，以及自定义密码后，调用注册账号接口：

```
app
  .auth()
  .signUpWithPhoneCode(phoneNumber, phoneCode, password)
  .then(() => {
    // 手机短信注册账号
  });
```

> ⚠ **注意**
>
> 密码长度不小于 8 位，不大于 32 位，需要包含字母和数字。

步骤3：使用手机号加密码或手机号加短信验证码登录

```
app
  .auth()
  .signInWithPhoneCodeOrPassword({
    phoneNumber,
    phoneCode, // 非必填，验证码和密码至少二选一
    password // 非必填，验证码和密码至少二选一
  })
  .then((loginState) => {
    // 登录成功
  });
```

管理用户

最近更新时间：2023-09-13 16:55:03

创建用户

开发者可以调用以下登录方式，登录或者创建一个用户：

- 邮箱登录
- 微信登录
- 自定义登录
- 用户名密码登录
- 匿名登录

获取当前登录的用户

订阅登录状态变化的回调函数

获取当前用户，推荐在 `Auth` 对象上设置一个回调函数，每当用户登录状态转变时，会触发这个回调函数，并且获得当前的 `LoginState`：

```
import cloudbase from "@cloudbase/js-sdk";

const app = cloudbase.init({
  env: "your-env-id"
});
const auth = app.auth();

// 设置一个观察者
auth.onLoginStateChanged((loginState) => {
  if (loginState) {
    // 此时用户已经登录
  } else {
    // 没有登录
  }
});
```

直接获取当前用户

您还可以使用 `Auth.currentUser` 属性来获取当前登录的用户。如果用户未登录，则 `currentUser` 为 `null`：

```
const user = auth.currentUser;

if (user) {
  // 此时用户已经登录
} else {
  // 没有登录
}
```

获取用户个人资料

您可以通过 `User` 对象的各个属性来获取用户的个人资料信息：

```
const user = auth.currentUser;
let uid, nickName, gender, avatarUrl, location;

if (user) {
  // 云开发唯一用户 id
  uid = user.uid;

  // 昵称
  nickName = user.nickName;

  // 性别
  gender = user.gender;

  // 头像URL
  avatarUrl = user.avatarUrl;

  // 用户地理位置
  location = user.location;
}
```

更新用户个人资料

您可以使用 `User.update` 方法来更新用户的个人资料信息。例如：

```
const user = auth.currentUser;

user
  .update({
    nickName: "Tony Stark",
    gender: "MALE",
    avatarUrl: "https://..."
  })
  .then(() => {
    // 更新用户资料成功
  });
```

刷新用户资料信息

对于一个多端应用，用户可能在其中某个端上更新过自己的个人资料信息，此时其它端上可能需要刷新信息：

```
const user = auth.currentUser;

// 刷新用户信息
user.refresh().then(() => {
  // 刷新后，获取到的用户信息即为最新的信息
```

```
  const { nickName, gender, avatarUrl } = user;
});
```

实践教程

最近更新时间：2024-06-11 15:03:31

避免重复登录

执行登录流程之前，我们非常建议您**先判断用户端是否已经登录 CloudBase**，如已经登录，那么不需要执行登录流程，以**避免无意义的重复登录。**

```
const auth = app.auth();

// 应用初始化时
if (auth.hasLoginState()) {
  // 此时已经登录
} else {
  // 此时未登录，执行您的登录流程
}
```

登录状态的持久保留

您可以指定登录状态如何持久保留。默认为 `local` ，相关选项包括：

值	说明
session	在 SessionStorage 中保留登录状态，当前页面关闭后会被清除。
local	在本地存储中长期地保留登录状态。
none	在内存中保留登录状态，当前页面刷新、重定向之后会被清除。

例如，对于网页应用，建议选择是 `local` ，即在用户关闭浏览器之后仍保留该用户的会话。这样，用户不需要每次访问该网页时重复登录，避免给用户带来诸多不便体验。

```
const auth = app.auth({
  persistence: "local"
});
```

数据库
概述

最近更新时间：2023-09-13 16:55:03

云数据库是 CloudBase 提供的核心功能之一，提供基础读写、聚合搜索、数据库事务、实时推送等功能。

基础概念

记录（Record / Document）

云数据库是一种**文档型数据库**，数据库中的每条记录都是一个类似 JSON 格式的对象，例如：

```
{
  "name": "Tom",
  "age": 18,
  "location": {
    "country": "China",
    "province": "Guangdong",
    "city": "Shenzhen"
  }
}
```

集合（Collection）

- 集合由**多条记录**组成，任何记录必须从属于某个集合。
- 集合是读写操作的主要对象，每个集合都有一个**集合名**，例如 users、articles 等。

数据库（Database）

每个云开发环境下**有且只有一个**数据库实例，数据库实例中，可以创建多个集合。

调用方式

云数据库可以在**用户端**（例如 Web 网页、小程序）内调用，也可以在**服务端**（例如服务器、云函数）内调用。

用户端调用

通过用户端调用时，需要先进行云开发的 登录鉴权 ，然后**以用户的身份**进行数据库的读写操作。

> **说明**
> 用户端的代码是可能对外暴露的（例如 Web 网页），攻击者可能通过抓取、伪造请求的方式盗用或消耗您的 CloudBase 资源，所以我们提供了用户端上的 登录鉴权 机制来保护您的资源安全。

Web

```
const cloudbase = require("@cloudbase/js-sdk");
```

```
const app = cloudbase.init({
  env: "xxxx"
});

/**
  登录鉴权流程，此处代码略，请参考：
  https://cloud.tencent.com/document/product/876/41728
*/

// 1. 获取数据库引用
var db = app.database();

// 2. 构造查询语句
db
  // collection() 方法获取一个集合的引用
  .collection("books")
  // where() 方法传入一个 query 对象，数据库返回集合中字段等于指定值的 JSON 文档。
  .where({
    name: "The Catcher in the Rye"
  })
  // get() 方法会触发网络请求，往数据库取数据
  .get()
  .then(function (res) {
    console.log(res);
    // 输出 [{ "name": "The Catcher in the Rye", ... }]
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

// 2. 构造查询语句
db
  // collection() 方法获取一个集合的引用
  .collection("books")
  // where() 方法传入一个 query 对象，数据库返回集合中字段等于指定值的 JSON 文档。
  .where({
    name: "The Catcher in the Rye"
  })
  // get() 方法会触发网络请求，往数据库取数据
  .get()
  .then(function (res) {
    console.log(res);
    // 输出 [{ "name": "The Catcher in the Rye", ... }]
  });
```

服务端调用

通过服务端调用时，需要在 SDK 初始化参数中，填入腾讯云密钥（SecretID 和 SecretKey），然后**以管理员身份**进行数据库的读写操作。

> **说明**
> 在 CloudBase 云函数内使用服务端 SDK 时，开发者不需要填入腾讯云密钥即可使用。

```
const cloudbase = require('@cloudbase/node-sdk')

const app = cloudbase.init({})

// 1. 获取数据库引用
var db = app.database();

exports.main = async (event, context) => {
  // 2. 构造查询语句
  const res = await db
    // collection() 方法获取一个集合的引用
    .collection("books")
    // where() 方法传入一个 query 对象，数据库返回集合中字段等于指定值的 JSON 文档。
    .where({
      name: "The Catcher in the Rye"
    })
    // get() 方法会触发网络请求，往数据库取数据
    .get()

  return {
    res
  }
}
```

权限控制

上文中提到，用户端与服务端是**以不同的身份和权限**调用云数据库的。

服务端调用

服务端上，是以**管理员身份**调用云数据库的，**拥有读取、写入、修改、删除任意数据的权限**。所以服务端**又称管理端**。

用户端调用

用户端上，需要进行 登录鉴权 之后，以**当前用户的身份**调用云数据库，受到**数据库权限**的控制。
云数据库支持四种基础的数据库权限，如下：

权限	说明	使用场景
仅创建者可写，所有人可读	数据只有创建者可写、所有人可读	文章、公开评论

仅创建者可读写	数据只有创建者可读写，其他用户不可读写	私密相册、用户私密数据
仅管理端可写，所有人可读	该数据只有管理端可写，所有人可读	商品信息、配置信息
仅管理端可读写	该数据只有管理端可读写	不对外暴露的数据

例如，当数据库权限设置为**仅创建者可写，所有人可读**时，来自用户端的调用只能修改、删除当前用户身份的数据，但可以读到其它用户创建的数据。

> **说明**
> 在某些复杂场景下，如果基础权限控制无法满足您的需求，您可以使用 自定义安全规则，通过编写规则语句来完成权限的设置。

特殊字段

_id 字段

云数据库中，每条记录都有一个 `_id` 字段作为这条数据的**唯一标识**，在插入记录时会自动生成。您也可以使用自定义的 `_id`，但需要保证**全局唯一性**。

_openid 字段

每条记录中**可能**存在 `_openid` 字段，用于标识记录的创建者，它会在插入记录时，根据用户身份自动生成。

调用来源	`_openid` 含义
微信小程序	用户的微信 OpenID
Web SDK	用户的云开发 Uid

> **说明**
> 在服务端（例如云函数）或者管理端（例如控制台）中创建的记录，不会自动生成 `_openid` 字段，因为这是属于**管理员**创建的记录，并不属于某个用户。

数据类型

最近更新时间：2023-08-23 10:18:31

云开发数据库提供以下几种数据类型：

- String：字符串
- Number：数字
- Object：对象
- Array：数组
- Bool：布尔值
- GeoPoint：地理位置点
- Date：时间
- Null

下面对几个需要额外说明的字段做下补充说明。

Date

Date 类型用于创建客户端时间，精确到毫秒，可以用 JavaScript 内置 Date 对象创建。如果需要使用服务端时间，应该用 API 中提供的 serverDate 对象来创建一个服务端当前时间的标记。
我们的数据库有针对日期类型的优化，建议您使用时都用 Date 或 serverDate 构造时间对象。

GeoPoint

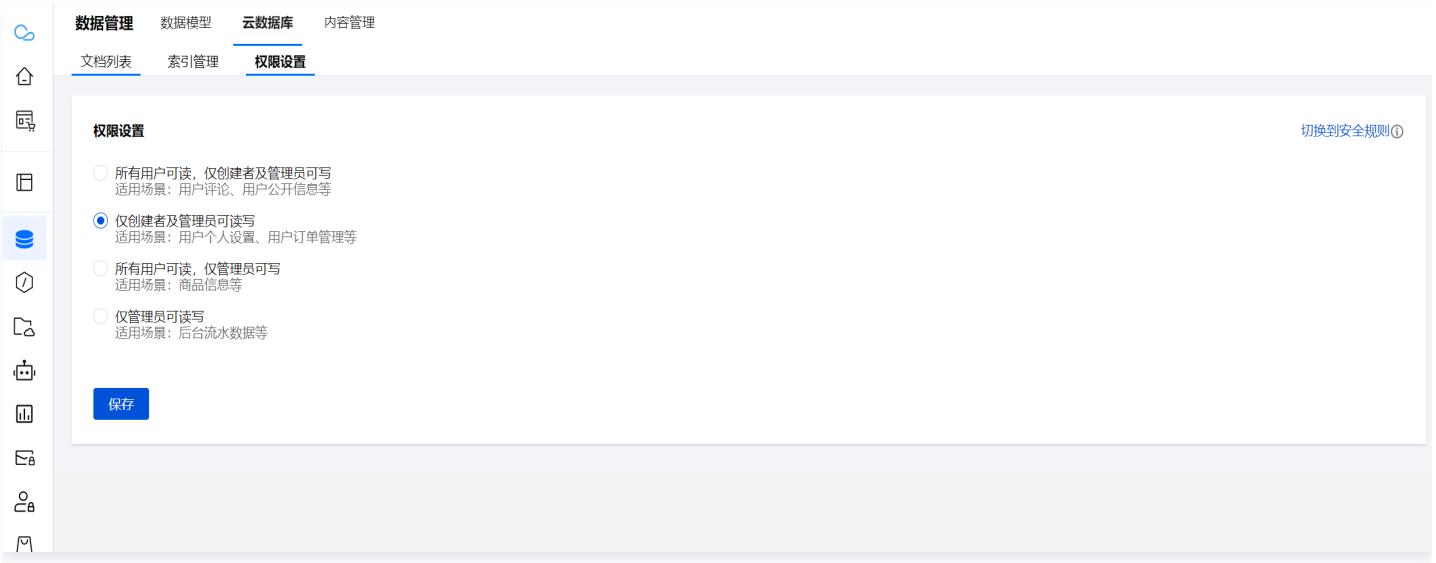
GeoPoint 类型用于表示地理位置点，用经纬度唯一标记一个点，这是一个特殊的数据存储类型。注意，如果需要对类型为地理位置的字段进行查找，一定要建立地理位置索引。
具体的地理位置 API 请参见 Geo API 文档。

管理数据库

最近更新时间：2024-09-26 17:39:21

设置权限

前往 云开发统一工作台，选择要访问的环境，进入云数据库页面。
针对每个集合设置对应的权限，可在**权限设置**下设置：



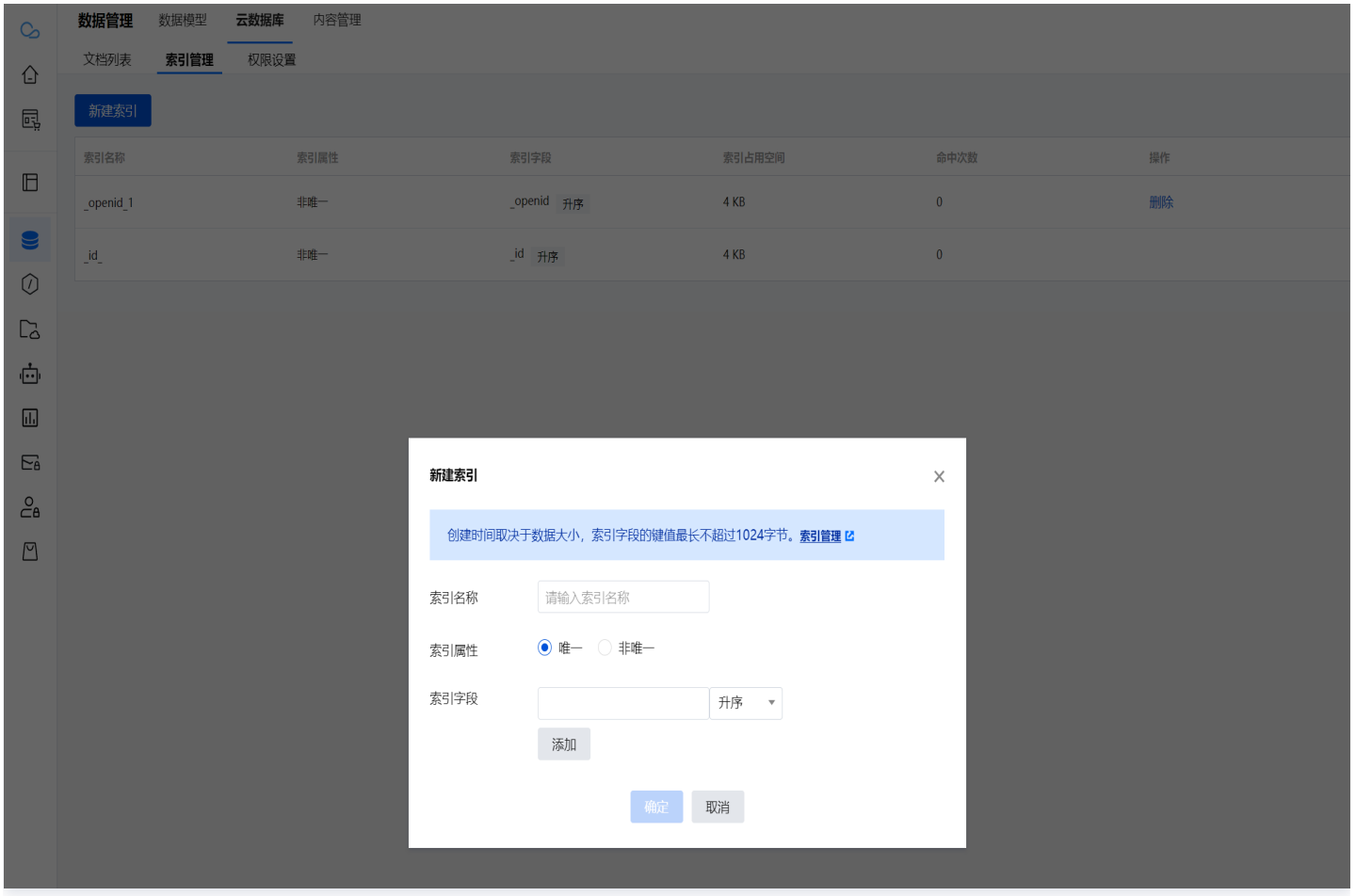
数据库权限详情参见 权限控制 。

索引管理

在集合中为查询条件的字段建立索引，是保证数据库性能、提升用户体验的重要手段。

操作步骤

1. 前往 云开发统一工作台，选择要访问的环境，进入云数据库页面。
2. 切换到**数据库集合**页，并选择需要导入数据的集合。
3. 进入**索引管理**页面可进行**删除**或**新建索引**操作。



单字段索引

您可以为查询条件对应的字段创建单字段索引，如果该字段是嵌套字段，可以使用**点表示法**。例如对如下格式的记录中的 color 字段进行索引时，可以用 style.color 表示。

```
{
  "_id": "",
  "style": {
    "color": ""
  }
}
```

在设置单字段索引时，可任意指定索引的排序为升序或降序，数据库总能在对索引字段的排序查询中，进行正确的排序。

组合索引

组合索引即一个索引包含多个字段。当查询条件使用的字段包含在索引定义的所有字段或前缀字段里时，会命中索引，优化查询性能。

> **说明：**

索引前缀即组合索引的字段中定义的前 1 到多个字段，例如对集合 students 中 name、age、score 三个字段按顺序定义了组合索引，那么该索引的前缀包含：

- name
- name, age

能命中索引的查询字段组合包含：

- name
- name, age
- name, age, score

```
{
  "_id": "1",
  "name": "luke",
  "age": 26,
  "score": 80
}
```

组合索引具有以下特点：

1. **字段顺序决定组合索引效果：**
   例如，定义组合索引分别为 name, age 与 age, name 是不同的。当组合索引为 name, age 时，其索引前缀为 name，对字段 name 的查询可以命中 name, age 索引，而对字段 age 的查询无法命中该索引，因为 age 不属于 name, age 的前缀（反之字段 age 能命中 age, name 索引）。
2. **查询字段排序影响命中索引：**
   组合索引为 **age：升序，score：降序**，字段排序对索引命中效果如下：

age：升序，score：降序	age：降序，score：升序	age：升序，score：升序	age：降序，score：降序	score：升序/降序，age：升序/降序
命中	命中	未命中	未命中	未命中

   组合索引为 **age：升序，score：升序**，字段排序对索引命中效果如下：

age：升序，score：升序	age：降序，score：降序	age：升序，score：降序	age：降序，score：升序	score：升序/降序，age：升序/降序
命中	命中	未命中	未命中	未命中

地理位置索引

云开发目前支持建立平面几何的地理位置索引，使用地理位置查询功能时，必须为地理位置数据的字段建立地理位置索引。
例如对含地理位置字段 point 的集合建立地理位置索引：

```
{
  "_id": "",
  "point": new db.Geo.Point(50, 50)
}
```

索引使用注意事项

唯一性限制

创建索引时，索引属性选择**唯一**，即可添加唯一性限制。此限制会要求集合中**索引字段对应的值不能重复**。
例如，某个集合内建立了索引字段 foo，且属性为**唯一**，那么在这个集合内，要求不能存在 foo 字段相同的文档。

> **说明：**
> 需特别注意的是，假如**记录中不存在某个字段，则对索引字段来说其值默认为 null**。如果索引有唯一性限制，则不允许存在两个或以上的该字段为空 / 不存在该字段的记录。

大小限制

- 索引的字段大小限制**不能超过 1024 字节**。
- 添加索引时，如果集合中已有文档索引字段超过 1024 字节，添加索引时将报错。
- 已设置索引的字段，如果插入一个文档，文档中该字段超过 1024 字节将会报错。

> **说明：**
> 每个英文字母(不分大小写)占一字节的空间，每个中文汉字占两字节的空间。

正则表达式

正则查询无法使用索引提升性能。

导入数据

云开发支持在 云开发 CloudBase 控制台 进行数据导入。

操作步骤

1. 登录 云开发 CloudBase 控制台 ，并单击前往云开发统一工作台。
2. 切换到**数据库集合**页，并选择需要导入数据的集合。进入对应的集合管理页面，单击**导入**。

3. 选择要导入文件的格式以及冲突处理模式，单击**导入**，即可开始导入的过程。

说明：

目前提供了 Insert、Upsert 两种冲突处理模式。

- Insert 模式会在导入时总是插入新记录。
- Upsert 模式会判断有无该条记录，如果有则更新记录，否则就插入一条新记录。

JSON、CSV 文件必须是 UTF-8 的编码格式，且其内容类似 MongoDB 的导出格式，示例代码如下：

JSON 示例

```
{
    "_id": "xxxxxx",
    "age": 45
}
{
    "_id": "yyyyyy",
    "age": 21
}
```

CSV 示例

```
_id, age
xxxxxx, 45
yyyyyy, 21
```

说明：

- JSON 数据不是数组，而是类似 JSON Lines，即各个记录对象之间使用 \n 分隔，而非逗号。
- JSON 数据每个键值对的键名首尾不能是 .，例如 ".a"、"abc."，且不能包含多个连续的 .，例如 "a..b"。
- 键名不能重复，且不能有歧义，例如 {"a": 1, "a": 2} 或 {"a": {"b": 1}, "a.b": 2}。
- 时间格式须为 ISODate 格式，例如 "date": { "$date" : "2018-08-31T17:30:00.882Z" }。
- 当使用 Insert 冲突处理模式时，同一文件不能存在重复的 _id 字段，或与数据库已有记录相同的 _id 字段。
- CSV 格式的数据默认以第一行作为导入后的所有键名，余下的每一行则是与首行键名一一对应的键值记录。

4. 导入完成后，您可以在提示信息中看到本次导入记录的情况。

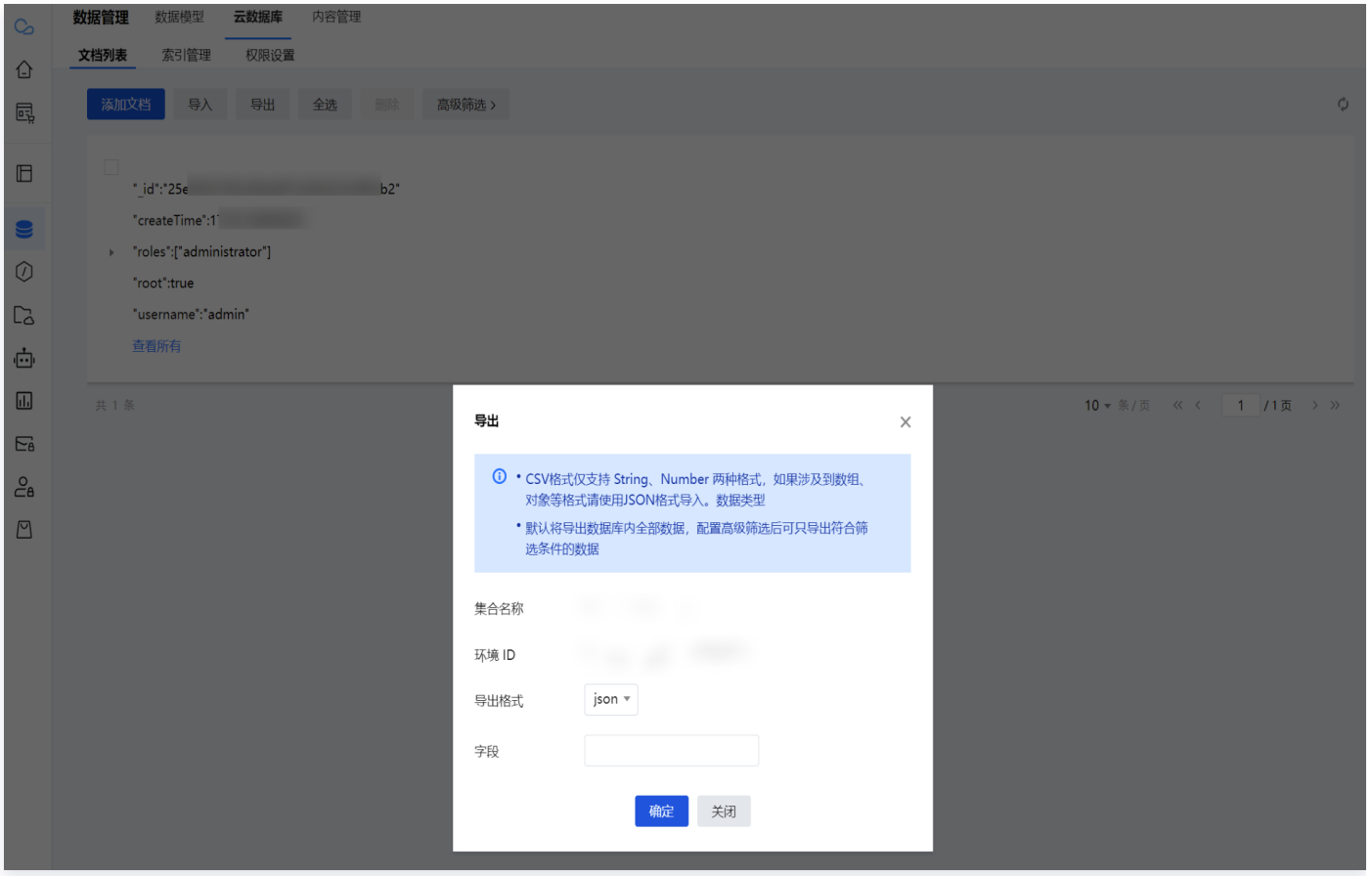
导出数据

云开发支持导出集合已有的数据（目前仅支持导出 CSV、JSON 格式的文件数据）。

操作步骤

1. 登录 云开发 CloudBase 控制台 ，并点击前往云开发统一工作台。
2. 切换到**数据库集合**页，并选择需要导出数据的集合，单击**导出**。
3. 选择要导出的格式、保存的位置、以及字段，单击**导出**，即可开始导出的过程。
   - 当选择导出格式为 JSON 时，若不填写字段项，则默认导出所有数据。
   - 当选择导出格式为 CSV 时，则字段为必填项。字段之间使用英文逗号隔开。
     字段示例代码如下所示：

```
_id, name, age, gender;
```


数据管理
数据模型
云数据库
内容管理
文档列表
索引管理
权限设置
添加文档
导入
导出
全选
删除
高级筛选
"_id":"25e
b2"
"createTime":1
"roles":["administrator"]
"root":true
"username":"admin"
查看所有
共 1 条
10 条/页
1 / 1 页
导出
CSV格式仅支持 String、Number 两种格式，如果涉及到数组、对象等格式请使用JSON格式导入。数据类型
默认将导出数据库内全部数据，配置高级筛选后可只导出符合筛选条件的数据
集合名称
环境 ID
导出格式
json
字段
确定
关闭

初始化

最近更新时间：2023-09-13 15:40:22

新建第一个集合

目前仅支持通过云函数端 SDK 及 云开发 CloudBase 控制台 新建集合。

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({
  env: "your-env-id",
});
const db = app.database();

exports.main = async (event, context) => {
  const res = await db.createCollection("todos");
  console.log(res);
};
```

获取数据库的引用

在开始使用数据库 API 进行增删改查操作之前，需要先获取数据库的引用 db，示例代码如下：

Web

```
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "your-env-id",
});
const db = app.database();
```

小程序

```
const db = wx.cloud.database();
```

此外，在小程序端，如需获取其他环境的数据库引用，可以在调用时传入一个含 `config` 字段的参数，在其中通过 `env` 字段指定要使用的环境，如：

```
const testDB = wx.cloud.database({
  config: {
    env: "your-env-id",
  },
});
const db = app.database();
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({
  env: "your-env-id",
  secretId: "...",
  secretKey: "...",
});
const db = app.database();
```

获取一个集合的引用

要操作一个集合，需先获取它的引用。在获取了数据库的引用后，就可以通过数据库引用上的 collection 方法获取一个集合的引用了，例如获取待办事项清单集合：

Web

```
const todos = db.collection("todos");
```

小程序

```
const todos = db.collection("todos");
```

Node.js

```
const todos = db.collection("todos");
```

获取集合的引用并不会发起网络请求去拉取它的数据，我们可以通过此引用在该集合上进行增删查改的操作，除此之外，还可以通过集合上的 doc 方法来获取集合中一个指定 ID 的记录的引用。同理，记录的引用可以用于对特定记录进行更新和删除操作。
假设我们有一个待办事项的 ID 为 `todo-identifiant-aleatoire`，那么我们可以通过 doc 方法获取它的引用：

Web

```
const todo = db.collection("todos").doc("todo-identifiant-aleatoire");
```

小程序

```
const todo = db.collection("todos").doc("todo-identifiant-aleatoire");
```

Node.js

```
const todo = db.collection("todos").doc("todo-identifiant-aleatoire");
```

插入数据

最近更新时间：2024-11-14 11:46:02

准备工作

在插入数据前，请先创建集合，具体可参见 新建第一个集合 。
假设已有一个集合 books，其中包含以下格式记录：

```
[
  {
    "_id": "xxx",
    "category": "Novel",
    "name": "The Catcher in the Rye",
    "onSale": true,
    "sales": 80
  }
]
```

插入一条数据

Web

小程序端插入的数据，需要放在 `data` 字段内。

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("books")
  .add({
    // _id: 'todo-identifiant-aleatoire', // 可选自定义 _id，在此处场景下用数据库自动分配的就可以了
    category: "Computer",
    name: "Thinking in Java",
    onSale: true,
    sales: 100
  })
  .then((res) => {
    console.log(res);
  });
```

Node.js

```
const cloudbase = require('@cloudbase/node-sdk')

const app = cloudbase.init({})

// 1. 获取数据库引用
var db = app.database()
exports.main = async (event, context) => {
  const res = await db.collection('books')
    .add({
      category: 'Computer',
      name: 'Thinking in Java',
      onSale: true,
      sales: 100
    })
  return {
    res
  }
}
```

插入多条数据

目前仅支持通过服务端 SDK 使用。

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
var db = app.database();
exports.main = async () => {
  const res = await db.collection("todos")
    .add([
      {
        name: 'The Moon and Six pence',
        category: 'Novel',
        saling: false,
        sales: 30
      },
      {
        name: '吾□は猫である',
```

```
        category: 'Novel',
        saling: false,
        sales: 90
      }
    ])
  return {
    res
  }
}
```

在创建成功之后，我们可以在控制台中查看到刚新增的数据。

读取数据

最近更新时间：2024-09-30 10:08:01

假设我们已有一个集合 todos，其中包含以下格式记录：

```
[
  {
    "_id": "todo-identifiant-aleatoire",
    "due": Date("2018-09-01"),
    "style": {
      "color": "white"
    },
    "tags": ["cloud", "database"],
    "process": 20,
    "done": false
  },
  {
    "_id": "todo-identifiant-aleatoire-2",
    "due": Date("2018-12-25"),
    "tags": ["cloud", "database"],
    "style": {
      "color": "yellow"
    },
    "process": 50,
    "done": false
  }
  // more...
]
```

获取一个记录的数据

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .get()
  .then((res) => {
    // 2. 获取到返回数据后，在控制台打印查询得到的数据
    // res.data 包含该记录的数据，本例中为上述 todos 集合上 id 为 todo-identifiant-aleatoire
的文档，即：
```

```
    // [
    //   {
    //     "_id": "todo-identifiant-aleatoire",
    //     "due": Date("2018-09-01"),
    //     "style": {
    //       "color": "white"
    //     },
    //     "tags": ["cloud", "database"],
    //     "process": 20,
    //     "done": false
    //   }
    // ]
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .get()
  .then((res) => {
    // 2. 获取到返回数据后，在控制台打印查询得到的数据
    // res.data 包含该记录的数据，本例中为上述 todos 集合上 id 为 todo-identifiant-aleatoire
的文档，即：
    // {
    //   "_id": "todo-identifiant-aleatoire",
    //   "due": Date("2018-09-01"),
    //   "style": {
    //     "color": "white"
    //   },
    //   "tags": ["cloud", "database"],
    //   "process": 20,
    //   "done": false
    // }
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();
exports.main = async (event, context) => {
```

```
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire")
    .get();

  // 2. 获取到返回数据后，将数据返回
  // res.data 包含该记录的数据，本例中为上述 todos 集合上 id 为 todo-identifiant-aleatoire
的文档，即：
  // [
  //   {
  //     "_id": "todo-identifiant-aleatoire",
  //     "due": Date("2018-09-01"),
  //     "style": {
  //       "color": "white"
  //     },
  //     "tags": ["cloud", "database"],
  //     "process": 20,
  //     "done": false
  //   }
  // ]
  return {
    res,
  };
};
```

说明

由于数据库权限，用户端可能会读不到指定 _id 的数据，需要把数据库权限设定为 **仅创建者可写，所有人可读** 或 **仅管理端可写，所有人可读**，具体可以参考 数据库权限 。

获取多个记录的数据

我们也可以一次性获取多条记录。通过调用集合上的 where 方法可以指定查询条件，再调用 get 方法即可只返回满足指定查询条件的记录，例如获取所有未完成的待办事项。示例代码如下：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("todos")
  .where({
    done: false
```

```
  })
  .get()
  .then((res) => {
    // res.data 是包含以上定义的两条记录的数组
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

db.collection("todos")
  .where({
    done: false
  })
  .get()
  .then((res) => {
    // res.data 是包含以上定义的两条记录的数组
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
var db = app.database();

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .where({
      done: false
    })
    .get();

  return {
    res
  };
};
```

where 方法接收一个对象参数，该对象中每个字段和它的值构成一个需满足的匹配条件，各个字段间的关系是 与 的关系，即需同时满足这些匹配条件，在这个例子中，就是查询出 done 等于 false 的记录。

查询嵌套字段

在查询条件中我们也可以指定匹配一个嵌套字段的值，例如找出标为黄色的待办事项：

Web

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("todos")
  .where({
    style: {
      color: "yellow"
    }
  })
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

db.collection("todos")
  .where({
    style: {
      color: "yellow"
    }
  })
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
var db = app.database();

exports.main = async (event, context) => {
  const res = await db.collection("todos")
    .where({
      style: {
        color: "yellow"
      }
    })
    .get()

  return {
    res
  }
}
```

使用点表示法表示嵌套字段：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("todos")
  .where({
    "style.color": "yellow"
  })
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
```

```
const db = wx.cloud.database();

db.collection("todos")
  .where({
    "style.color": "yellow"
  })
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .where({
      "style.color": "yellow"
    })
    .get();

  return {
    res
  };
};
```

获取一个集合的数据

如果要获取一个集合的数据，例如获取 todos 集合上的所有记录，可以在集合上调用 get 方法获取，但通常不建议这么使用，在客户端中我们需要尽量避免一次性获取过量的数据，只应获取必要的数据。为了防止误操作以及保护用户体验，在获取集合数据时，小程序端请求时，服务器默认且最多返回 20 条记录，Web 端及云函数端 SDK 请求时最多返回 1000 条，默认 100 条。

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
```

```
var db = app.database();

db.collection("todos")
  .get()
  .then((res) => {
    // res.data 是一个包含集合中有权限访问的所有记录的数据
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

db.collection("todos")
  .get()
  .then((res) => {
    // res.data 是一个包含集合中有权限访问的所有记录的数据
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

exports.main = async (event, context) => {
  const res = await db.collection("todos").get();
  return {
    res
  };
};
```

查询数据

最近更新时间：2023-09-13 15:48:31

使用数据库 API 提供的 where 方法我们可以构造复杂的查询条件完成复杂的查询任务。在本节中我们还是使用 读取数据 中使用的示例数据。

查询指令

假设我们需要查询进度大于 30% 的待办事项，那么传入对象表示全等匹配的方式就无法满足了，这时就需要用到查询指令。数据库 API 提供了大于、小于等多种查询指令，这些指令都暴露在 `db.command` 对象上。例如查询进度大于 30% 的待办事项：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

const _ = db.command;
db.collection("todos")
  .where({
    // gt 方法用于指定一个 "大于" 条件，此处 _.gt(30) 是一个 "大于 30" 的条件
    progress: _.gt(30)
  })
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const _ = db.command;
db.collection("todos")
  .where({
    // gt 方法用于指定一个 "大于" 条件，此处 _.gt(30) 是一个 "大于 30" 的条件
    progress: _.gt(30)
  })
  .get()
  .then((res) => {
    console.log(res.data);
```

```
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

const _ = db.command;

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .where({
      // gt 方法用于指定一个 "大于" 条件，此处 _.gt(30) 是一个 "大于 30" 的条件
      progress: _.gt(30)
    })
    .get();
  return {
    res
  };
};
```

API 提供了以下查询指令：

查询指令	说明
eq	等于
neq	不等于
lt	小于
lte	小于或等于
gt	大于
gte	大于或等于
in	字段值在给定数组中
nin	字段值不在给定数组中

具体的查询指令 API 文档可参考各 SDK API。

逻辑指令

除了指定一个字段满足一个条件之外，我们还可以通过指定一个字段需同时满足多个条件，例如我们 **查询进度小于或等于 50% 或 颜色为白色或黄色** 的待办事项：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

const _ = db.command;
db.collection("todos")
  .where(
    _.or([
      {
        progress: _.lte(50)
      },
      {
        style: {
          color: _.in(["white", "yellow"])
        }
      }
    ])
  )
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const _ = db.command;
db.collection("todos")
  .where(
    _.or([
      {
        progress: _.lte(50)
      },
      {
        style: {
          color: _.in(["white", "yellow"])
```

```
        }
      }
    ])
  )
  .get()
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require('@cloudbase/node-sdk');

const app = cloudbase.init({})
// 1. 获取数据库引用
const db = app.database()

const _ = db.command

exports.main = async (event, context) => {
  const res = await db.collection('todos')
  .where(
    _.or([
      {
        progress: _.lte(50)
      },
      {
        style: {
          color: _.in(['white', 'yellow'])
        }
      }
    ])
  )
  .get()

  return {
    res
  }
}
```

具体的查询指令 API 文档可参考各 SDK API。

更新数据

最近更新时间：2023-06-12 15:36:24

在本节中我们还是沿用 读取数据 章节中使用的数据。
更新数据主要有两个方法：

API	说明
update	局部更新一个或多个记录。
set	替换更新一个记录。

局部更新

使用 `update` 方法可以局部更新一个记录或一个集合中的记录，局部更新意味着只有指定的字段会得到更新，其他字段不受影响。例如我们可以用以下代码将一个待办事项设置为已完成：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .update({
    // 表示将 done 字段置为 true
    done: true
  })
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

db.collection("todos")
```

```
  .doc("todo-identifiant-aleatoire")
  .update({
    // data 传入需要局部更新的数据
    data: {
      // 表示将 done 字段置为 true
      done: true
    }
  })
  .then((res) => {
    console.log(res.data);
  });
```

> **注意**
>
> 小程序端更新的数据，需要放在 `data` 字段内。

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire")
    .update({
      // 表示将 done 字段置为 true
      done: true
    });
  return {
    res
  };
};
```

除了用指定值更新字段外，数据库 API 还提供了一系列的更新指令用于执行更复杂的更新操作，更新指令可以通过 `db.command` 取得：

更新指令	说明

set	设置字段为指定值
remove	删除字段
inc	原子自增字段值
mul	原子自乘字段值
push	如字段值为数组，往数组尾部增加指定值
pop	如字段值为数组，从数组尾部删除一个元素
shift	如字段值为数组，从数组头部删除一个元素
unshift	如字段值为数组，往数组头部增加指定值

例如我们可以将一个待办事项的进度+10%。示例代码如下：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .update({
    progress: _.inc(10)
  })
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .update({
    data: {
      // 表示指示数据库将字段自增 10
```

```
      progress: _.inc(10)
    }
  })
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

const _ = db.command;

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire")
    .update({
      // 表示指示数据库将字段自增 10
      progress: _.inc(10)
    });
  return {
    res
  };
};
```

用 `inc` 指令而不是取出值、加 10 再写进去的好处在于这个写操作是个原子操作，不会受到并发写的影响，例如同时有两名用户 A 和 B 取了同一个字段值，然后分别加上 10 和 20 再写进数据库，那么这个字段最终结果会是加了 20 而不是 30。如果使用 `inc` 指令则不会有这个问题。

如果字段是个数组，那么我们可以使用 `push`、`pop`、`shift` 和 `unshift` 对数组进行原子更新操作，例如给一条待办事项加多一个标签：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();
```

```
const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .update({
    tags: _.push("mini-program")
  })
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .update({
    data: {
      tags: _.push("mini-program")
    }
  })
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
var db = app.database();

const _ = db.command;

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire")
    .update({
      tags: _.push("mini-program")
    });
```

```
  return {
    res
  };
};
```

更完整详细的更新指令请参见 数据库文档 。

替换更新

如果需要替换更新一条记录，可以在记录上使用 set 方法，替换更新意味着用传入的对象替换指定的记录：

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
  .set({
    description: "learn cloud database",
    due: new Date("2018-09-01"),
    tags: ["cloud", "database"],
    style: {
      color: "skyblue"
    },
    location: new db.Geo.Point(113, 23),
    done: false
  })
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const _ = db.command;
db.collection("todos")
  .doc("todo-identifiant-aleatoire")
```

```
  .set({
    data: {
      description: "learn cloud database",
      due: new Date("2018-09-01"),
      tags: ["cloud", "database"],
      style: {
        color: "skyblue"
      },
      location: new db.Geo.Point(113, 23),
      done: false
    }
  })
  .then((res) => {
    console.log(res.data);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

const _ = db.command;
exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire")
    .set({
      description: "learn cloud database",
      due: new Date("2018-09-01"),
      tags: ["cloud", "database"],
      style: {
        color: "skyblue"
      },
      location: new db.Geo.Point(113, 23),
      done: false
    });
  return {
    res
  };
};
```

如果指定 ID 的记录不存在，则会自动创建该记录，该记录将拥有指定的 ID。

批量更新

通过 `where().update()` 可以找到符合查询条件的文档列表，并进行更新（服务端 node-sdk 同时支持只更新找到的第一条文档）。

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

db.collection("todos")
  .where({ done: false })
  .update({
    // 表示将 集合中 done为false 的 文档中 done 字段置为 true
    done: true
  })
  .then((res) => {
    console.log(res.data);
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

db.collection("todos")
  .where({ done: false })
  .update({
    // data 传入需要局部更新的数据
    data: {
      // 表示将 done 字段置为 true
      done: true
    }
  })
  .then((res) => {
    console.log(res.data);
  });
```

> **注意**
> 小程序端更新的数据，需要放在 data 字段内。

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({});
// 1. 获取数据库引用
const db = app.database();

exports.main = async (event, context) => {
  const res1 = await db
    .collection("todos")
    .where({ done: false })
    .update({
      // 表示将 done 字段置为 true
      done: true
    });

  // 2. 部分场景 需批量查询后，仅更新找到的第一条
  const res2 = await db.collection("todos")
    .where({ done: true })
    .options({ multiple: false })
    .update({
      process: 100 // 仅将满足done字段为true 的文档列表中的 第一条文档，更新process字段为
100
    })
  return {
    res1,
    res2
  };
};
```

删除数据

最近更新时间：2023-09-13 15:43:11

在本节中我们还是沿用 读取数据 章节中使用的数据作为示例。

删除一条记录

对记录使用 remove() 方法可以删除该条记录。

> **说明**
> 客户端上只能删除符合权限的数据，具体请参见 权限控制 。

示例代码如下：

Web

```
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "xxxx"
});
var db = app.database();

db.collection("todos")
  .doc("doc-id")
  .remove()
  .then((res) => {
    console.log(res);
  });
```

小程序

```
const db = wx.cloud.database();

db.collection("todos")
  .doc("doc-id")
  .remove()
  .then((res) => {
    console.log(res);
  });
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({});
```

```
const db = app.database();

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .doc("todo-identifiant-aleatoire-2")
    .remove();
  return {
    res
  };
};
```

删除多条记录

如果需要删除多个数据，可通过 where 语句选取多条记录执行删除。
示例代码如下：

Web

```
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "xxxx"
});
var db = app.database();

db.collection("todos")
  .where({
    done: true
  })
  .remove()
  .then((res) => {
    console.log(res);
  });
```

小程序

```
const db = wx.cloud.database();

db.collection("todos")
  .where({
    done: true
  })
  .remove()
  .then((res) => {
    console.log(res);
  });
```

Node.js

```
// 使用了 async await 语法
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init();
const db = app.database();
const _ = db.command;

exports.main = async (event, context) => {
  const res = await db
    .collection("todos")
    .where({
      done: true
    })
    .remove();
  return {
    res
  };
};
```

备份与回档

最近更新时间：2025-04-23 18:18:43

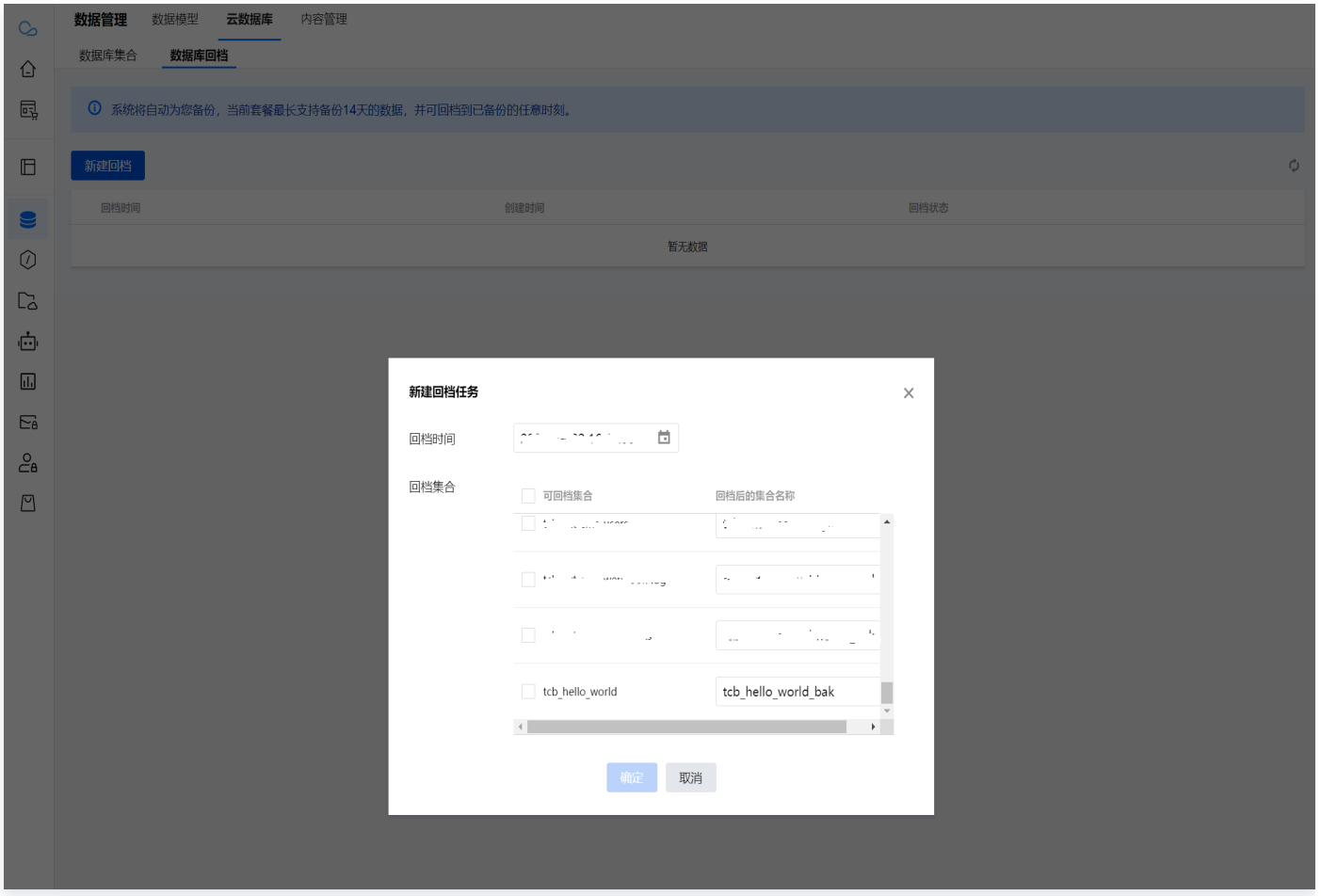
介绍

云开发提供了数据库回档功能。系统会自动开启数据库备份，并于每日凌晨自动进行一次数据备份，最长保存 14 天的备份数据。如有需要，开发者可在云控制台上通过新建回档任务将集合回档（还原）至指定时间点。

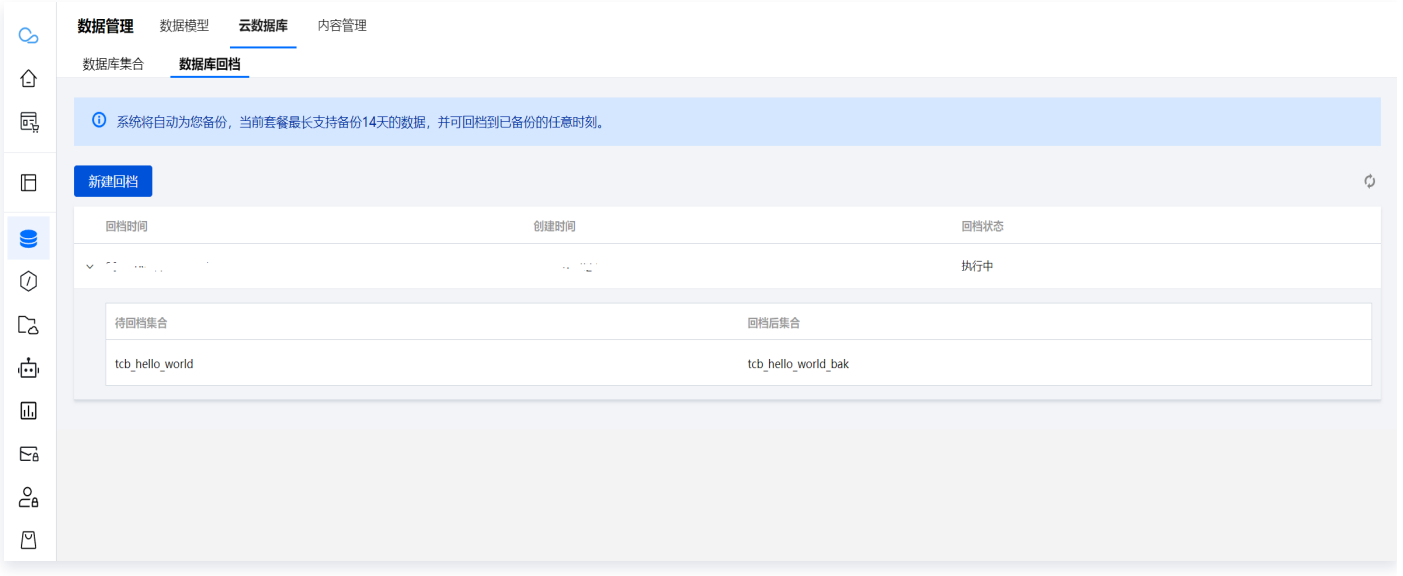
> **说明：**
> 回档期间，数据库的数据访问不受影响。回档完成后，开发者可在集合列表中看到原有数据库集合和回档后的集合。

新建回档

1. 登录 云开发平台，进入**云数据库**菜单，单击顶部**云数据库**页签。
2. 单击**数据库回档**，**新建回档**任务。
3. 单击**新建回档**后，可选择所需回档的时间点和需要回档的集合。请注意：
   - 一次回档任务只能设置一个回档时间，所有待回档集合的回档时间都以此时间点为准。
   - 一次回档任务可选择多个集合，支持全选。
   - 每个待回档集合都可单独设置回档后的集合名称。
   - 系统会默认生成回档后的集合名称，生成规则为：待回档集合名称_bak。
   - 回档后集合名称不可与已有集合名称重复。



4. 单击确定后，开发者可在数据库回档页面查看回档进度。请注意：
   - 为避免数据冲突，当前有回档任务在执行时，将无法创建新的回档任务。
   - 回档完成后，开发者可在集合列表中看到原有数据库集合和回档后的集合。



数据库事务

最近更新时间：2024-08-20 14:58:51

云开发支持数据库事务，并保证事务的 ACID 特性。本文将介绍数据库事务的使用场景、案例、原理以及实践教程，来帮助开发者完成更复杂的业务需求。

使用场景

云开发提供的云数据库是基于文档的非关系型数据库。不同于传统的关系型数据库，开发者可以直接在单文档中嵌套子文档，以描述更复杂的数据结构。在大多数场景中，单文档完全可以满足需求。但在一些场景中，使用数据库事务的优势更明显：

- 从传统关系型数据库迁移到云开发：数据模型平滑迁移，业务代码改造成本低。
- 涉及多个文档或多个集合的业务流程：保证一系列读写操作完全成功或者完全失败，防止出现中间态。

> **说明：**
> 目前数据库事务只支持在服务端运行，只有服务端 SDK 支持事务。

支持的方法

目前支持 4 种操作事务流程的方法：

API	说明
startTransaction	发起事务
commit	提交事务
rollback	回滚事务
runTransaction	自动提交事务

目前支持事务中 5 种读写数据的方法：

API	说明
get	查询文档
add	插入文档
delete	删除文档
update	更新文档
set	更新文档，文档不存在时，会自动创建

您可以前往 数据库事务，查看更详细的 API 说明。

使用案例

为了帮助您快速体会到数据库事务的重要性和便捷性，这里以**清空购物车**的需求为例，介绍数据库事务在复杂业务场景中的使用。
假设商品数据放在了 `goods` 集合中，如下所示：

```
[
  {
    "_id": "item1",
    "inventory": 20,
    "name": "商品a",
    "price": 10
  },
  {
    "_id": "item2",
    "inventory": 10,
    "name": "商品b",
    "price": 5
  }
]
```

用户的数据放在了 `users` 集合中，如下所示：

```
[
  {
    "_id": "user1",
    "balance": "1000", // 账户余额
    "cart": [
      // 购物车
      {
        "id": "item1", // 商品id
        "num": 1 // 购买数量
      },
      {
        "id": "item2",
        "num": 1
      }
    ],
    "name": "用户1"
  }
]
```

当用户 1 清空购物车时，业务的整体流程是：

1. 计算购物车中的商品总价。
2. 减少对应商品的库存。
3. 更新用户 1 的账户余额。
4. 清空用户 1 的购物车数据。

我们将这些操作放入一个事务中执行，代码实现如下：

```
// Node.js 环境
const cloudbase = require('@cloudbase/node-sdk')

const app = cloudbase.init({})
```

```
const db = app.database()

exports.main = async (event, context) => {
  const userId = 'user1'
  const transaction = await db.startTransaction()
  const usersCollection = transaction.collection('users')
  const goodsCollection = transaction.collection('goods')

  // 1. 获取用户信息
  const user = await usersCollection.doc(userId).get()

  // 2. 获取购物车数据和对应的商品信息
  const { cart, balance } = user.data
  const goods = []
  for (const { id } of cart) {
    const good = await goodsCollection.doc(id).get()
    goods.push(good.data)
  }

  let totalPrice = 0
  for (let i = 0; i < cart.length; ++i) {
    // 3. 计算购物车中的商品总价
    totalPrice += cart[i].num * goods[i].price
    // 4. 更新商品库存
    await goodsCollection.doc(goods[i]._id).set({
      inventory: goods[i].inventory - cart[i].num
    })
  }

  await usersCollection.doc(userId).set({
    balance: balance - totalPrice, // 5. 更新账户余额
    cart: [] // 6. 完成购买后，清空购物车
  })

  await transaction.commit()

  // 从数据库中查询最新的用户，库存信息
  const usersInfo = await db.collection('users').get()
  const goodsInfo = await db.collection('goods').get()
  return {
    usersInfo,
    goodsInfo
  }
}
```

注意：

为了更简洁地体现事务在复杂业务场景中的优势，案例中没有对库存、余额等信息进行额外的代码检查。

从“清空购物车”的案例中可以看出，数据库事务极大地节省了开发的成本，避免引入复杂的数据库设计，让开发者的精力更聚焦于当前业务。

原理介绍

本小节会介绍数据库事务的底层原理，以加深您对数据库事务的理解，更好地使用数据库事务。

快照隔离

在调用 `db.startTransaction()` 开始事务之后，并没有立即生成一份快照，快照是在第一次读之后才会生成。在没有调用 `transaction.commit()` 提交事务前，所有的读写操作都是在快照上进行，不会影响文档原本的数据。在成功提交事务后，快照上的数据才会落盘，相关文档数据完成更新。
假设对于商品 A 来说，它的库存还有 13 个：

```
{
    "id": "xxxxxx",
    "name": "商品A",
    "inventory": 13
}
```

如果消费者发起了购买商品 A 的事务，在购买事务未成功提交前，所有的变更都是在快照上进行，不会影响商品 A 的数据，所以其他消费者看到的商品 A 的库存依然是 13。

锁与写冲突

当事务修改文档时，会锁定相关文档，使其不受其他更改的影响，直到事务结束。因此，外部的普通写入，会被阻塞。
如果一个事务无法获取到试图修改的文档的锁，可能是因为另一个事务已经持有该锁，那么事务会终止，并出现写冲突。

> **说明：**
> 读取文档的操作不需要与文档修改相同的锁。这意味着即使当前事务对某个文档进行了未提交的写操作，其他事务仍然可以读取这个文档的内容。根据“快照隔离”，读取的文档内容是文档未提交的状态。

因此，为了使代码更健壮，推荐在进行事务操作时，使用 `try-catch` 来捕获异常。代码示例如下：

```
const cloudbase = require("@cloudbase/node-sdk");

const app = cloudbase.init({
  env: "xxx"
});
// 1. 获取数据库引用
const db = app.database();

// 2. 模拟事务操作
async function main() {
  try {
    const transaction = await db.startTransaction();
    // ... ...
    // 涉及文档更改的事务操作
    // ... ...
```

```
    await transaction.commit();
  } catch (error) {
    // 发生写冲突时，进行异常处理
    console.error(error.message);
  }
}
```

实践教程

为了更好的使用事务，开发者应该遵循几种实践教程：

- 避免创建长时间运行的事务，或者执行过多操作。因为事务会创建快照，所有的后续写入操作都会在缓存中积累，直到事务提交或者终止。当一个事务中的操作过多时，可能会影响数据库的性能。当事务运行时间过长（通常指的是超过 30s），事务可能会被自动终止。推荐将事务拆分成更小的事务，以防止这些情况的发生。
- 避免在进行 DDL 操作时（例如：创建索引、删除数据库），进行事务操作。在 DDL 操作期间，尝试访问相关资源的事务无法获得锁，从而导致新事务终止。
- 在进行使用云开发提供的 SDK 操作事务时，推荐配合 `try-catch` 来捕获异常，从而尽早发现和处理写冲突、网络异常等问题。

实时推送

最近更新时间：2025-06-13 11:49:01

云开发数据库支持监听集合中符合查询条件的数据的更新事件。

建立监听

使用 `watch()` 方法即可建立监听，并且返回 `watcher` 对象，用于关闭监听。
符合条件的文档有任何变化，都会触发 `onChange` 回调。

Web

```
const cloudbase = require("@cloudbase/js-sdk");

const app = cloudbase.init({
  env: "xxxx"
});
// 1. 获取数据库引用
var db = app.database();

const watcher = db
  .collection("todos")
  .where({
    // query...
  })
  .watch({
    onChange: function (snapshot) {
      console.log("snapshot", snapshot);
    },
    onError: function (err) {
      console.error("the watch closed because of error", err);
    }
  });
```

小程序

```
// 1. 获取数据库引用
const db = wx.cloud.database();

const watcher = db
  .collection("todos")
  .where({
    // query...
  })
  .watch({
    onChange: function (snapshot) {
```

```
      console.log("snapshot", snapshot);
    },
    onError: function (err) {
      console.error("the watch closed because of error", err);
    }
  });
```

关闭监听

调用 `watcher.close()` 即可关闭监听。

Web

```
watcher.close();
```

小程序

```
watcher.close();
```

说明：

- 适用场景（实时推送适用于广播场景，同时具有连接数限制，对于高并发单播场景不建议使用）：
  - 单播场景：不同用户 watch 的 where 条件均不同。
  - 广播场景：所有用户 watch 的 where 条件均相同。
- 系统限制：
  - 监听记录数限制。

    一次监听的记录数上限为5000，若超出上限会抛错并停止监听。监听过大量的数据时初始化会较慢，对监听效率也有影响，如果预期监听发起时少于5000，但后续有可能超过5000，请注意在即将超过时重新监听并保证不超过5000。
  - 最大连接数限制。

    从2025年6月30日起，实时推送的最大连接数限制跟随套餐，具体限制数值可以查看 套餐及价格文档；如有更大连接数上限需求的活动，请至少提前30天 联系我们 。
  - 注意集合权限设置。

    集合的读权限设置在实时数据推送里同样生效，如果权限是设置为仅可读用户自己的数据，则监听的时候无法监听到非用户自己创建的数据。

聚合搜索

最近更新时间：2023-09-13 16:55:04

聚合主要用来处理数据（统计平均值，求和等）并返回计算后的数据结果。

流水线/管道

聚合是一个流水线式的批处理作业，初始文档经过多个阶段的流水线处理后，得到转换后的聚合结果。
假设已有一个集合 books，其中包含以下格式记录：

```
[
  {
    "_id": "xxx",
    "category": "Novel",
    "name": "The Catcher in the Rye",
    "onSale": true,
    "sales": 80
  }
]
```

聚合示例如下：

```
// 云函数端示例
const cloudbase = require("@cloudbase/node-sdk")
const app = cloudbase.init()
const db = app.database()
const $ = db.command.aggregate

exports.main = async (event, context) => {
    const res = await db.collection('books').aggregate()
    .match({
        onSale: true // 是否正在出售
    })
    .group({
        // 按 category 字段分组
        _id: '$category',
        // 让输出的每组记录有一个 avgSales 字段，其值是组内所有记录的 sales 字段的平均值
        avgSales: $.avg('$sales')
    })
    .end()
    return {
        res
    }
}
```

说明

> **第一阶段：match** 阶段过滤出了集合中的文档数据（ `onSale:true` 表示找出正在出售的书籍）并传给下一个阶段。
> **第二阶段：group** 阶段基于 `category` 字段进行分组，并统计出每组中所有记录的 `sales` 字段平均值。

API 及操作符

参考 Node.js SDK API 文档，一览所有的聚合阶段 API 及 操作符 。

优化执行

利用索引

match 和 sort 如果是在流水线的开头的话是可以利用索引的。geoNear 也可以利用地理位置索引，但要注意的是，geoNear 必须是流水线的第一个阶段。

尽早缩小数据集

在不需要集合的全集情况下，应该尽早的通过 match 、limit 和 skip 缩小要处理的记录数量。

注意事项

除了 match 阶段，在各个聚合阶段中传入的对象中，可使用的操作符都是聚合操作符，需要特别注意的是，match 进行的是查询匹配，因此语法同普通查询 where 的语法，用的是普通查询操作符。

安全规则

最近更新时间：2024-01-22 11:07:11

为了保护用户的数据安全，云开发提供更灵活、可扩展、更细粒度的安全规则能力，开发者可以在云后台或者小程序开发工具上自定义安全规则，限制 C 端用户对数据库的访问权限。

> **注意：**
> 与之前数据库的权限设置一样，安全规则对数据库的权限控制是文档级别的访问控制，对文档的属性访问控制目前还不支持。
> **示例：**
> 安全规则仅限制 C 端用户的访问请求，对于开发者的请求（例如通过云函数的访问请求）不做任何限制。

配置

每个集合可以单独配置安全规则，通过 `json` 来描述安全规则配置，示例配置如下：

```
{
  "read": "auth.uid==doc.uid",
  "write": "doc.name=='zzz'"
}
```

`json` 配置的 `key`，是用户的操作类型，`value` 是一个表达式。当表达式执行结果的值是 `true` 的时候，用户的操作允许执行；否则，用户的操作则不被允许。

操作类型	说明	默认值
read	读文档	false
write	写文档，可以细分为 create、update、delete	false
create	新建文档	无
update	更新文档	无
delete	删除文档	无

说明

如果开发者没有详细指定写操作具体类型的规则时，默认读取 `write` 的规则。例如说开发者的安全规则配置如下：

```
{
  "read": ".....",
  "write": "....",
  "create": "...."
}
```

用户的新建文档操作会读取 `create` 的对应规则，而 `update` 操作会读取 `write` 的规则。

表达式

表达式是伪代码的语句，配置的时候不能过长，单条表达式限制 1024 字符。

变量

全局变量

变量名	类型	说明
request	Request	请求挂载的根对象
auth	Auth	用户登录信息，字段说明参见下文
now	Number	当前时间
doc	Any	文档数据或查询条件

Request

字段名	类型	说明
data	Object	请求数据，create，update 时存在，代表请求 data

Auth

字段名	类型	说明
loginType	String	登录方式，微信小程序过来的请求是没有此值的；web 登录下才有此值。
uid	String	用户唯一 ID，微信小程序的请求没有此值
openid	String	用户 openid，仅在微信登录方式下存在值

LoginType 枚举

枚举值	登录方式说明
WECHAT_PUBLIC	微信公众号
WECHAT_OPEN	微信开放平台
ANONYMOUS	匿名登录
EMAIL	邮箱登录
CUSTOM	自定义登录

运算符

运算符	说明	示例	示例解释(集合查询)
==	等于	auth.uid == 'zzz'	用户的 uid 为 zzz
!=	不等于	auth.uid != 'zzz'	用户的 uid 不为 zzz

>	大于	doc.age > 10	查询条件的 age 属性大于 10
>=	大于等于	doc.age >= 10	查询条件的 age 属性大于等于 10
<	小于	doc.age < 10	查询条件的 age 属性小于 10
<=	小于等于	doc.age <= 10	查询条件的 age 属性小于等于 10
in	存在于集合中	auth.uid in ['zzz','aaa']	用户的 uid 是['zzz','aaa']中的一个
!(xx in [])	不存在于集合中，使用 in 的方式描述 `!(a in [1,2,3])`	!(auth.uid in ['zzz','aaa'])	用户的 uid 不是['zzz','aaa']中的任何一个
&&	与	auth.uid == 'zzz' && doc.age>10	用户的 uid 为 zzz 并且查询条件的 age 属性大于 10
||	或者	auth.uid == 'zzz' || doc.age>10	用户的 uid 为 zzz 或者查询条件的 age 属性大于 10
.	对象元素访问符	auth.uid	用户的 uid
[]	数组访问符属性	get('database.collection_a.user')[auth.uid] == 'zzz'	`collection_a` 集合中 `id` 为 `user` 的文档，key 为用户 uid 的属性值为 zzz

函数

目前仅支持 get 函数，唯一的参数必须为 `database.集合名称.文档id` 。通过访问其它文档的数据来判断用户操作是否符合安全规则。

get

`get(path: string): Document | null | undefined`

根据参数获取指定 doc 内容，返回 undefined 或 null 表示文档不存在，否则为 Document 对象，表示查询得到的数据。

入参	类型	描述
path	String	非空，格式为 `database.集合名称.文档id` 的字符串，值可以通过多种计算方式得到，例如使用字符串模板进行拼接：（ `database.${doc.collction}.${doc._id}` ）

示例

1. 用户的权限是写在一个独立的文档，用一个数值表示用户的权限范围：

```
{
  "read": "get('database.test.123')[auth.uid] in [1,2,3]",
  "delete": "get('xxxx')[auth.uid] == 1 && doc.user in ['ersed','sfsdf'] "
}
```

2. 集合 A 包含 shopId、orderId 关联关系，集合 B 包含 owner，shopId 关联关系，对集合 A 查询，希望限制只查到当前用户有权限的 shop 的订单：

```
{
  "read:": "auth.openid in get(`database.B.${doc.shopId}`).owner"
}
```

限制

- get 参数中存在的变量 doc 需要在 query 条件中以 == 或 in 方式出现，若以 in 方式出现，只允许 in 唯一值， 即 doc.shopId in array, array.length == 1。
- 一个表达式最多可以有 `3` 个 `get` 函数，最多可以访问 `10` 个不同的文档。
- get 函数的嵌套深度最多为 2， 即 `get(get(path))` 。

> **说明**
> 在未使用变量的情况下，每个 `get` 会产生一次读操作，在使用变量时，每个 get，每个变量值会产生一次读操作，例如：规则 `get(database.collection.${doc._id}).test` ，在查询 `_.or([{_id:1},{_id:2},{_id:3},{_id:4},{_id:5}])` 下会产生 5 次读取。系统会对同 doc，同 field 的读取进行缓存。

查询

有效的查询

在实际使用的过程中，查询主要分为两种，**指定文档 ID 的查询和集合查询**。

- **指定文档 ID 的查询：** 通过 `doc` 条件指定单个文档 ID。
- **集合查询：** 可以是通过 `where` 条件的查询或者是聚合搜索 `match` 限制条件的查询，如果是聚合搜索，只匹配第一个 `match` 限制条件。

> **说明**
> 查询条件中，如果 `key` 是 `_openid` 并且 `value` 是 `{openid}` ，或者 `key` 是 `uid` 并且 `value` 是 `{uid}` ，则在服务端自动将 value 替换为实际的值。

集合查询

对于集合查询，查询条件必须是安全规则的子集， `doc` 此时表示的是查询条件。这个子集是指规则上所有可能的子集，而不是实际数据的子集。
操作类型包括： `read` 、 `update` 、 `delete` 。

示例

```
// colleciton_a 的安全规则的配置
{
    "read":"doc.age>10"
}

// 符合安全规则
let queryRes = db.collection('colleciton_a').where({
    age:_.gt(10)
}).get()
```

```
// 不符合安全规则
let queryRes = db.collection('colleciton_a').where({
    age:_.gt(8)
}).get

// 符合安全规则
let res = await db.collection('collection_a').aggregate().match({
  age: _.gt(10)
}).project({
  age: 1
}).end()

// 不符合安全规则
let res = await db.collection('collection_a').aggregate().match({
  age: _.gt(8)
}).project({
  age: 1
}).end()
```

指定文档 ID 查询（需迁移改造）

操作类型包括：`read`、`update`、`delete`、`create`。

- 如果操作类型是 `create`，会校验写入的数据是否符合安全规则的限制条件。
- 如果操作类型是 `read`、`update`、`delete`，同时安全规则包含了 doc 的限制，则会先从 db 中读取一次文档数据，然后判断是否符合安全规则。

> **说明**
>
> - 对于 `update` 操作，只会校验存在于数据库中已有的文档数据，不会对写入的数据进行校验；同时不保证这个操作的原子性。
> - 由于安全规则要求查询条件是安全规则的子集（所有对 doc 的限制必须在查询条件中出现且查询条件限制范围是规则限制的子集，例如：查询 a>20 是规则 a>10 的子集），同时摒弃了旧有权限配置的隐式默认行为，因此启动安全规则需要开发者注意以下升级/兼容处理。
> - 因 `doc` 操作（doc.get, doc.set 等）是仅指定 _id 进行的操作，其查询条件仅包含 `{_id:"xxx"}`，因此大部分情况下并不会满足安全规则的子集要求（除非在 "read": true 下进行读操作或在 "write": true 的情况下进行写操作），因此需要转换为等价的、查询条件包含安全规则或是其子集的形式。

示例

```
// colleciton_a 的安全规则的配置
{
    "read":"doc._openid == auth.openid"
}

// 文档id='ccc'的数据是
{
```

```
  age:12
}

// 不符合安全规则，未满足子集需求
let queryRes = db.collection('colleciton_a').doc('ccc').get()

// 符合安全规则，改写为 where 查询
let queryRes = db.collection('colleciton_a').where({_id: "ccc", _openid: "
{openid}"}).get()
```

说明

因基础权限控制下查询条件可以不传 _openid，而安全规则权限要求显示传入以保证查询条件符合安全规则，因此所有查询条件均需传入 openid。可使用模板 "{openid}" 或 "{uid}" 来代指当前登录用户的 openid 或 uid。

支持的数据库 command

logic command

command	description
or	|| 逻辑或
and	&& 逻辑与

query command

command	description
eq	==
ne/neq	!=
gt	>
gte	>=
lt	<
lte	<=
in	in
nin	!(in [])

update command

command	description
set	覆盖写，{key: set(object)}
remove	删除字段，{key: remove()}

计费

安全规则本身不收费，但是安全规则额外的数据访问会统计到计费中。

- get 函数会产生额外的数据访问。
- 指定文档 ID 查询的所有写操作会产生一次数据访问。

权限设置对比

权限类型	设置示例
所有用户可读，仅创建者及管理员可写	`{ "read": true, "write": "doc._openid == auth.openid", // 登录方式为微信 "write": "doc._openid == auth.uid" // 登录方式为非微信 }`
仅创建者及管理员可读写	`{ "read": "doc._openid == auth.openid", //登录方式为微信 "read": "doc._openid == auth.uid", // 登录方式为非微信 "write": "doc._openid == auth.openid", //登录方式为微信 "write": "doc._openid == auth.uid" // 登录方式为非微信 }`
所有用户可读，仅管理员可写	`{ "read": true, "write": false }`
仅管理员可读写	`{ "read": false, "write": false }`

角色权限控制

您可以利用 CloudBase 的数据模型以及自定义的安全规则在您的应用中实现基于角色的访问权限控制。

示例

假设您正在构建一款协作式撰文应用，用户可以按照以下安全要求在其中撰写**故事**和**评论**。每个故事都有一名**所有者**，故事可共享给**撰写者**。

- 撰写者：除了拥有评论者所拥有的全部访问权限之外，还可以编辑故事内容。
- 所有者：可以编辑故事的任意部分，并且可以控制其他用户的访问权限。

> **说明**
> 普通用户只能查看故事和评论，写自己的评论，不能编辑故事。

数据结构

假设您的应用有一个 `stories` 集合，其中每个文档代表一个故事。
还有一个 `comments` 集合，其中每个文档代表一条针对该故事的评论。
一个 `roles` 集合，每个集合包含一个故事的用户的角色。
为了跟踪访问角色，需要添加一个 roles 字段，用来将用户 ID 映射至角色。

stories 集合

每一个 `story` 文档数据如下：

```
{
  "id": `storyid`,
  "title": "A Great Story",
  "content": "Once upon a time ..."
}
```

roles 集合

每一个 `role` 文档数据如下：

```
{
  "id": `storyid`,
  "roles": {
    "alice": "owner",
    "bob": "writer",
    "david": "writer"
    // ...
  }
}
```

comments 集合

每一个 `comment` 文档数据如下，评论仅包含三个字段，留言者的用户 ID 、内容、storyid。

```
{
  "id": `commentId`,
  "storyid": `storyid`,
  "user": "alice",
  "content": "I think this is a great story!"
}
```

规则

既然数据库中记录有用户角色，那么便需要编写安全规则来进行角色验证。下列规则假定应用使用的是 TCB 身份验证，所以 auth.uid 变量为用户 ID。

roles 添加规则

管理者可以更改角色，允许故事撰写者读取角色：

```
{
  "write": "doc.roles[auth.uid] === 'owner' ",
  "read": "doc.roles[auth.uid] in ['owner', 'writer']"
}
```

stories 添加规则

管理者和故事撰写者可以更改故事，其它人可以读取故事：

```
{
  "read": true,
  "write": "get(`database.roles.${doc.id}`).roles[auth.uid] in ['owner', 'writer'] "
}
```

comments 规则

允许所有人发表评论。
只有评论的拥有者，可以更新和删除评论：

```
{
  "read": true,
  "create": true,
  "update": "doc.user == auth.uid",
  "delete": "doc.user == auth.uid "
}
```

云存储

概览

最近更新时间：2024-09-03 10:41:22

CloudBase 云存储提供稳定、安全、低成本、简单易用的云端存储服务，支持任意数量和形式的非结构化数据存储，例如图片、文档、音频、视频、文件等。
您可以通过控制台、CloudBase CLI、HTTP API、SDK 等方式将文件上传到云端存储空间内，并对云端文件进行**文件管理**、**权限管理**和**缓存设置**等。

主要特性

默认支持 CDN 加速

云存储无需进行繁杂的配置，默认支持 CDN 加速，并提供免费的 CDN 域名。CDN 会将云存储的内容分发至最接近用户的节点，直接由服务节点快速响应，可以有效降低用户访问延迟。

身份验证集成

CloudBase 云存储与 CloudBase 用户身份验证无缝集成，您可以结合用户身份认证和安全规则对云存储里的文件设置访问权限，例如云存储的文件只对已登录用户公开或仅限使用微信登录的用户访问等。

高可扩展性

CloudBase 云存储可以与 CloudBase 其它多种服务配合使用，例如：将文件的 fileID 存储到云数据库。

上传文件

最近更新时间：2023-09-13 16:55:04

您可以上传任意数量、格式的文件至 CloudBase 云存储，也可以自定义文件、目录的路径和名字。

说明

- 默认情况下，只有通过了 CloudBase 身份验证 的用户才可以向云存储空间上传文件，因此在用户端（例如 Web）上传文件时需先进行登录认证。
- 您也可以使用 自定义安全规则，为云存储设置更宽松或更严格的读写权限。

使用 SDK 可以向云存储空间上传文件，并返回该文件全局唯一标识 **fileID**。

Web

```
//第一步，引入 Web SDK，
import tcb from "@cloudbase/js-sdk";

//第二步，初始化
const app = tcb.init({
  env: "your-env-id"
});

/**
  第三步，登录鉴权流程，此处代码略，请参考：
  https://cloud.tencent.com/document/product/876/41728
*/

app
  .uploadFile({
    // 云存储的路径
    cloudPath: "dirname/filename",
    // 需要上传的文件，File 类型
    filePath: document.getElementById("file").files[0]
  })
  .then((res) => {
    // 返回文件 ID
    console.log(res.fileID);
  });
```

微信小程序

```
//需先使用 wx.cloud.init 初始化，小程序端无需再引入 SDK ，且免鉴权
wx.cloud
  .uploadFile({
    cloudPath: "example.png", // 上传至云端的路径
```

```
    filePath: "" // 小程序临时文件路径，需结合小程序相关 API 获取
  })
  .then((res) => {
    // 返回文件 ID
    console.log(res.fileID);
  });
```

Node.js

```
const tcb = require("@cloudbase/node-sdk");
const fs = require("fs");

const app = tcb.init();
app
  .uploadFile({
    cloudPath: "path/test.jpg",
    fileContent: fs.createReadStream("test.jpg")
  })
  .then((res) => {
    // 返回文件 ID
    console.log(res.fileID);
  });
```

说明

- 上传时文件名需要符合 文件名规范 。
- cloudPath 为云存储文件或文件夹的相对根目录的路径，为 **目录/文件名** 的形式，cloudPath 不需要以 `/` 开头。
- 如果将文件上传至同一路径则是覆盖写，默认情况下，不允许 A 用户覆盖写 B 用户的文件。

获取临时链接

最近更新时间：2023-06-12 15:49:33

您可以使用 fileID 换取云存储空间指定文件的 HTTPS 链接（云存储提供免费的 CDN 域名）。

说明

- 公有读的文件获取的 HTTPS 链接不会过期，例如默认情况下的权限就是公有读，获取的链接永久有效。
- 私有读的文件获取的 HTTPS 链接为临时链接，例如您可以结合用户身份认证和安全规则设置文件的权限为仅文件的上传创建者或管理员可读，此时只有通过了云开发身份验证的用户才有权限换取临时链接。
- 有效期可以动态设置，超过有效期再请求临时链接时会被拒绝，保证了文件的安全。
- 一次最多可以取50个，更多需分批处理。
- CDN 会默认为后缀 .js、.html、.css、.xml、.json、.shtml、.htm，大小为 256Byte - 2MB 范围内的资源开启 Gzip 压缩。

使用 SDK 调用 getTempFileURL 方法传入文件的 fileID ，就可以换取云存储空间指定文件的 HTTPS 链接。

Web

```
//第一步，引入 Web SDK，
import tcb from "@cloudbase/js-sdk";

//第二步，初始化
const app = tcb.init({
  env: "your-env-id"
});

//第三步，登录认证，下面非完整代码，需选择登录方式，具体可以参见 快速开始 > 登录与用户案例
const auth = app.auth({
  persistence: "local" //用户显示退出或更改密码之前的30天一直有效
});

app
  .getTempFileURL({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    // fileList 是一个有如下结构的对象数组
    // [{
    //    fileID: 'cloud://webtestjimmy-5328c3.7765-webtestjimmy-5328c3-1251059088/腾讯云.png', // 文件 ID
    //    tempFileURL: '', // 临时文件网络链接
    //    maxAge: 120 * 60 * 1000, // 有效期
    // }]
    console.log(res.fileList);
  });
```

微信小程序

```
//需先使用 wx.cloud.init 初始化，小程序端无需再引入 SDK ，且免鉴权
wx.cloud
  .getTempFileURL({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    // fileList 是一个有如下结构的对象数组
    // [{
    //    fileID: 'cloud://webtestjimmy-5328c3.7765-webtestjimmy-5328c3-1251059088/腾讯云.png', // 文件 ID
    //    tempFileURL: '', // 临时文件网络链接
    //    maxAge: 120 * 60 * 1000, // 有效期
    // }]
    console.log(res.fileList);
  });
```

Node.js

```
const tcb = require("@cloudbase/node-sdk");
const app = tcb.init();

app
  .getTempFileURL({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    // fileList 是一个有如下结构的对象数组
    // [{
    //    fileID: 'cloud://webtestjimmy-5328c3.7765-webtestjimmy-5328c3-1251059088/腾讯云.png', // 文件 ID
    //    tempFileURL: '', // 临时文件网络链接
    //    maxAge: 120 * 60 * 1000, // 有效期
    // }]
    console.log(res.fileList);
  });
```

删除文件

最近更新时间：2023-09-13 15:19:32

默认情况下，只有通过了 CloudBase 身份验证 的用户才可以向云存储空间上传文件，因此在用户端（例如 Web）删除文件时需先进行登录认证。

说明

- 单次调用最多可以删除 50 个文件，更多需分批处理。
- 默认情况下，只有该文件的上传创建者和管理员才有权删除相应的文件，不允许 A 用户删除 B 用户的文件。
- 您也可以使用 自定义安全规则，为云存储设置更宽松或更严格的读写权限。

使用 SDK 调用 deleteFile 方法可以删除云存储空间单个或多个指定文件。

Web

```
//第一步，引入 Web SDK，
import tcb from "@cloudbase/js-sdk";

//第二步，初始化
const app = tcb.init({
  env: "your-env-id"
});

//第三步，登录认证，下面非完整代码，需选择登录方式，具体可以参见快速开始 > 登录与用户案例
const auth = app.auth({
  persistence: "local" //用户显示退出或更改密码之前的30天一直有效
});

app
  .deleteFile({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    console.log(res.fileList);
  });
```

微信小程序

```
//需先使用 wx.cloud.init 初始化，小程序端无需再引入 SDK ，且免鉴权
wx.cloud
  .deleteFile({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    console.log(res.fileList);
```

```
  });
```

Node.js

```
const tcb = require("@cloudbase/node-sdk");
const app = tcb.init();

app
  .deleteFile({
    fileList: ["cloud://a/b/c", "cloud://d/e/f"]
  })
  .then((res) => {
    console.log(res.fileList);
  });
```

下载文件

最近更新时间：2025-03-07 10:48:12

默认情况下，CloudBase 云存储内的文件对所有用户公开可读。

> **说明：**
> 您也可以使用 自定义安全规则，为云存储设置更宽松或更严格的读写权限。

使用 SDK 可以下载云存储空间里的文件，调用时只需传入云存储文件全网唯一的 fileID 。

Web

```
//第一步，引入 Web SDK
import tcb from "@cloudbase/js-sdk";

//第二步，初始化
const app = tcb.init({
  env: "your-env-id"
});

/**
  第三步，登录鉴权流程，此处代码略，请参考：
  https://cloud.tencent.com/document/product/876/41728
*/

app
  .downloadFile({
    fileID: "cloud://a/b/c"
  })
  .then((res) => {
    console.log(res);
  });
```

微信小程序

```
// 需先使用 wx.cloud.init 初始化，小程序端无需再引入 SDK，且免鉴权
wx.cloud
  .downloadFile({
    fileID: "cloud://a/b/c" // 文件 ID
  })
  .then((res) => {
    // 返回临时文件路径
    console.log(res.tempFilePath);
  });
```

Node.js

```
const tcb = require("@cloudbase/node-sdk");

const app = tcb.init({
  env: "your-env-id"
});

app
  .downloadFile({
    fileID: "cloud://a/b/c"
  })
  .then((res) => {
    // fileContent 类型为 Buffer
    console.log(res.fileContent);
  });
```

说明：
如果您需要在浏览器中可以直接下载云存储里的文件，或将云存储作为图床，可以参考 获取临时链接 。

文件名命名限制

最近更新时间：2023-09-13 16:55:04

- 不能为空。
- 不能以/开头。
- 不能出现连续/。
- 编码长度最大为 850 个字节。
- 推荐使用大小写英文字母、数字，即[a-z，A-Z，0-9]和符号 -，!，_，.，* 及其组合。
- 不支持 ASCII 控制字符中的字符上(↑)，字符下(↓)，字符右(→)，字符左(←)，分别对应 CAN(24)，EM(25)，SUB(26)，ESC(27)。
- 如果用户上传的文件或文件夹的名字带有中文，在访问和请求这个文件或文件夹时，中文部分将按照 URL Encode 规则转化为百分号编码。
- 不建议使用的特殊字符: ` ^ " \ { } [ ] ~ % # \ > < 及 ASCII 128-255 十进制。
- 可能需特殊处理后再使用的特殊字符: ， : ; = & $ @ + ? (空格) 及 ASCII 字符范围：00-1F 十六进制 (0-31 十进制) 以及 7F (127 十进制) 。

云存储安全规则

最近更新时间：2025-05-13 09:59:22

安全规则是在基础权限控制基础上，更高级、灵活、可扩展的一种权限控制方式，由**身份验证**、**授权**和**安全规则表达式**三部分构成。云存储安全规则可用于确定哪些人对云存储桶中存储的文件拥有读取和写入权限，也可用于文件中包含的元数据校验。

说明：

- 您可以在云开发控制台对云存储进行安全规则权限的设置，具体请参见 控制台可视化管理 。
- 云开发控制台和服务端始终有所有文件读写权限，安全规则的配置仅对客户端（小程序端、Web 等）发起的请求有效。
- 修改更新安全规则，权限生效需要1 – 3分钟，请耐心等待。
- 发布之前，请务必评估您的规则，确保它们可以为您的应用提供所需的更高级别的安全性。如果发布应用时设置的是 public 规则，可能会导致您存储的数据遭到意外访问或未经授权的访问。

基础权限控制与云存储安全规则

- 前往 云开发平台，在左侧菜单选择**对象存储**，可以在**云储存**界面进行**权限设置**。

- 云存储提供基础的权限控制，默认为“所有用户可读，仅创建者可读写”，基础权限控制都可以通过安全规则权限控制得到。

权限类型	设置示例
所有用户可读，仅创建者及管理员可写	`{ "read": true, "write": "resource.openid == auth.openid", // 登录方式为微信 "write": "resource.openid == auth.uid" // 登录方式为非微信`

	} }

仅创建者及管理员可读写	{
	"read": "resource.openid == auth.openid", //登录方式为微信
	"read": "resource.openid == auth.uid", // 登录方式为非微信
	"write": "resource.openid == auth.openid", //登录方式为微信
	"write": "resource.openid == auth.uid" // 登录方式为非微信
	}

所有用户可读，仅管理员可写	{
	"read": true,
	"write": false
	}

仅管理员可读写	{
	"read": false,
	"write": false
	}

身份验证

云存储安全规则结合集成的用户身份验证可以实现身份校验能力，开发者可以根据用户身份信息进行精准的资源访问控制。

示例

当 C 端用户登录后，安全规则中 `auth` 变量会变成一个包含该用户唯一 ID（`auth.uid`）和登录方式（`auth.loginType`）的对象。反之用户未登录，则 `auth` 的值为 null。通过 `auth` 规则可以实现对每个用户的数据访问控制。
经过身份认证的用户发起请求时，系统会使用用户唯一 id `uid` 及用户登录方式 `loginType` 填充 `auth` 变量。
通过 `auth` 变量，可以用以下常用方式来根据身份对文件访问进行控制：

- 公开：不判断 `auth` 值。
- 只对已登录用户公开：检查 `auth` 不为 `null` 。
- 用户私有：检查 `auth.uid` 是否等于资源 `openid` 。
- 仅对某种特殊的登录方式进行判断，限制匿名登录用户访问，检查 `auth.loginType` 不为 `ANONYMOUS` 。

授权

识别用户身份只是安全工作的一部分。在知道用户的身份后，开发者需要一种方法来控制该用户对云存储中的文件的访问权限。

> **说明：**
> 云存储支持存储桶级别的授权规则，通过设置安全规则表达式可以对云存储空间内的所有文件的读写操作进行限制。

安全规则表达式

安全规则表达式通过 `json` 来描述，允许在满足条件时执行 `read` 及 `write` 操作，示例配置如下：

```
{
  "read": boolean | condition expression string,
}
```

`json` 配置的 `key`，是用户的**操作类型**，`value` 是一个表达式。当表达式执行结果的值是 `true` 的时候，用户的操作允许执行；否则，用户的操作则不被允许。

操作类型	说明	默认值
read	读取文件，例如：download	false
write	上传/覆盖文件，删除文件	false

规则验证所依据的上下文通过 `auth`、`resource` 对象获得，其提供可验证身份的上下文信息（`auth.uid`）和对象所有权（`resource.openid`）等信息。

```
{
  "read": "auth.uid == resource.openid",
  "write": "auth != null"
}
```

示例

公开

任何不考虑 `auth` 的规则均可被视为 `public` 规则，因为它不考虑用户的身份验证上下文，这些规则在呈现公开数据（静态资源内容）的场景下是很适用。

```
{
  "read": "resource.openid != null"
}
```

对登录的用户开放

在某些情况下，您可能希望限制只有登录用户可以访问用户数据。例如，登录用户才可以查看论坛中的讨论。由于所有未登录用户的 `auth` 变量为 `null`，因此可以设置如下规则：

```
{
```

```
  "read": "auth != null"
}
```

用户私有

`auth` 最常见的使用场景在为个人用户资源提供精细的访问控制，例如上传私人照片等。云存储文件中包含了文件所有者信息（用户唯一 ID），在规则中可以如此限制：

```
{
  "read": "auth.uid == resource.openid",
  "write": "auth.uid == resource.openid"
}
```

实际案例

在一个相册应用中，希望所有登录用户都可以上传，浏览广场图片，不允许未登录用户访问，但可以使用匿名登录访问，匿名身份下只可以浏览，不可上传，则可以对存储设置如下规则。

```
{
  "read": "auth != null",
  "write": "auth.loginType != 'ANONYMOUS' && auth.openid == resource.openid"
}
```

参考

更多详细信息请参见 安全规则 。

云调用拓展

最近更新时间：2024-11-06 16:54:42

云存储可以与云开发提供的云调用拓展服务产生联动，提供**存储+处理**一体化解决方案，您可以结合实际需要安装云开发相应的云调用能力，如图像处理、图像标签、图像盲水印、图像安全审核等。

> **说明：**
> 您可以通过以下两种方式处理图片：为图像 URL 拼接参数进行单次处理、客户端和云函数端持久化图像处理。

图像 URL 拼接参数

例如在安装了图像处理拓展能力的情况下，您可以直接拿云存储的 HTTPS 链接（下载地址或临时链接）进行拼接，拼接之后的链接我们既可以在小程序、Web 网页里使用，也可以用于图床，例如原始图片下载地址为：

```
https://786c-xly-xrlur-130xxxxx6.tcb.qcloud.la/hehe.jpg?
sign=b8axxxxxxxxxxxxxxxxxxxxxb4cd7&t=15xxxxx49
```

下载地址后面拼接参数 `&imageMogr2/thumbnail/!20p`，就可以将图片等比例缩小到原来的 20% 。

```
//将图片等比例缩小到原来的20%
https://786c-xly-xrlur-1xxxx6086.tcb.qcloud.la/hehe.jpg?
sign=b8xxxxxxxxxxd4786449b4cd7&t=159xxxx49&imageMogr2/thumbnail/!20p
```

更多案例如下：

```
const download_url = "https://786c-xly-xrlur-130xxxxx6.tcb.qcloud.la/hehe.jpg?
sign=b8ac757xxxxxxxxxx86449b4cd7&t=1xxxxxxx049"
//缩放宽度，高度不变，下面案例为宽度为原图50%，高度不变
download_url&imageMogr2/thumbnail/!50px

//缩放高度，宽度不变，下面案例为高度为原图50%，宽度不变
download_url&imageMogr2/thumbnail/!x50p

//指定目标图片的宽度（单位为px），高度等比压缩，注意下面的是x，不是px，p与x在拼接里代表着不同的意思
download_url&imageMogr2/thumbnail/640x

//指定目标图片的高度（单位为px），宽度等比压缩：
download_url&imageMogr2/thumbnail/x355

//限定缩略图的宽度和高度的最大值分别为 Width 和 Height，进行等比缩放
download_url&imageMogr2/thumbnail/640x355

//限定缩略图的宽度和高度的最小值分别为 Width 和 Height，进行等比缩放
download_url&imageMogr2/thumbnail/640x355r

//忽略原图宽高比例，指定图片宽度为 Width，高度为 Height ，强行缩放图片，可能导致目标图片变形
```

```
download_url&imageMogr2/thumbnail/640x355!

//等比缩放图片，缩放后的图像，总像素数量不超过 Area
download_url&imageMogr2/thumbnail/150000@

//取半径为300，进行内切圆裁剪
download_url&imageMogr2/iradius/300

//取半径为100px，进行圆角裁剪
download_url&imageMogr2/rradius/100

//顺时针旋转90度
download_url&imageMogr2/rotate/90

//将jpg格式的原图片转换为 png 格式
download_url&imageMogr2/format/png

//模糊半径取8，sigma 值取5，进行高斯模糊处理
download_url&imageMogr2/blur/8x5

//获取图片的基础信息，返回的是json格式，我们可以使用https请求来查看图片的format格式,width宽度、
height高度，size大小，photo_rgb主色调
download_url&imageInfo
```

持久化图像处理

安装拓展 SDK 到项目。

```
//Web端、管理端、云函数端等需安装 cloudbase/extension-ci 到项目
npm install --save @cloudbase/extension-ci@latest

//微信小程序需要安装 cloudbase/extension-ci-wxmp 到项目
npm install --save @cloudbase/extension-ci-wxmp@latest
```

- 可能您的开发者工具会报以下错误：`https://786c-xly-xrlur-1300446086.pic.ap-shanghai.myqcloud.com` 不在以下 `request` 合法域名列表中，请参考文档：`https://developers.weixin.qq.com/miniprogram/dev/framework/ability/network.html`，这个要按照参考文档将链接加入到合法域名当中，不然不会生成图片。
- `action` 是操作类型，它的值可以为：ImageProcess 图像处理，DetectType 图片安全审核，WaterMark 图片盲水印、DetectLabel 图像标签等。
- `operations` 是图像处理参数。

Web

```
const extCi = require("@cloudbase/extension-ci");
const tcb = require("tcb-js-sdk");

const readFile = async function(file) {
```

```
  let reader = new FileReader();
  let res = await new Promise(resolve =&gt; {
    reader.onload = function(e) {
      let arrayBuffer = new Uint8Array(e.target.result);
      resolve(arrayBuffer);
    };
    reader.readAsArrayBuffer(file);
  });
  return res;
};
let file = document.getElementById("selectFile").files[0];
let fileContent = await readFile(file);
```

Node.js

```
const tcb = require("@cloudbase/node-sdk");
const app = tcb.init();
const extCi = require("@cloudbase/extension-ci");
tcb.registerExtension(extCi);

async function process() {
  try {
    const opts = {
      rules: [
        {
          fileid: "/image_process/tcbdemo.jpeg",
          rule: "imageMogr2/format/png"
        }
      ]
    };
    const res = await app.invokeExtension("CloudInfinite", {
      action: "ImageProcess",
      cloudPath: "tcbdemo.jpg",
      fileContent,
      operations: opts
    });
    console.log(res);
    return res;
  } catch (err) {
    console.log(err);
  }
}
```

使用控制台管理云存储

最近更新时间：2025-02-14 11:41:12

您可以使用 腾讯云 CloudBase 控制台 对云存储进行管理。

文件列表

1. 登录 云开发平台，进入 云存储 页面。
2. 进入文件管理页面，您可以查看云存储空间中所有的文件。
3. 单击文件名或**详情**，即可查看关于此文件的所有信息，例如文件名称、文件大小、存储位置、更新时间等。

权限设置

如需设置文件权限，选择**权限设置**，根据实际需求，勾选存储权限并保存；您还可以自定义安全规则。

缓存配置

什么是缓存配置

- 云存储内的文件默认 CDN 加速，开发者可以通过改变缓存配置来控制 CDN 遵循的过期规则。
- 合理的配置缓存时间，能够有效的提升命中率，降低回源率，节省您的带宽。
- CDN 节点上缓存的用户资源也面临**过期**问题。
  - 若资源处于未过期状态，用户请求到达节点后，节点会将此资源直接返回给用户，提升获取速度。
  - 若资源处于过期状态（即超过了设置的有效时间），用户请求会由节点发送至源站，重新获取内容并缓存至节点，同时返回给用户。

如何进行缓存配置

1. 进入**缓存配置** tab 页，您即可查看**缓存配置**模块。

2. 单击**编辑缓存配置**，您可以根据自身业务需求，在默认配置上添加缓存时间配置。 默认配置是对所有文件缓存 2 分钟。
3. 支持三种配置方式：
   - 按**文件类型**设置缓存过期时间。
     - 配置缓存时间时可填入多项，每项用 `;` 隔开，内容区分大小写，必须是以 `.` 开头的文件后缀，例如 `.png` 。
     - 刷新时间设置为 0 时，不缓存，所有请求转发至用户源站，缓存时间设置最大值不能超过 365 天。
     - 输入框默认提示：例如 `.png` ， `.jpg` ， `.php` 。
   - 按**文件夹**设置缓存过期时间。
     - 配置缓存时间时可填入多项，每项用 `;` 隔开，内容区分大小写，必须是以 `/` 开头的文件夹。
     - 刷新时间设置为 0 时，不缓存，所有请求转发至用户源站，缓存时间设置最大值不能超过 365 天。
   - 按**文件**设置缓存过期时间。
     配置缓存时间时可填入多项，每项用 `;` 隔开，内容区分大小写，支持匹配某一类型文件，例如 `/test/abc/\*.jpg` 。
4. 单击**保存**，将进行配置部署，您需等待 5 分钟左右。

> **说明：**
> 配置部署中，若您再次编辑缓存配置，会覆盖之前的配置。以最后一次部署结果为准。

策略优先级

配置项列表**底部**优先级高于**顶部**优先级。您可以拖动列表前面的移动图标即可调整优先级。

> **说明：**
> 当设置了多条缓存策略时，相互之间会有重复。

假设某域名已配置如下缓存配置：

目标	缓存时长
所有文件	2 分钟
.php .jsp .aspx	0 秒
.jpg .png .gif	300 秒
/test/*.jpg	400 秒
/test/abc.jpg	200 秒

假设域名为 `www.test.com` ，资源为 `www.test.com/test/abc.jpg` ，其匹配方式如下：

- 匹配第一条所有文件，命中，此时缓存时间为2分钟。
- 匹配第二条，未命中。
- 匹配第三条，命中，此时缓存时间为300秒。
- 匹配第四条，命中，此时缓存时间为400秒。
- 匹配第五条，命中，此时缓存时间为200秒。

因此最终缓存时间为200秒，以最后一次匹配生效。

使用 CloudBase CLI 管理云存储

最近更新时间：2023-09-13 16:55:04

使用 CloudBase CLI 命令行界面交互工具，您可以非常方便的管理项目的云存储资源，例如批量上传/下载文件/文件夹，获取文件访问链接与信息，设置/获取云存储的权限等。

下载文件/文件夹

您可以使用下面的命令来下载文件：

```
cloudbase storage download cloudPath localPath
```

下载文件夹时，需要指定 `--dir` 参数。
例如以下命令，会将云存储根目录下的 **cloudbase** 目录中的所有文件，下载到本地 **download** 目录中：

```
cloudbase storage download cloudbase ./download --dir
```

上传文件/文件夹

您可以使用下面的命令将本地电脑的文件或文件夹上传到云存储，当 CLI 检测到 **localPath** 为文件夹时，会自动上传文件内的所有文件，如果重复上传会覆盖。

```
cloudbase storage upload localPath cloudPath
```

当不传入 cloudPath，文件会上传到云端的根目录下，同时文件夹的层次结构会被保留，例如下面的命令会把项目根目录的 download 文件夹里的内容直接上传到云存储的根目录里，download 的子文件夹会成为云存储的二级目录。

```
cloudbase storage upload ./download
```

删除文件/文件夹

如果您想删除云存储里的文件或文件夹，可以使用下面的命令，删除文件夹时，需要指定 `--dir` 参数。

```
# 删除文件
cloudbase storage delete cloudPath

# 删除文件夹
cloudbase storage delete cloudPath --dir
```

获取文件/文件夹信息

在不打开云开发控制台或网页控制台，您也可以通过 Cloudbase Cli 工具了解云存储里的文件夹或文件的信息，例如在终端里输入以下命令可以列出云存储里的文件夹里的所有文件信息，例如大小、修改时间、key、Etag 等信息。

```
cloudbase storage list cloudPath
```

列出文件夹内文件列表

例如您可以直接使用以下命令打印云存储根目录里的所有文件，其中二级目录里的文件会用路径的方式显示。

```
cloudbase storage list cloudPath
```

参考

更多详细信息请参见 CloudBase CLI 命令行工具 。

管理端 SDK

最近更新时间：2023-09-13 16:55:04

您可以将管理端 manager-node SDK 部署在本地、云端服务器甚至云函数里对云存储空间内的文件进行批量上传、下载、删除、获取文件信息、获取临时链接以及设置文件权限等。

上传文件

在管理端调用 uploadFile 方法进行单个文件的上传。

```
const CloudBase = require("@cloudbase/manager-node");

const path = require("path");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function uploadFile() {
  await storage.uploadFile({
    localPath: path.resolve("./data.txt"),
    cloudPath: "files/data.txt",
    onProgress: (data) => {}
  });
}

uploadFile();
```

上传文件夹

在管理端调用 uploadDirectory 可以将文件夹内的文件批量上传到云存储空间。

```
const CloudBase = require("@cloudbase/manager-node");

const path = require("path");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function uploadDirectory() {
  await storage.uploadDirectory({
    localPath: path.resolve("./files"),
    cloudPath: "",
```

```
    onProgress: (data) => {}
  });
}

uploadDirectory();
```

下载文件

在管理端调用 downloadFile 方法进行单个文件的下载。

```
const CloudBase = require("@cloudbase/manager-node");

const path = require("path");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function downloadFile() {
  await storage.downloadFile({
    cloudPath: "files/data.txt",
    localPath: path.resolve("./data.txt")
  });
}

downloadFile();
```

下载文件夹

在管理端调用 downloadDirectory 方法可以将云存储空间内指定文件夹下载到管理端指定文件夹。

```
const CloudBase = require("@cloudbase/manager-node");

const path = require("path");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function downloadDirectory() {
  await storage.downloadDirectory({
    cloudPath: "files/music",
    localPath: path.resolve("./music")
  });
}
```

```
downloadDirectory();
```

删除文件

在管理端调用 deleteFile 方法进行单个或多个文件的批量删除。

```
const CloudBase = require("@cloudbase/manager-node");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function deleteFile() {
  await storage.deleteFile(["files/data.txt", "uploads/logo.png"]);
}

deleteFile();
```

删除文件夹

在管理端调用 deleteDirectory 方法可以删除云存储空间内整个文件夹。

```
const CloudBase = require("@cloudbase/manager-node");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function deleteDirectory() {
  await storage.deleteDirectory("files/");
}

deleteDirectory();
```

获取文件信息

在管理端调用 getFileInfo 方法可以获取文件的大小、类型、修改时间、对象的实体标签 ETag 。

```
const CloudBase = require("@cloudbase/manager-node");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
```

```
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function getFileInfo() {
  const info = await storage.getFileInfo("files/data.txt");
  console.log(info);
}

getFileInfo();
```

列出文件夹内文件列表

在管理端调用 listDirectoryFiles 列出云存储空间内指定文件夹下的所有文件以及文件信息。

```
const CloudBase = require("@cloudbase/manager-node");

const { storage } = new CloudBase({
  secretId: "Your SecretId",
  secretKey: "Your SecretKey",
  envId: "Your envId" // 云开发环境ID，可在腾讯云云开发控制台获取
});

async function listDirectoryFiles() {
  const res1 = await storage.listDirectoryFiles("dir/data");

  const res2 = await storage.listDirectoryFiles("dir/data", 20);

  const res3 = await storage.listDirectoryFiles("dir/data", 20, "dir/dat");

  for (let item in res1) {
    console.log(item);
  }
}
listDirectoryFiles();
```

参考

管理端 SDK 还可以获取文件临时链接、获取/设置文件存储权限，更多详细信息请参见 管理端 manager-node sdk 。

云函数

概述

最近更新时间：2023-09-13 16:55:04

使用 CloudBase 的云函数，您可以以函数的形式运行后端代码，响应 SDK 的调用或者 HTTP 请求。您的代码会储存在云端，并且在托管环境中运行，无需管理或运维自己的服务器。

主要特性

多端访问

您编写的云函数可以响应各种 CloudBase SDK 的调用，也可以使用 以 HTTP 请求的形式调用云函数 。

> **说明**
> 云函数和 CloudBase 服务天然集成，内部**无需密钥**即可直接使用 CloudBase 服务端 SDK 调用您的云开发资源。

零运维和弹性伸缩

只需要使用 CloudBase CLI 工具 或者在 腾讯云 CloudBase 控制台 上部署云函数，CloudBase 会根据您云函数的使用情况，自动伸缩计算资源，无需人工管理。

快速开始

最近更新时间：2025-04-08 15:45:51

步骤1：创建目录和云函数文件

1. 在本地创建一个空的文件夹，作为项目的根目录。
2. 进入项目根目录，创建 functions 文件夹。
3. 在 functions 下创建 hello_world 文件夹，包含 index.js 与 package.json 两个文件。

此时目录结构如下：

```
└── functions
    └── hello_world
        ├── index.js
        └── package.json
```

index.js

```
exports.main = async function () {
  return "Hello World!";
};
```

package.json

```
{
  "name": "hello_world",
  "version": "1.0.0",
  "main": "index.js"
}
```

步骤2：发布云函数

命令行工具

1. 安装并登录 CLI 工具 。
2. 在项目根目录运行以下命令，并且使用默认配置：

```
cloudbase fn deploy hello_world -e <env-id>
```

小程序开发者工具

您需要先下载并安装 小程序开发者工具 。
如果您是首次在微信小程序中使用云开发，请参见 第一个云开发小程序 。
在微信小程序中首次使用云函数请参见操作指引 我的第一个云函数 。

步骤3：调用云函数

调用云函数有两种方法：

- 使用云开发 SDK。
- 使用 HTTP 访问服务 。

使用 SDK 调用云函数

Web

```
//初始化SDK实例
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "xxxx-yyy"
});
app
  .callFunction({
    // 云函数名称
    name: "hello_world",
    // 传给云函数的参数
    data: {
      a: 1
    }
  })
  .then((res) => {
    console.log(res);
  })
  .catch(console.error);
```

小程序

```
wx.cloud
  .callFunction({
    // 云函数名称
    name: "hello_world",
    // 传给云函数的参数
    data: {
      a: 1,
      b: 2
    }
  })
  .then((res) => {
```

```
    console.log(res.result); // 3
  })
  .catch(console.error);
```

Node.js

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({
  env: "xxxx-yyy"
});
app
  .callFunction({
    // 云函数名称
    name: "hello_world",
    // 传给云函数的参数
    data: {
      a: 1
    }
  })
  .then((res) => {
    console.log(res);
  })
  .catch(console.error);
```

使用 HTTP 调用云函数

执行以下命令创建一条 HTTP 服务路由，路径为 `/hello` ，指向的云函数为 `hello_world` ：

```
cloudbase service create -p hello -f hello_world -e <env-id>
```

随后便可以通过 `https://<env-id>.service.tcloudbase.com/hello` 调用云函数，并获得返回结果。

步骤4（可选）：在云函数内使用 Node.js SDK

首先，我们在 `functions/hello_world/` 路径下，执行以下命令，安装 Node.js SDK：

```
npm install --save @cloudbase/node-sdk
```

将 `functions/hello_world/index.js` 内容改为：

```
const cloudbase = require("@cloudbase/node-sdk");
exports.main = async function () {
  if (cloudbase) {
    return "Happy Hack With Cloudbase!";
  }
```

```
};
```

然后我们重新部署云函数：

```
cloudbase fn deploy hello_world -e <env-id>
```

在云函数的调用结果里，我们便可以看到结果：

```
app
  .callFunction({
    name: "hello_world"
  })
  .then((res) => {
    console.log(res.result); // 'Happy Hack With Cloudbase!'
  });
```

> **注意：**
> 在线调试暂不支持 Python，您可以在本地调试完成后再上传代码。

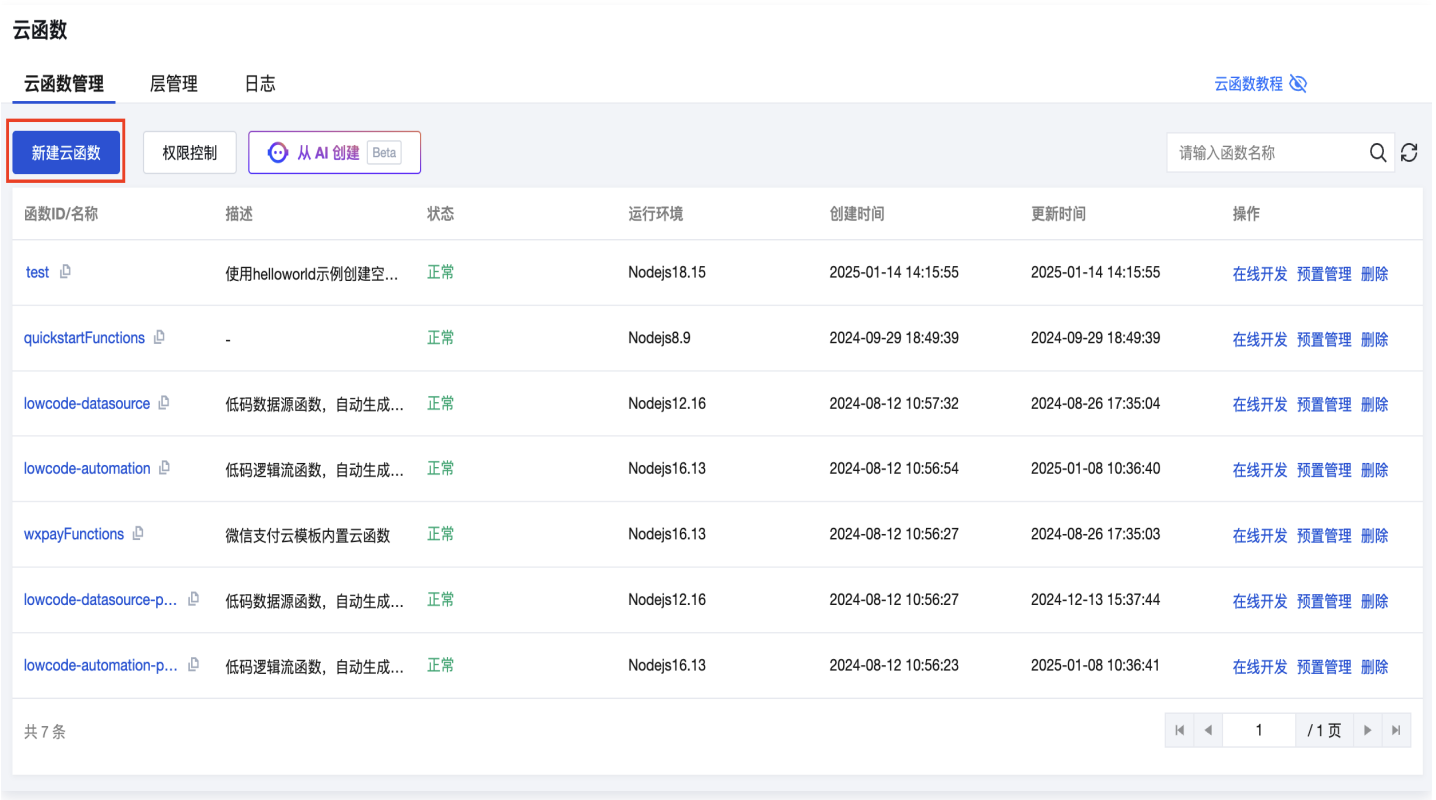
管理云函数

最近更新时间：2025-01-16 10:45:32

您可以使用 CloudBase CLI 工具 或者 云开发平台 来完成对云函数的管理和配置。

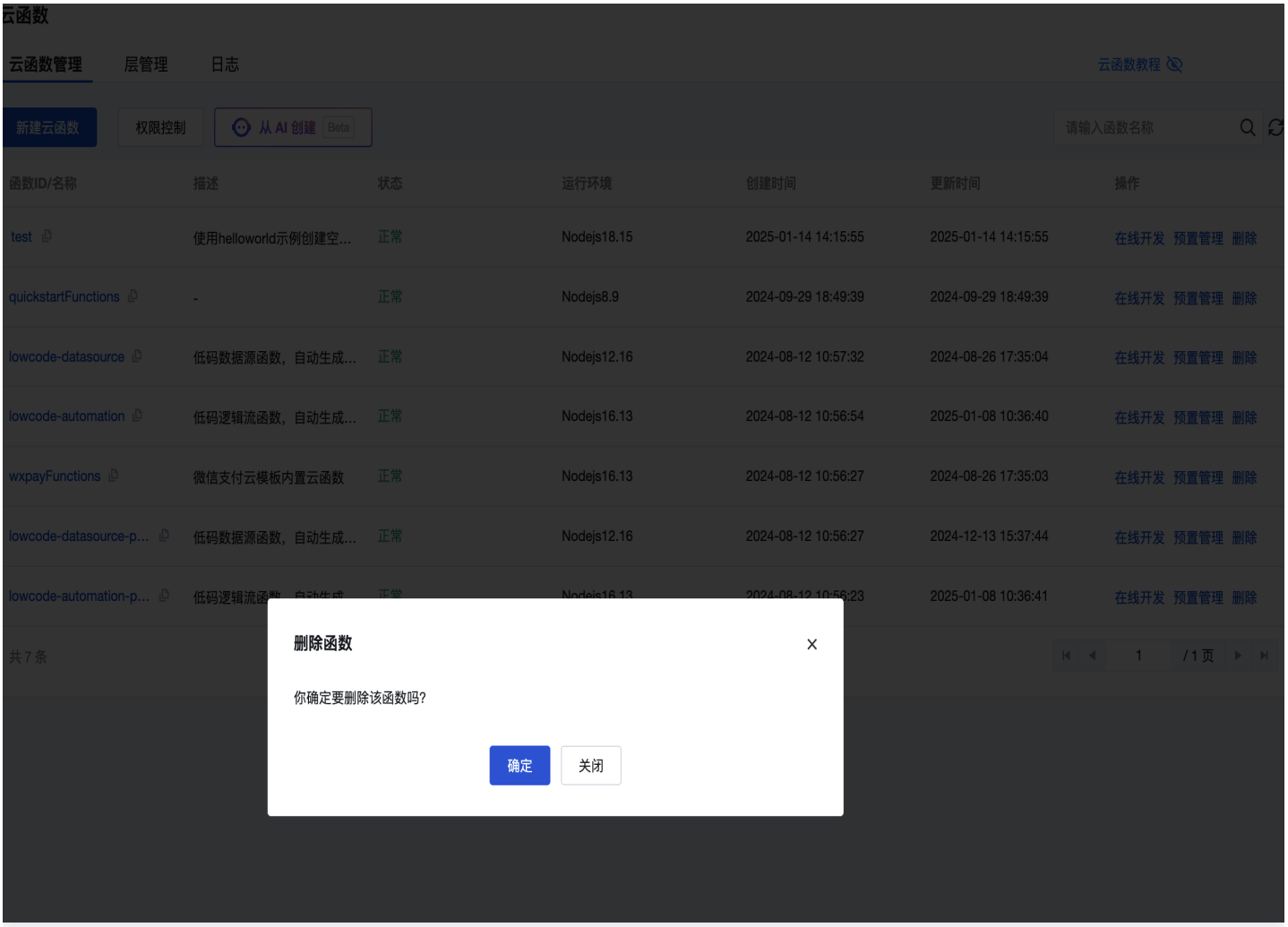
新建云函数

登录 云开发平台，进入 云函数 页面，单击**新建云函数**，填写函数名并单击**确定**即创建并部署成功。



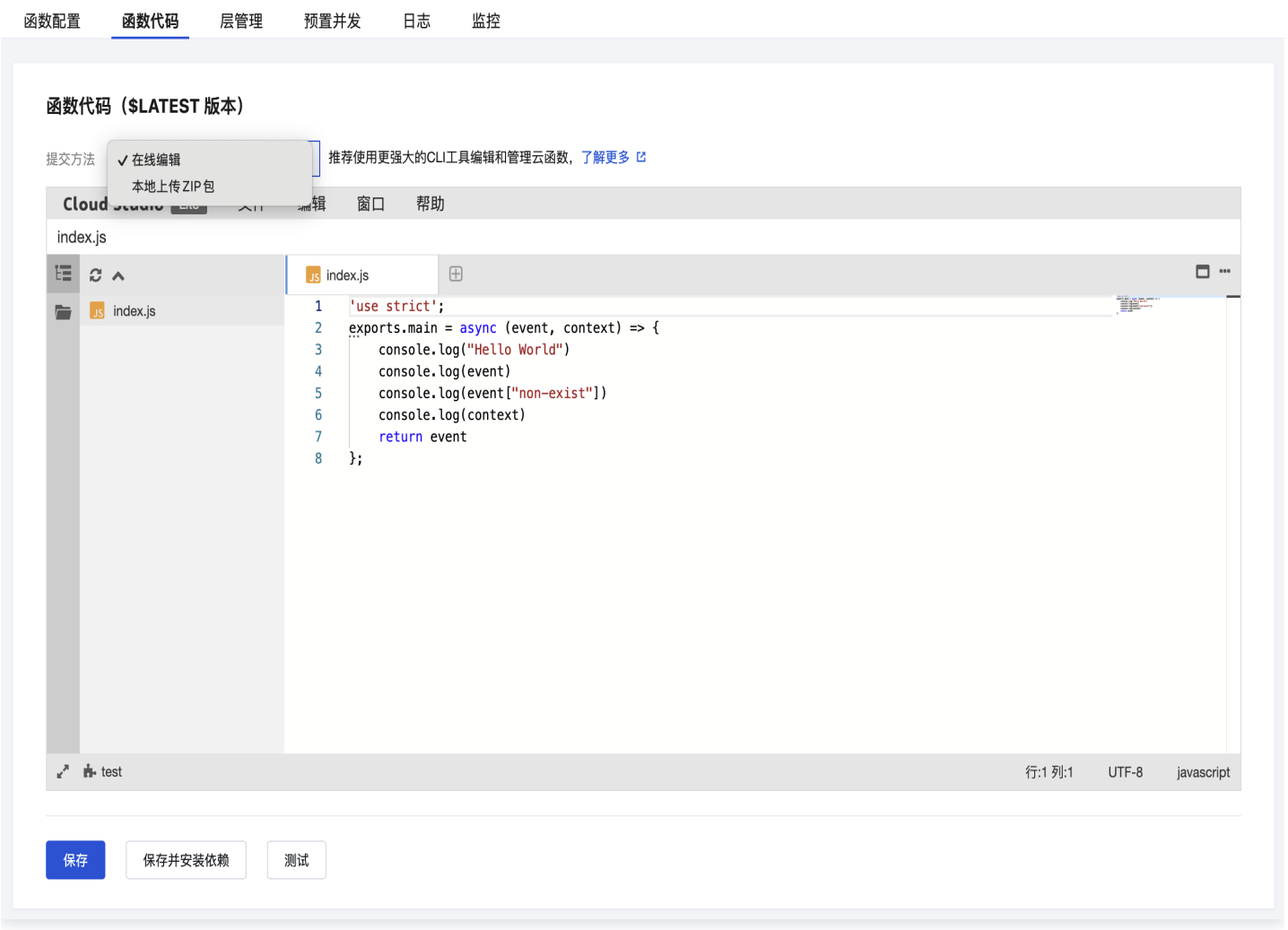
删除云函数

点击删除按钮，可以删除云函数。**删除后云函数不可恢复，且立即不可访问，请谨慎操作。**



更新并部署函数代码

您可以直接编辑函数的代码，或者将相关代码压缩成 zip 包，然后在控制台上传并部署。



配置云函数

在函数配置页面，选择要操作的云函数，可以更改云函数的各项配置。

内存配置

云函数运行时的最大内存限制，默认为 256M，最大 2048M。

超时时间

云函数的最大运行时间，默认时间为 20s，最大值为 900s。函数超过该时间仍未运行结束时，将被强制中断。

公网访问

云函数默认放通公网访问，用户可以自行控制是否关闭。

内网访问

您可以通过配置网络，使云函数可以访问指定 VPC 私有网络。

- 在配置为**未配置 VPC** 的情况下，云函数实例启动在默认的独立网络环境中，并具备外网访问能力。
- 在配置为指定 VPC 的情况下，云函数实例启动在指定 VPC 网络中，具备访问 VPC 内资源的能力。

> **说明**
> - 配置 VPC 后如期望获得外网访问能力，请确认 VPC 网络中配置了 公网网关、NAT 网关 等可访问外网的方式。

- 有关 VPC 网络配置的更多说明，请参见 VPC 私有网络说明 。
- 有关云函数网络配置的更多说明，请参见 网络配置详情 。

固定出口 IP

开启后，将会获得一个固定的公网出口 IP，并且该 IP 会和该环境下其他开启同样功能的函数共享。

说明

- 只有云函数开启公网访问的时候，该能力才能生效。
- 云开发环境和云函数命名空间是一一对应关系。

环境变量

您可以使用键/值对的形式定义可从函数代码访问的环境变量。

使用 CLI 工具管理云函数

使用 CloudBase CLI 工具可以方便地完成云函数的所有管理操作，例如：

```
  functions:list [options]                                          展示云函数列表
  functions:download [options] <functionName> [dest]                下载云函数代码
  functions:deploy [options] [functionName]                         部署云函数
  functions:delete [options] [functionName]                         删除云函数
  functions:detail [options] [functionName]                         获取云函数信息
  functions:code:update [options] <functionName>                    更新云函数代码
  functions:config:update [options] [functionName]                  更新云函数配置
  functions:copy [options] <functionName> <newFunctionName>         拷贝云函数
  functions:log [options] <functionName>                            打印云函数日志
  functions:trigger:create [options] [functionName]                 创建云函数触发器
  functions:trigger:delete [options] [functionName] [triggerName]   删除云函数触发器
  functions:invoke [options] [functionName]                         触发云端部署的云函数
  functions:run [options]                                           本地运行云函数（当前仅
支持 Node）
```

更多信息请参见 CloudBase CLI 工具 。

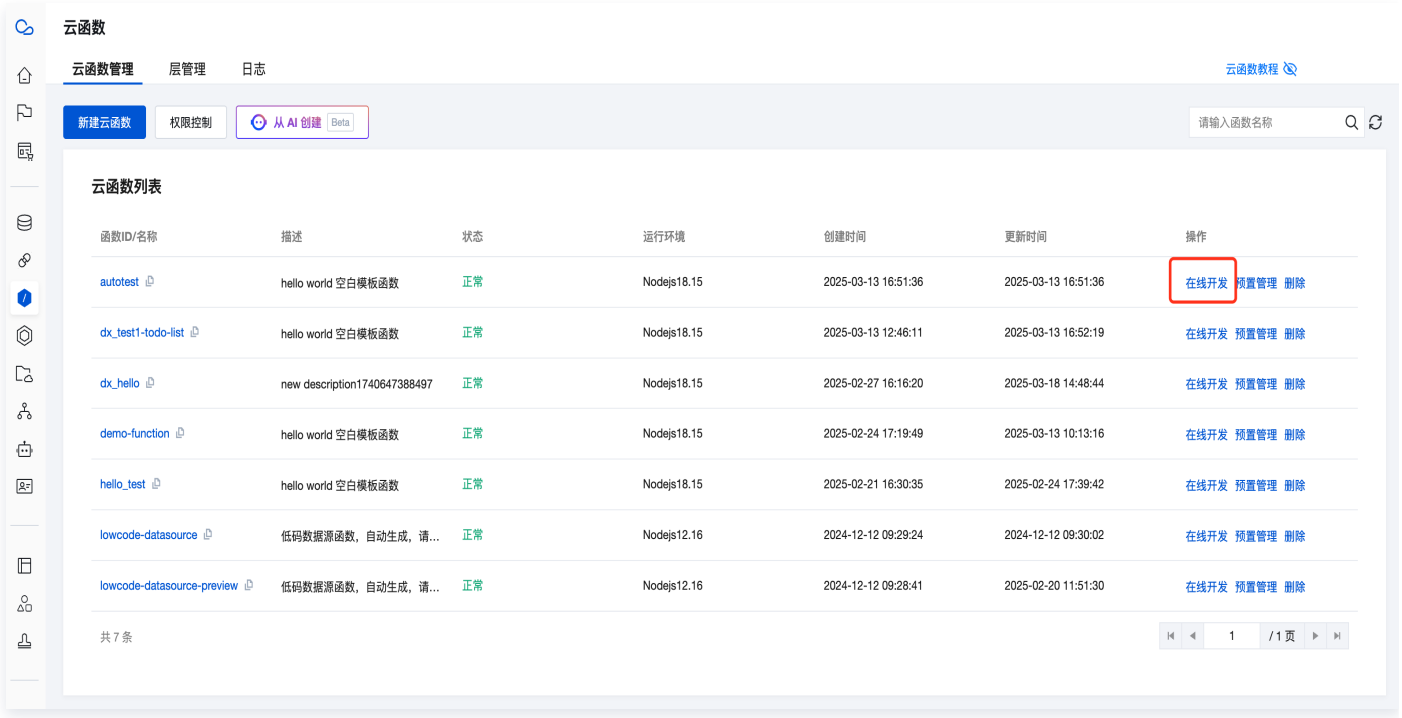
在线开发

最近更新时间：2025-04-30 11:42:52

云函数提供了**在线开发**能力，让您可以通过浏览器中的在线编辑器直接进行云函数的开发、调试、部署等操作。

操作步骤

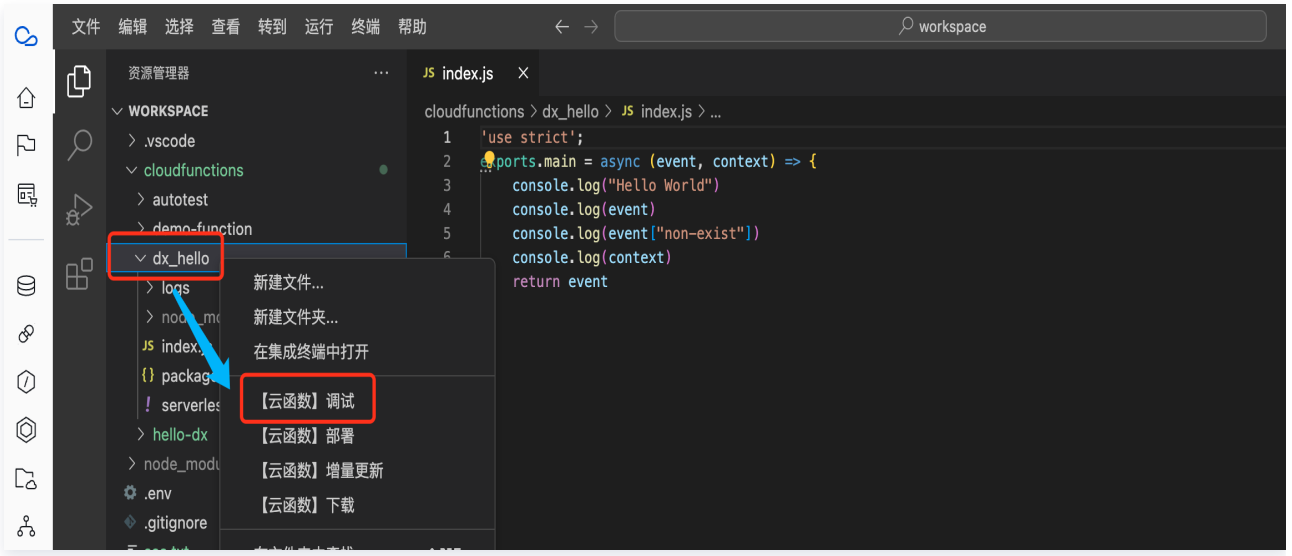
1. 单击云函数右侧的**在线开发**，即可进入在线编辑器。



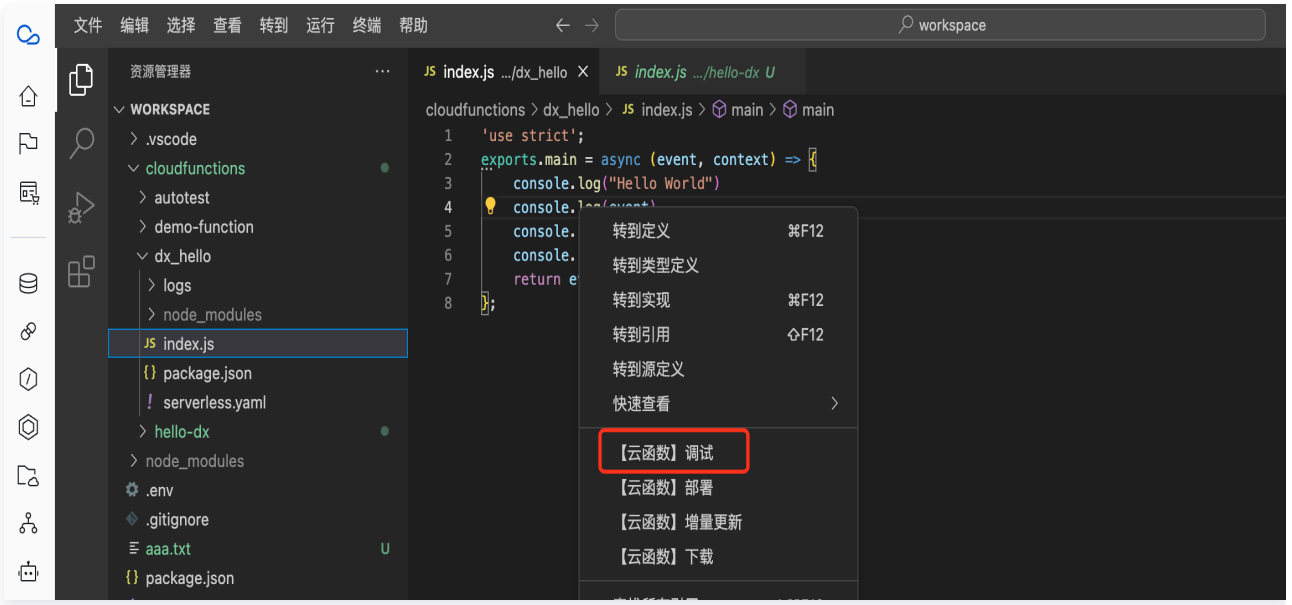
> **注意：**
> 如果进入编辑器的时间过长，您可以尝试返回再重新进入。

2. 进入在线编辑器后，您可对云函数进行调试、部署、增量更新及下载等操作。
   - 您可以在以下位置进行操作，下述以**调试**截图为例。
     - 资源管理器。在指定云函数的目录上单击右键菜单。

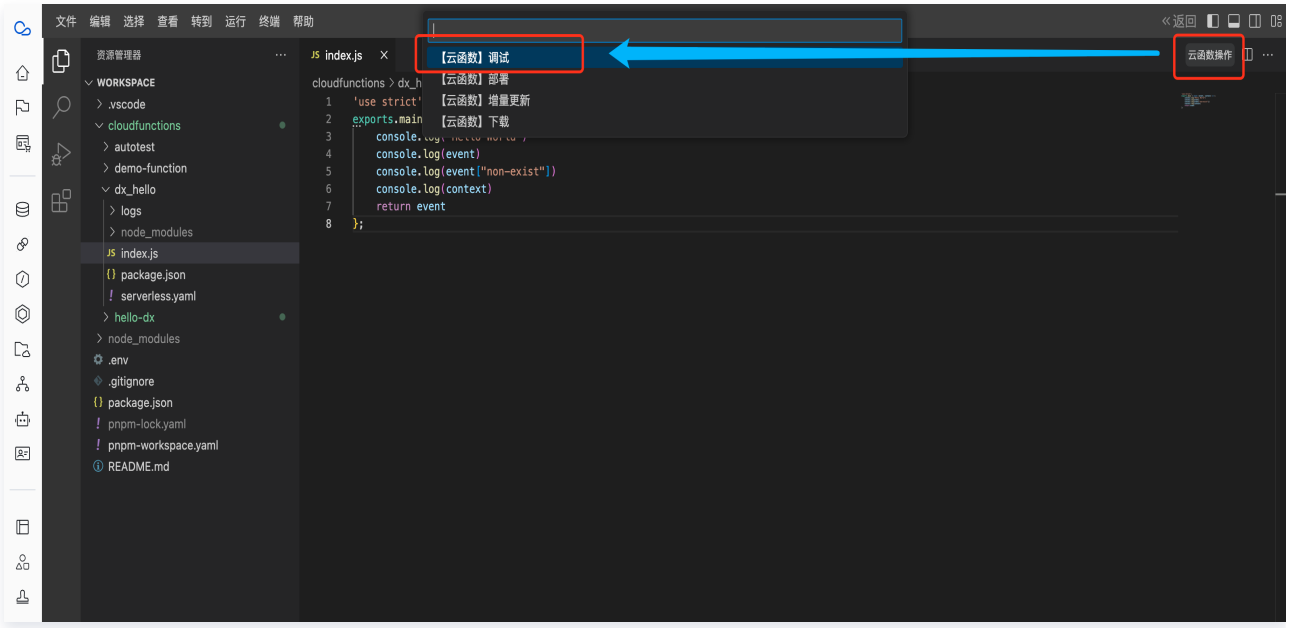
> **说明：**
> 云函数通常位于 cloudfunctions 目录下。



- 编辑区。在打开的云函数代码编辑区上单击右键菜单，选择【**云函数**】**调试**。

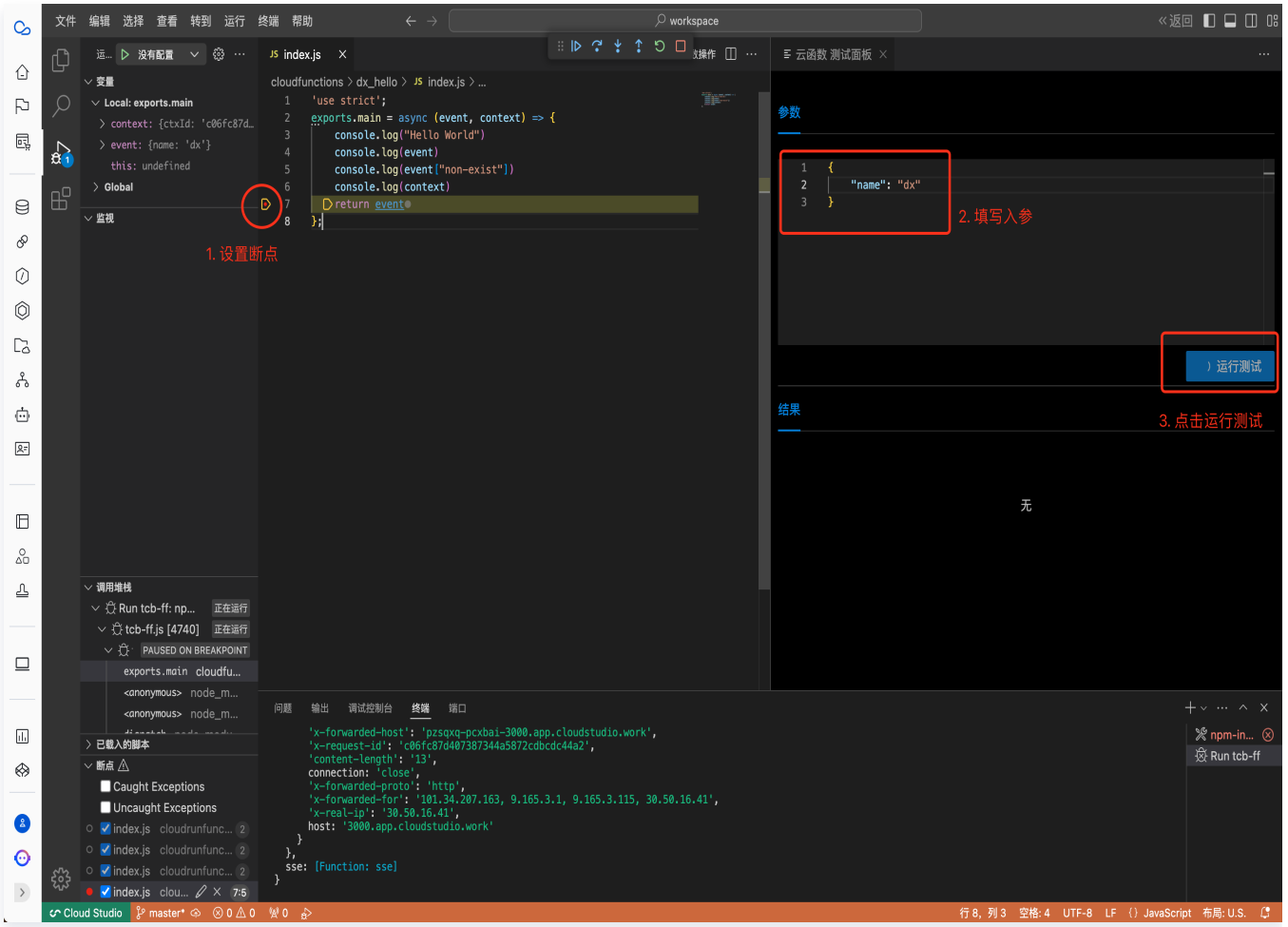


- 编辑区右上角。在打开的云函数代码编辑区右上角，单击**云函数操作**，选择【**云函数**】**调试**。



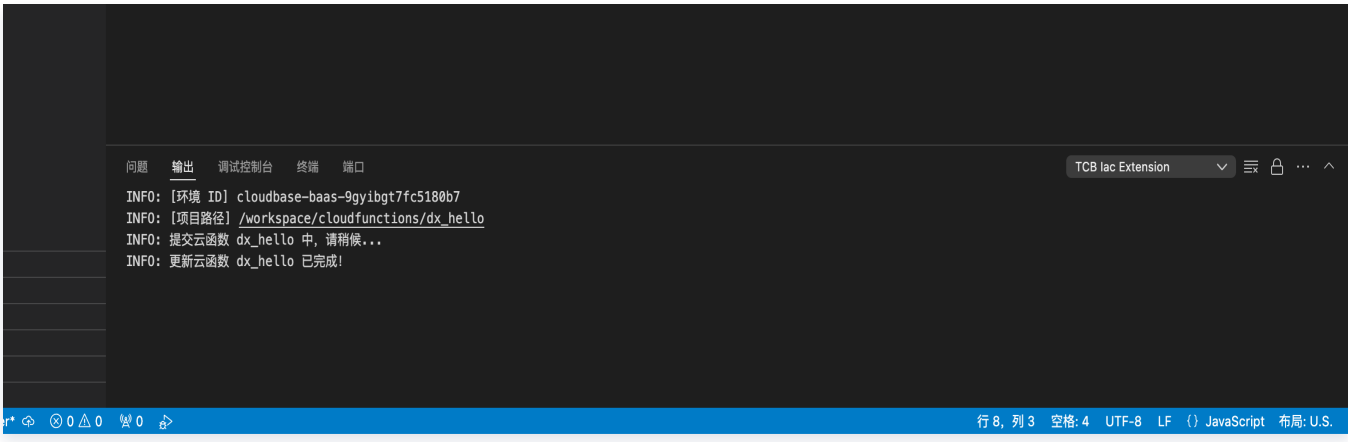
- **调试**

在在线编辑器中使用调试功能时，系统会自动启动调试服务并打开测试面板，随后您可以按下图所示进行调试操作。



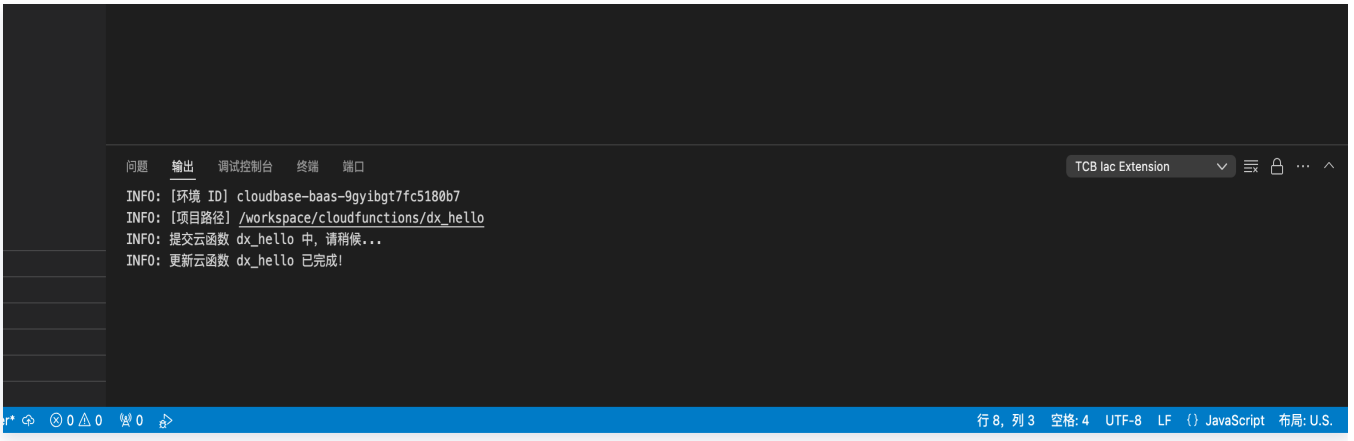
- **部署**

  在使用部署功能时，系统会进行完整的部署，即同时更新云函数的配置和代码。配置文件通常位于云函数目录下的 serverless.yaml 文件中。
  部署成功后，系统会显示以下提示：



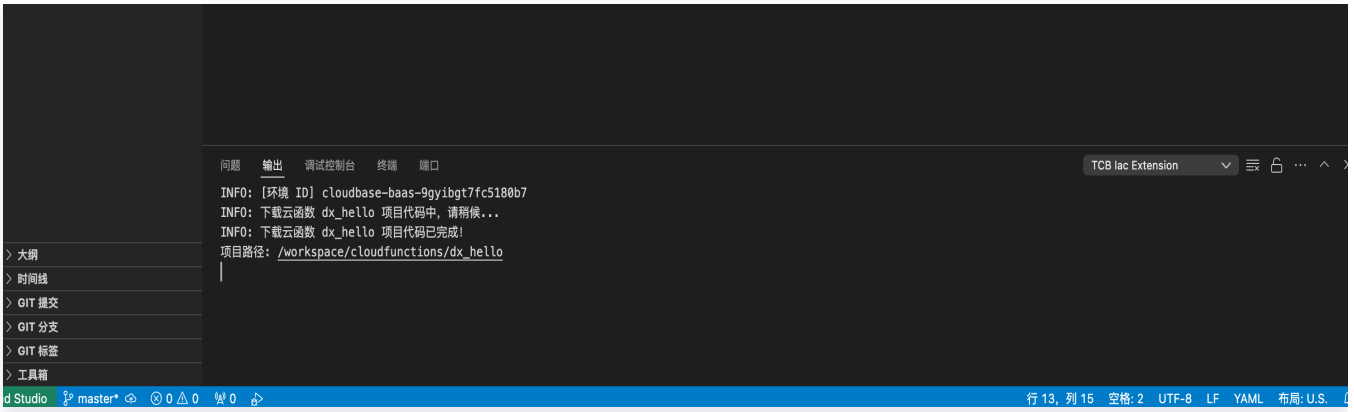
- **增量更新**

  在使用增量更新功能时，系统只会更新云函数的代码，速度较快。增量更新成功后，系统会显示以下提示：



- **下载**

  下载功能能够将云函数的代码及配置文件同步到本地。下载成功后，系统会显示以下提示：


问题
输出
调试控制台
终端
端口
TCB Iac Extension
INFO: [环境 ID] cloudbase-baas-9gyibgt7fc5180b7
INFO: 下载云函数 dx_hello 项目代码中，请稍候...
INFO: 下载云函数 dx_hello 项目代码已完成!
项目路径: /workspace/cloudfunctions/dx_hello
大纲
时间线
GIT 提交
GIT 分支
GIT 标签
工具箱
d Studio
master*
行 13，列 15
空格: 2
UTF-8
LF
YAML
布局: U.S.

层管理

最近更新时间：2024-10-14 16:40:11

如果您的云函数拥有较多的依赖库或公共代码文件，您可以使用云函数中的层进行管理。

> **说明**
> 使用层管理，您可以将依赖放在层中而不是部署包中，可确保部署包保持较小的体积。对于 Node.js、Python 和 PHP 函数，只需将部署程序包保持在 10MB 以下，就可以在控制台中在线编辑函数代码。

工作方式

创建与绑定

创建

创建层的压缩文件将按照层的版本进行存储。

绑定

层在与函数进行绑定时，将按照具体的层版本与函数版本进行绑定。

> **说明**
> 一个函数目前最多支持绑定 5 个层的具体版本，并在绑定时有一定顺序。

运行时加载与访问

已绑定层的函数被触发运行，启动并发实例时，将会解压加载函数的运行代码至 `/var/user/` 目录下，同时会将层内容解压加载至 `/opt` 目录下。

- 若需使用或访问的文件 file，放置在创建层时压缩文件的根目录下。则在解压加载后，可直接通过目录 `/opt/file` 访问到该文件。
- 若在创建层时，通过文件夹进行压缩 `dir/file` ，则在函数运行时需通过 `/opt/dir/file` 访问具体文件。

> **说明**
> 在函数绑定了多个层的情况下，层中文件的解压加载将按照绑定时的顺序进行。将按序号从小到大的顺序进行排序，排序越靠后则层加载时间也相应靠后，但均会在函数的并发实例启动前完成加载。在函数代码初始化时，就已经可使用层中的文件了。

推荐使用方式

层中通常用来存储不经常变更的静态文件或代码依赖库。在存储代码依赖库时，可以直接将可用的依赖库打包并上传至层中。

示例

- 在 Python 环境中，可以将依赖库的代码包文件夹直接打包并创建为层，则在函数代码中可直接通过 import 引用。
- 在 Nodejs 环境中，可以将项目的 node_modules 依赖库文件夹打包并创建为层，则在函数代码中可直接通过 require 引用。

通过使用层，可以将函数代码和依赖库或依赖的静态文件分离，保持函数代码较小体积。在使用命令行工具、IDE 插件或控制台编辑函数时，均可以快速上传更新。

说明事项

- 层中的文件将会添加到 `/opt` 目录中，此目录在函数执行期间可访问。
- 如果您的函数已绑定了多个层，这些层将按顺序合并到 `/opt` 目录中。如果同一个文件出现在多个层中，云函数将会保留最大序号层里的文件。

层管理相关操作

步骤1：创建层

1. 登录云开发控制台，进入到环境中，单击左侧云函数菜单，再单击 层管理，进入层列表页面。

2. 在新建层页面，根据实际需求设置层信息。如下图所示：

   - **层名称**：输入自定义层名称。
   - **描述**：层的描述信息，根据实际情况填写。
   - **层代码**：支持**本地上传 zip 包**，最大支持 50 M。确定提交方法后单击**上传**，在弹出的依赖包选择界面，选择需上传的依赖包并单击**确定**。
   - **添加运行环境**：该层的兼容运行环境，最多可设置 5 个。
3. 单击**确定**即可成功创建。

步骤2：云函数绑定层

1. 选择需进行层管理的函数，进入函数管理页面。
2. 选择**层管理**页签，并单击**绑定**。如下图所示：

3. 在弹出的**绑定层**窗口中，选择对应**层名称**及**层版本**。如下图所示：

4. 单击**确定**即可完成绑定。

多语言支持

最近更新时间：2024-06-11 14:27:01

CloudBase 云函数目前支持多种语言：

- Nodejs 18.15
- Nodejs 16.13
- Nodejs 14.18
- Nodejs 12.16
- Nodejs 10.15
- Nodejs 8.9（即将下线）
- Php 8.0
- Php 7.4
- Php 7.2
- Python 3.9
- Python 3.7
- Python 3.6
- Python 2.7
- Golang 1
- Java 8

跨语言调用

您可以在任何 SDK 中调用某个云函数，并且无需了解这个云函数是由什么语言编写的。例如，您可以在 Web 端内使用 Web SDK 调用 Python 编写的云函数。

Python 云函数

请参见 腾讯云 SCF 文档 - Python 。

Golang 云函数

请参见 腾讯云 SCF 文档 - Golang 。

PHP 云函数

请参见 腾讯云 SCF 文档 - PHP 。

运行机制

最近更新时间：2023-09-13 16:55:05

云函数的特点

无状态

CloudBase 会根据当前负载情况，自动控制云函数实例的数量，并且均衡地分发请求。连续的多次请求**有可能由同一实例处理，也可能不是。**
所以，**开发者在编写云函数时，应注意保证云函数是无状态的、幂等的**，即当次云函数的执行不依赖上一次云函数执行过程中在运行环境中残留的信息。

事件模型与触发器

云函数采用事件触发模型，每一次调用本质上是触发了一次云函数调用事件。
云函数的调用方，一般被称为**触发器**，目前 CloudBase 云函数支持以下触发器：

- SDK 调用触发。
- HTTP 触发（请参见 使用 HTTP 访问云函数 ）。
- 定时器触发（请参见 定时触发器 ）。

自动扩缩容

开发者无需关心云函数扩容和缩容的问题，平台会根据负载自动进行扩缩容。

云函数的入参

每个云函数的传入参数有两个对象， `event` 对象和 `context` 对象。

event 对象

`event` 对象指的是**触发云函数的事件**。例如：

- 在小程序端调用时， `event` 是小程序端调用云函数时传入的参数。
- 在使用 HTTP 请求的形式调用时， `event` 是 集成请求体 。

context 对象

`context` 对象包含了此调用的**调用信息和函数的运行状态**，可以使用 `context` 了解服务运行的情况。

简单示例

本段代码的含义指将传入的 a 和 b 相加并作为 sum 字段返回给调用端。
例如，我们定义一个云函数，命名为 add ，功能是将传入的两个参数 a 和 b 相加。示例代码如下：

```
// index.js 是入口文件，云函数被调用时会执行该文件导出的 main 方法
// event 包含了调用端调用该函数时传过来的参数，同时还包含了用户登录态 `openId` 和应用 `appId` 信息
exports.main = async (event, context) => {
  let { a, b } = event;
  let sum = a + b;
```

```
  return {
    sum
  };
};
```

运行环境

- 云函数运行在云端**容器化的 Unix 环境**中。
- 云函数在处理并发请求的时候会创建多个云函数实例。
- 每个云函数实例之间相互隔离，**没有公用的内存或硬盘空间**。
- 如果有临时读写文件的需求，**可以使用 `/tmp` 目录**，这是一块 512MB 的临时磁盘空间，用于处理单次云函数执行过程中的临时文件读写需求。
- 云函数实例的创建、管理、销毁等操作由平台自动完成。

> **说明**
> - 目前云函数的运行环境 CentOS 7.2，开发者编写代码时，**不应依赖特定的操作系统或操作系统的版本号**，因为运行环境可能随时会发生变化。
> - 函数的临时磁盘空间在函数执行完毕后可能被销毁，不应依赖和假设在磁盘空间存储的临时文件会一直存在。**如果需要持久化的存储，请使用 云存储 功能**。

运行时

当前支持以下运行时：

- Node.js 8.9
- Node.js 10.15
- PHP 7.2
- Python 2.7
- Python 3.6
- Golang 1.8

内存限制

内存默认限制为 256 MB，您也可以调整限制为 128 MB ~ 2048 MB。

函数并发量

函数的并发数量是指在任意指定时间对函数代码的执行数量。对于函数来说，每个请求就会执行一次。因此，请求量会影响函数的并发数。

> **说明**
> 当前默认情况下，单个云函数的最大并发函数实例数为1000。如果调用导致函数的并发数目超过了默认限制，则该调用会被阻塞，将不会执行这次调用。

- 公式：并发函数实例数 = 每秒请求量 × 函数执行时间（按秒）
- 例子：假定函数平均用时 0.2 秒（即 200 毫秒），每秒 300 个请求至函数。这样将同时生产 300 × 0.2 = 60个函数实例。

> **注意**
> 如果想达到函数的最大并发函数实例数 1000，函数平均耗时为 0.2 秒，则每秒请求数（QPS）需要达到 5000。

安装 Node.js 依赖

最近更新时间：2023-09-13 16:55:05

云函数安装 Node.js 依赖有两种方式：**本地 npm 安装**和**在线依赖安装**。

本地 npm 安装

使用 npm 安装第三方依赖，只能对每个云函数分别安装依赖。进入函数代码根目录，通过终端执行以下命令安装 request 库。

```
npm install request --save
```

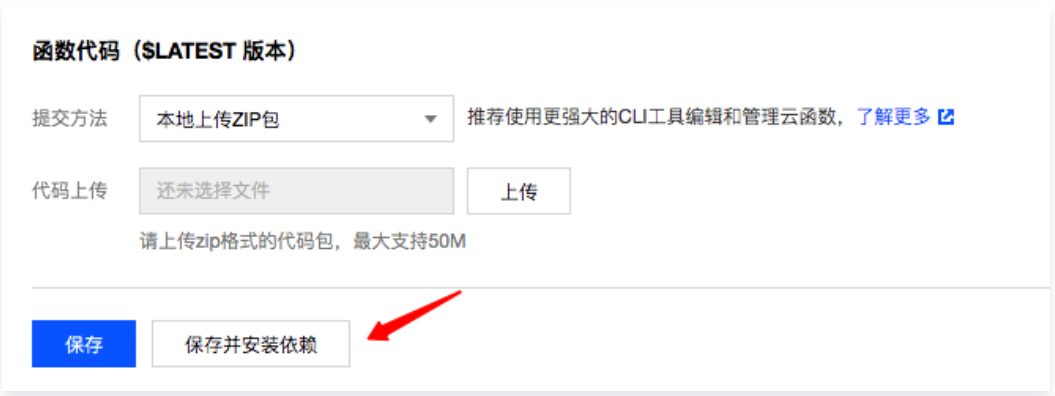
安装成功的依赖文件会作为该云函数代码的一部分，手动上传到云端使用。

在线依赖安装

CloudBase 提供了云端安装依赖，免去了在终端手动安装依赖的工作。

云开发 CloudBase 控制台

登录 云开发 CloudBase 控制台，在 函数编辑页面，在线编辑或者上传 zip 代码包之后，单击**保存并安装依赖**。



CloudBase CLI 工具

在配置文件 **cloudbaserc.json** 对应的云函数的配置项中添加。

```
installDependency:true
```

示例如下：

```
{
  "envId": "xxx",
  "functionRoot": "./functions",
  "functions": [
    {
      "name": "app",
```

```
      "config": {
        // 超时时间
        "timeout": 5,
        // 环境变量
        "envVariables": {
          "key": "value"
        },
        "runtime": "Nodejs10.15",
        "installDependency": true
      },
      // 调用云函数时的输入参数
      "params": {},
      "handler": "index.main"
    }
  ]
}
```

定时触发器

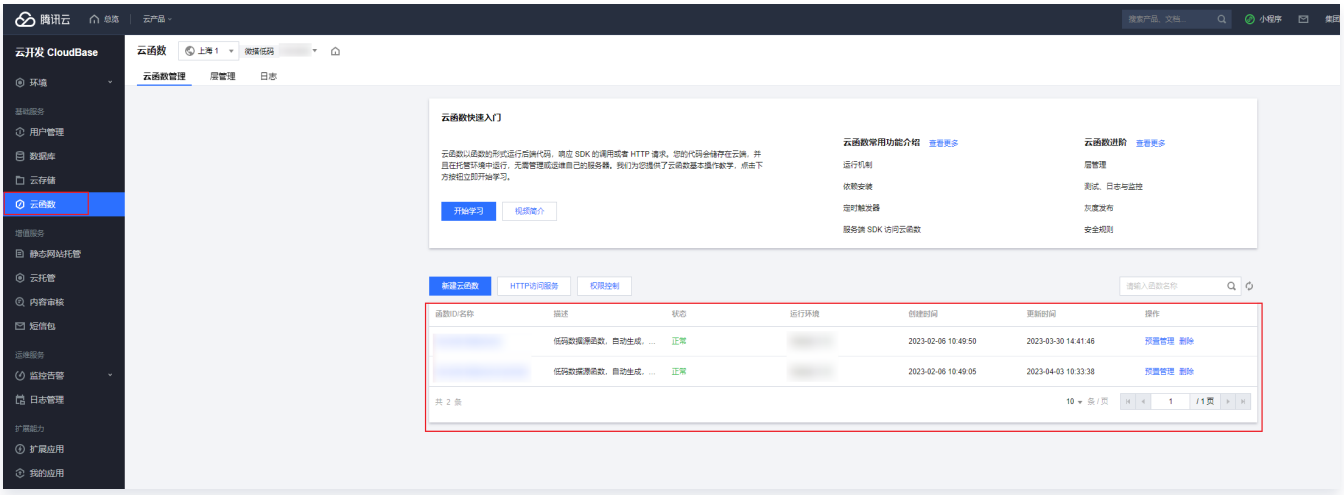
最近更新时间：2024-11-14 11:46:02

您可以使用定时触发器结合云函数，实现定时任务的功能。

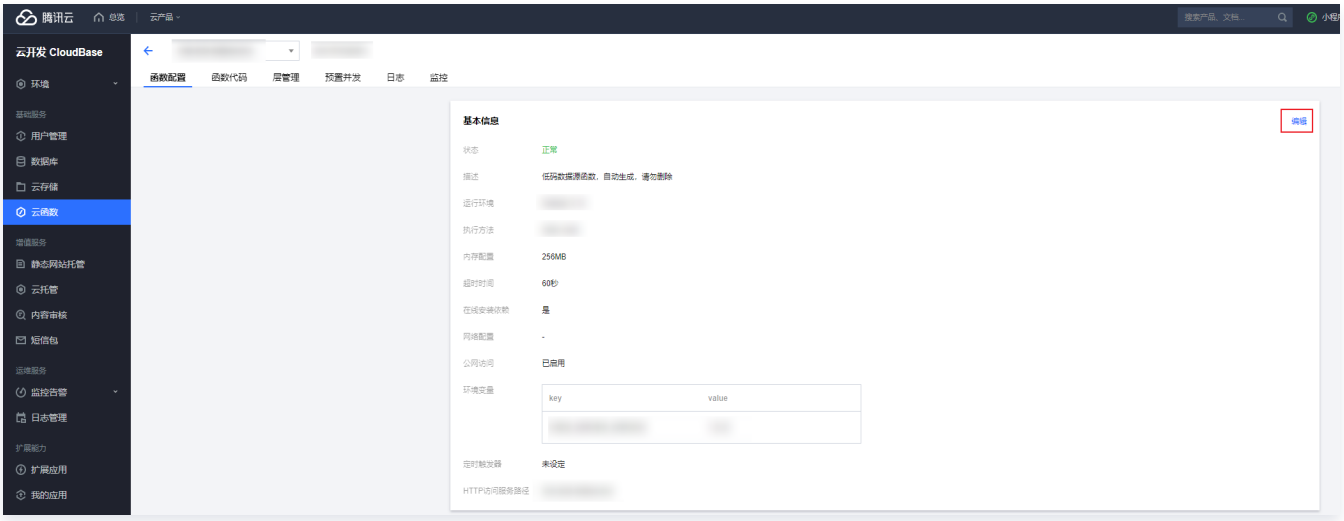
设置定时触发器

云开发 CloudBase 控制台

1. 进入 云函数管理页面，选择要配置的函数。



2. 单击**编辑**，修改表单的定时触发器选项，可以上传配置文件或配置内容。



CloudBase CLI 工具

详情请参见 CloudBase CLI 文档 。

配置示例

```
{
  // triggers 字段是触发器数组，目前仅支持一个触发器，即数组只能填写一个，不可添加多个
  "triggers": [
    {
      // name: 触发器的名字，规则见下方说明
      "name": "myTrigger",
      // type: 触发器类型，目前仅支持 timer （即定时触发器）
      "type": "timer",
      // config: 触发器配置，在定时触发器下，config 格式为 cron 表达式，规则见下方说明
      "config": "0 0 2 1 * * *"
    }
  ]
}
```

配置详解

字段规则

- 定时触发器名称（name）：最大支持 60 个字符，支持 `a-z` 、`A-Z` 、`0-9` 、`-` 和 `_` 。必须以字母开头，且一个函数下不支持同名的多个定时触发器。
- 定时触发器触发周期（config）：指定的函数触发时间。填写自定义标准的 Cron 表达式来决定何时触发函数。有关 Cron 表达式的更多信息，请参见以下内容。

Cron 表达式

Cron 表达式有七个必需字段，按空格分隔。其中，每个字段都有相应的取值范围：

排序	字段	值
第一位	秒	0 – 59 的整数
第二位	分钟	0 – 59 的整数
第三位	小时	0 – 23 的整数
第四位	日	1 – 31 的整数（需要考虑月的天数）
第五位	月	1 – 12 的整数或 JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV 和 DEC

第六位	星期	0 – 6 的整数或 MON、TUE、WED、THU、FRI、SAT 和 SUN，其中 0 指周日，1 指星期一，以此类推
第七位	年	1970 – 2099 的整数

通配符

通配符	含义
`,`	代表取用逗号隔开的字符的并集。例如：在“小时”字段中 1，2，3 表示 1 点、2 点和 3 点。
`-`	包含指定范围的所有值。例如：在“日”字段中，1 – 15 包含指定月份的 1 号到 15 号。
`*`	表示所有值。在“小时”字段中，表示每个小时。
`/`	指定增量。在“分钟”字段中，输入 1/10 以指定从第一分钟开始的每隔十分钟重复。例如，第 11 分钟、第 21 分钟和第 31 分钟，以此类推。

说明：
在 Cron 表达式中的“日”和“星期”字段同时指定值时，两者为“或”关系，即两者的条件均生效。

cronjob 示例

下面列举一些 Cron 表达式和相关含义：

- `*/5 * * * * * *` 表示每 5 秒触发一次。
- `0 0 2 1 * * *` 表示在每月的 1 日的凌晨 2 点触发。
- `0 15 10 * * MON-FRI *` 表示在周一到周五每天上午 10:15 触发。
- `0 0 10,14,16 * * * *` 表示在每天上午 10 点，下午 2 点，下午 4 点触发。
- `0 */30 9-17 * * * *` 表示在每天上午 9 点到下午 5 点内每半小时触发。
- `0 0 12 * * WED *` 表示在每个星期三中午 12 点触发。

使用服务端 SDK 访问 CloudBase

最近更新时间：2023-09-13 16:55:05

如需在云函数中访问 CloudBase 的各项服务，例如操作数据库、管理云文件等，可使用 CloudBase 服务端 SDK。
例如，您可以在 Node.js 云函数中，使用 CloudBase Node.js SDK 调用 CloudBase 服务。

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({
  env: cloudbase.SYMBOL_CURRENT_ENV
});

const db = app.database();

exports.main = async (event, context) => {
  return db.collection("todos").get();
};
```

说明
CloudBase 服务端 SDK 已经与云函数进行集成，无需手工填入密钥即可使用。

初始化 SDK

```
const cloudbase = require("@cloudbase/node-sdk");
const app = cloudbase.init({
  env: cloudbase.SYMBOL_CURRENT_ENV
});
```

调用云数据库

```
const db = app.database();
exports.main = async (event, context) => {
  return db.collection("todos").get();
};
```

调用云存储

```
exports.main = async (event, context) => {
  const fileStream = fs.createReadStream(path.join(__dirname, "demo.jpg"));
  return await app.uploadFile({
    cloudPath: "demo.jpg",
    fileContent: fileStream
  });
};
```

调用其它云函数

```
exports.main = async (event, context) => {
  return await cloud.callFunction({
    name: "sum",
    data: {
      x: 1,
      y: 2
    }
  });
};
```

获取用户信息

当从客户端调用云函数时，如在小程序中或者 web 端使用微信登录授权，云函数的传入参数中会被注入用户的 `openid`，开发者无需校验 `openid` 的正确性，可以直接使用该 `openid`。与 `openid` 一起注入云函数的还有其它相关的用户身份信息。

Node.js

```
//引用SDK
const tcb = require("@cloudbase/node-sdk");
//初始化SDK
const app = tcb.init();
//获取用户信息
const userInfo = await app.auth().getUserInfo();
const {
  openId, //微信openId，非微信授权登录则空
  appId, //微信appId，非微信授权登录则空
  uid, //用户唯一ID
  customUserId //开发者自定义的用户唯一id，非自定义登录则空
} = userInfo;
```

小程序 · 云开发

从小程序端调用云函数时，开发者可以在云函数内使用 wx-server-sdk 提供的 getWXContext 方法获取到每次调用的上下文（ `appid` 、 `openid` 等），无需维护复杂的鉴权机制，即可获取可信任的用户登录态（ `openid` ）。
示例代码：

```
// index.js
const cloud = require("wx-server-sdk");
```

```
exports.main = async (event, context) => {
  // 这里获取到的 openId、 appId 和 unionId 是可信的，注意 unionId 仅在满足 unionId 获取
条件时返回
  const { OPENID, APPID, UNIONID } = cloud.getWXContext();

  return {
    OPENID,
    APPID,
    UNIONID
  };
};
```

参考

更多详细信息请参见：

- Node.js SDK 文档
- wx-server-sdk

灰度发布

最近更新时间：2024-06-11 15:03:31

CloudBase 云函数拥有多版本管理的功能，多个版本间可以使用函数灰度能力来**调整请求流量的比例**，达到线上业务可灰度、可回滚的能力，保证线上发布业务平滑过渡。

相关概念

版本

每个云函数可以发布多个**版本**，版本就是一个函数在生成版本时刻的快照，包含代码和配置（超时时间、环境变量等）。

LATEST 版本

云函数始终存在一个 LATEST 版本，即最新版本，您上传、更新、部署云函数的代码，都是更改 LATEST 版本。

流量比例

您可以调配不同版本云函数的流量比例，CloudBase 会根据比例，自动分发请求流量。

一致性保证

CloudBase 使用用户的全局唯一 ID 来保证请求的一致性。
设置了流量比例后，如果某用户被分配到了某个流量区间，则该用户调用该云函数时一定会走到该流量区间对应的云函数版本，而不会出现随机分配的现象。
例如，云函数的 LATEST 流量占比 10%，版本 1 占比 90%，如果某个用户经系统判断落在了版本 1 上，则此用户对该云函数的所有请求都**一定会落在版本 1 上**，而不是 90% 概率到版本 1，10 % 概率到 LATEST。
对于没有用户信息的请求，灰度默认是随机调度。

如何灰度

推荐的灰度流程如下：

1. 从 LATEST 上发布一个新版本，此版本的快照与当前 LATEST 一致。
2. 将 LATEST 版本的流量设置为 0，将新发布的版本流量设置为 100%。
3. 更新云函数代码（此时 LATEST 版本已经更新，但流量为 0）。
4. 自行调配 LATEST 版本灰度比例。

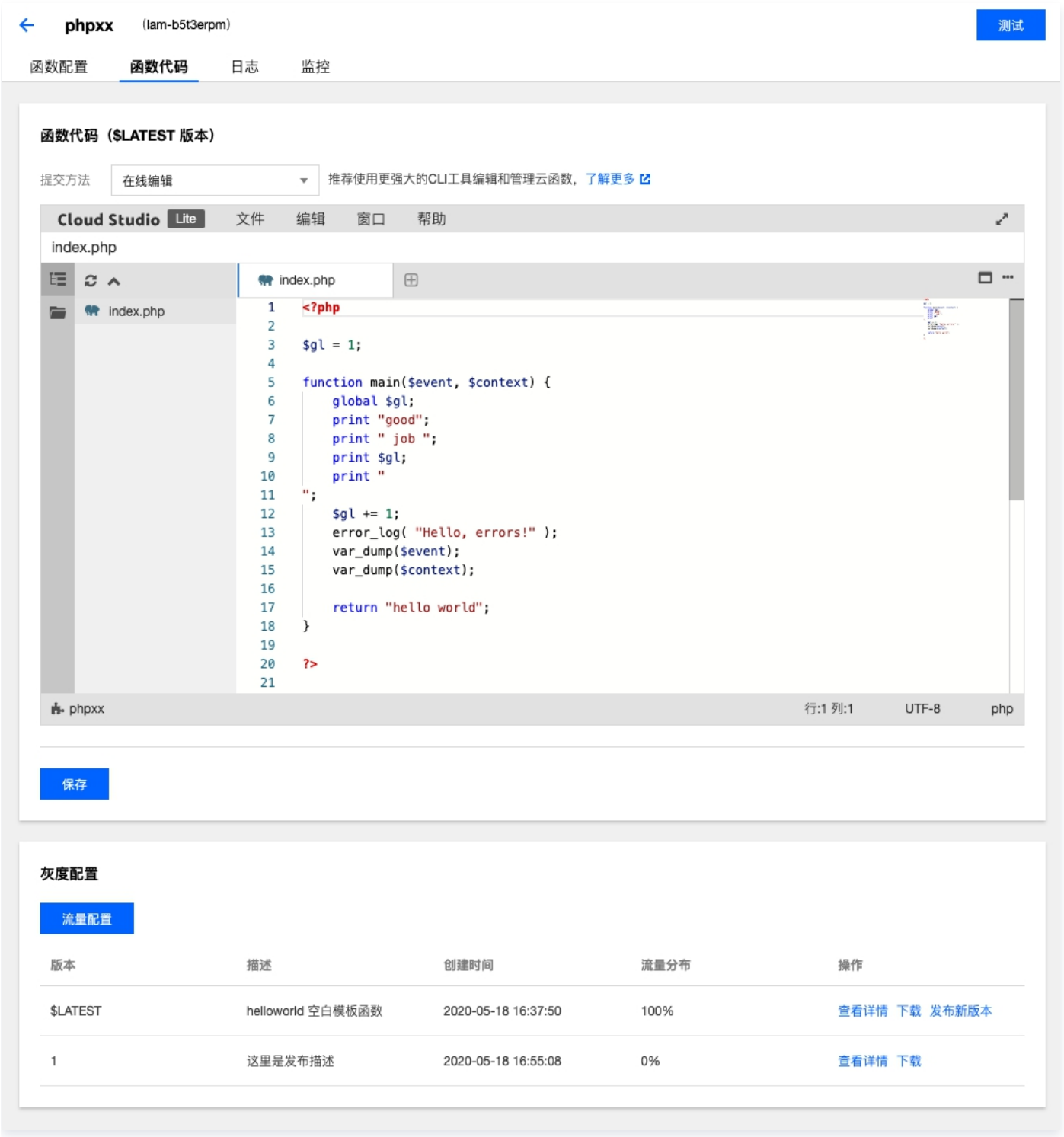
例如：

1. 发布版本 1。
2. 设置 100% 流量到版本 1。
3. 更改 LATEST 代码，并且部署。
4. 此时希望 10% 的线上流量给到需要灰度观察的最新代码，则设置 10% 流量给到 LATEST，90% 流量给版本 1。

灰度使用

生成版本

登录 CloudBase 云开发控制台，进入 云函数管理页面，单击想要灰度的函数：



在**灰度配置**中，单击**发布新版本**对 LATEST 生成一个新的版本，输入版本描述后，即可生成版本 1。

> **说明：**
> 生成版本后将**无法更改**此版本的代码。

配置流量比例

配置不同版本中的灰度比例，如下：

这里注意调整比例后，**业务立马生效，请谨慎配置后确认。**

实践教程

第一次使用

第一次云函数灰度时，只有 LATEST 版本，没有任何生成的版本，此时需要开发者先为当前 LATEST 版本发布一个版本：

1. 在控制台中选择云函数，发布新版本 1，将流量设置为 100%。
2. 更新 LATEST 版本，此时由于 LATEST 版本的流量为 0，所以不影响线上业务。
3. 将 LATEST 和 1 的流量比例分别设为 10% 和 90%。

此时完成第一次线上灰度，新版本（更新后的 LATEST）占比流量为 10%，老版本（版本 1）占比流量为 90%。

版本回退/全量

全量和回退版本，只需要把流量调整，使得 `100%` 导向指定灰度版本/正式版本即可。

测试、日志与监控

最近更新时间：2025-05-09 17:17:12

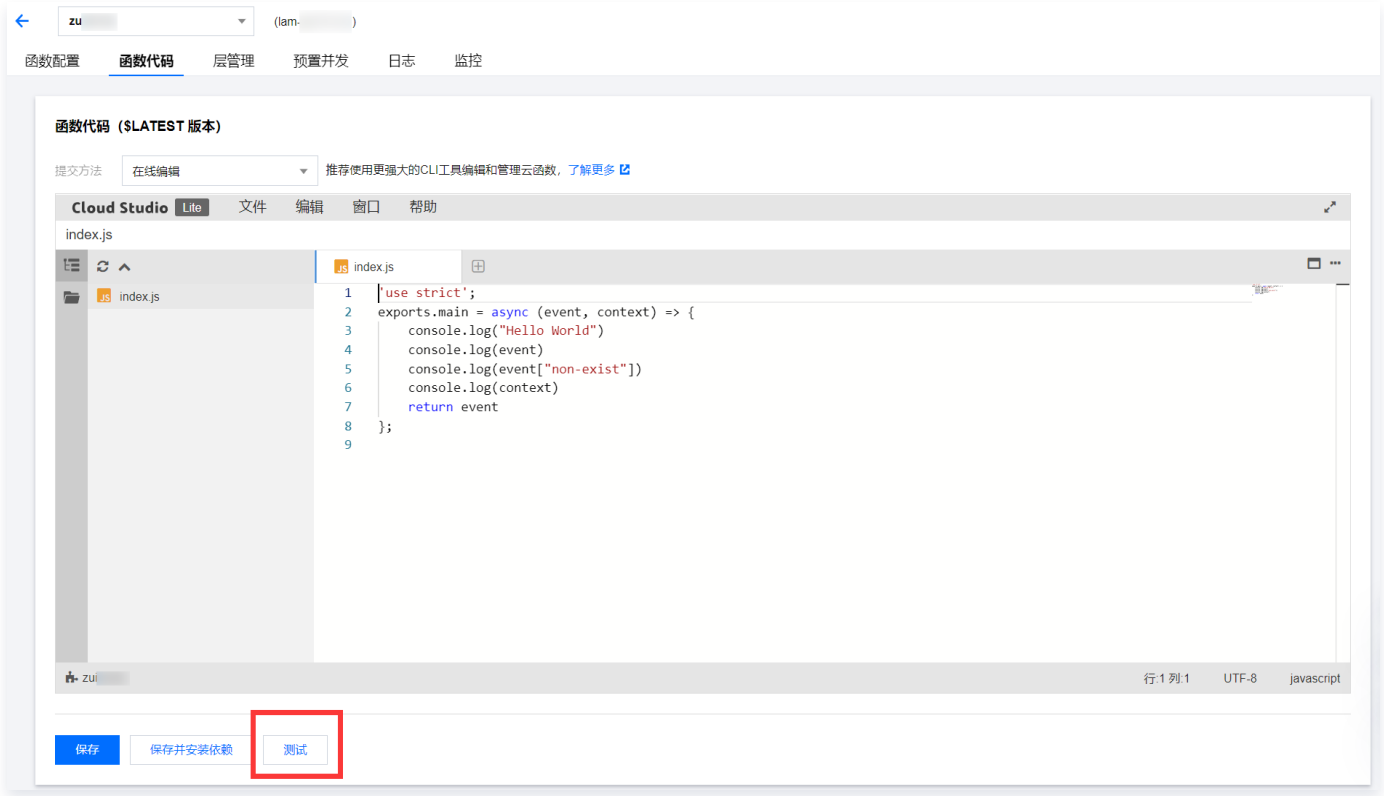
本文档指导您如何在云开发控制台进行测试、调用日志和监控数据。

测试

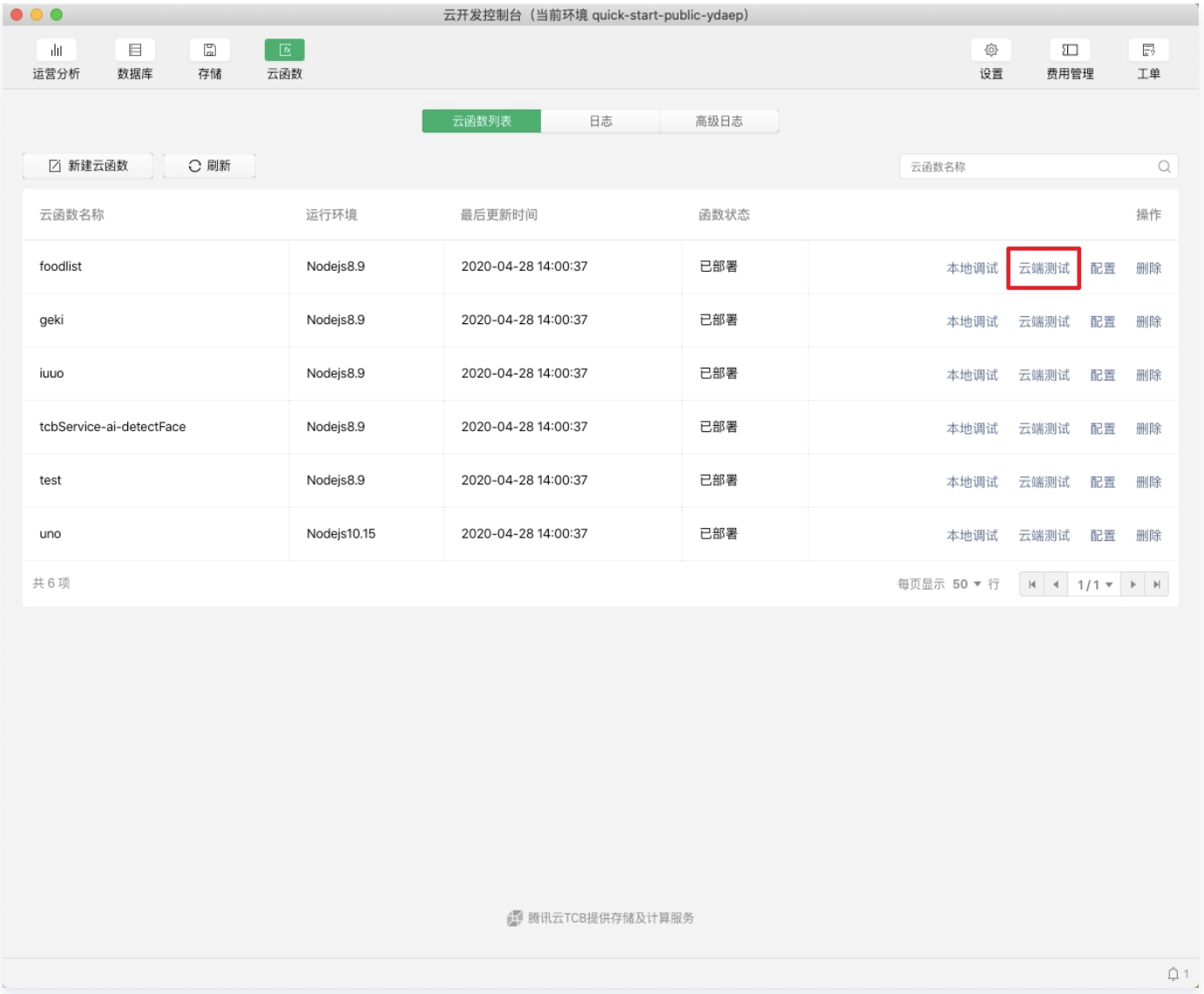
云开发提供了云函数测试功能，可以更加方便地调试您的代码。

1. 进入 云开发平台 > **计算** > **云函数**页面，选择并进入对应云函数的管理面板。
2. 在**函数代码**中单击**测试**，即可打开测试弹窗。

云开发 CloudBase 控制台



微信开发者工具



3. 单击**提交方法**下拉菜单，可以选择测试函数的模板方法，当前只支持 `Hello World` 事件模板。模板在测试时作为 event 参数传递给函数。
4. 在**测试参数**的编辑器中输入想测试的参数后，单击**执行**，即可运行代码。执行完毕后，运行结果将显示在**运行测试**栏中。

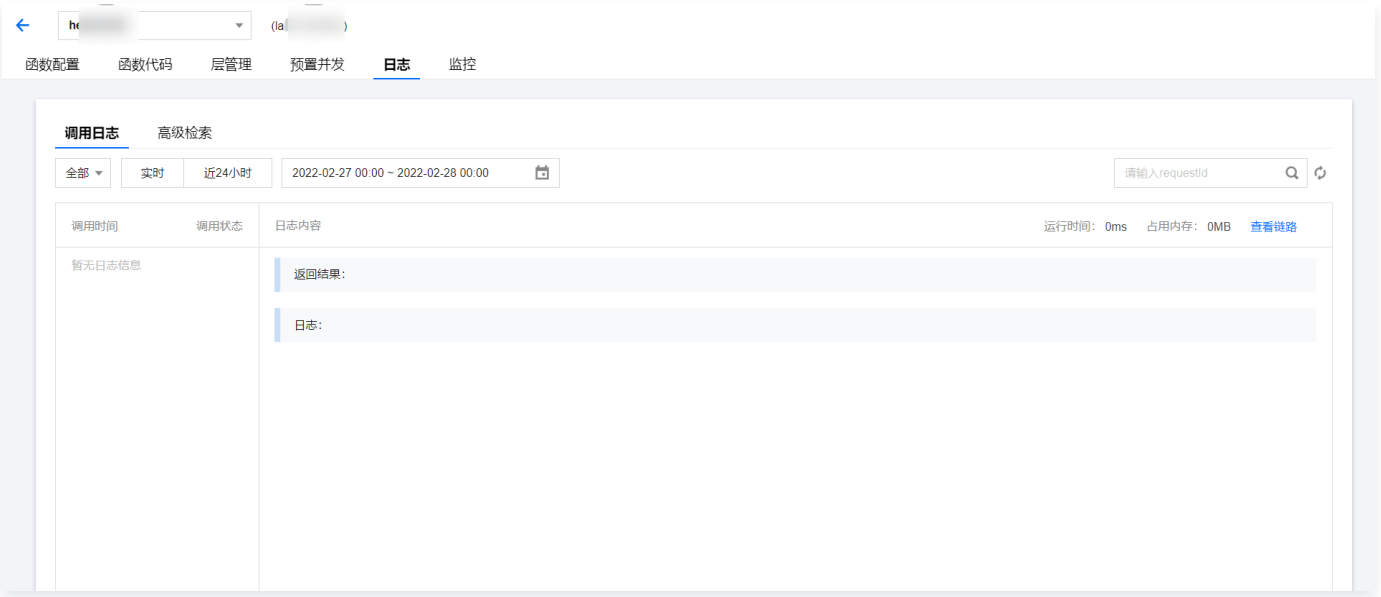
除了可视化的云函数测试功能，我们还提供命令行工具 scf-cli，帮助您在本地快速调试。

日志

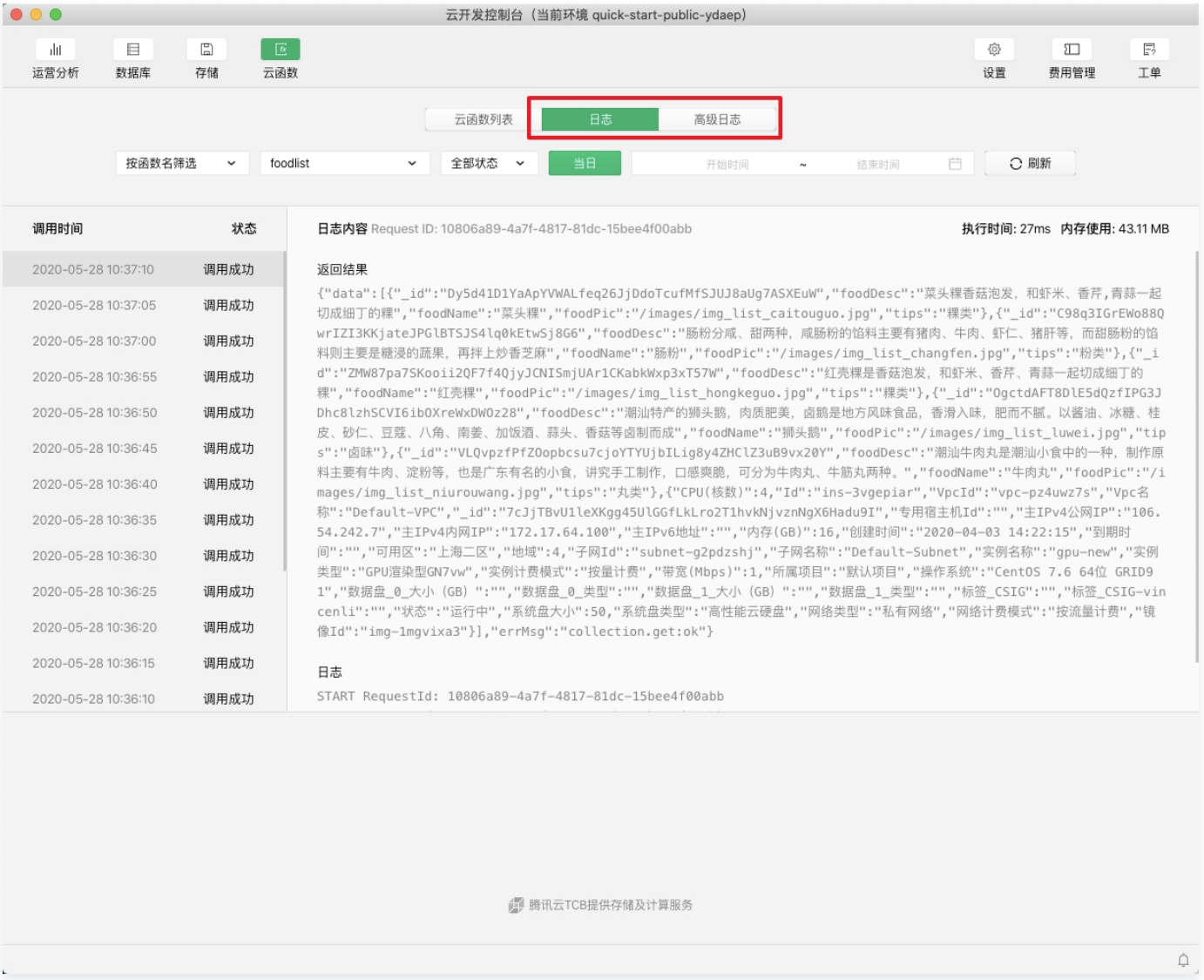
1. 进入 云开发平台 > **计算** > **云函数**页面，选择并进入对应云函数的管理面板。
2. 选择**日志**，进入日志页面，您可以查看云函数的调用日志，方便开发者对代码进行调试。

> **说明**
> 云开发提供了更加高级的日志系统，详情请参见 日志管理概述 。

云开发 CloudBase 控制台



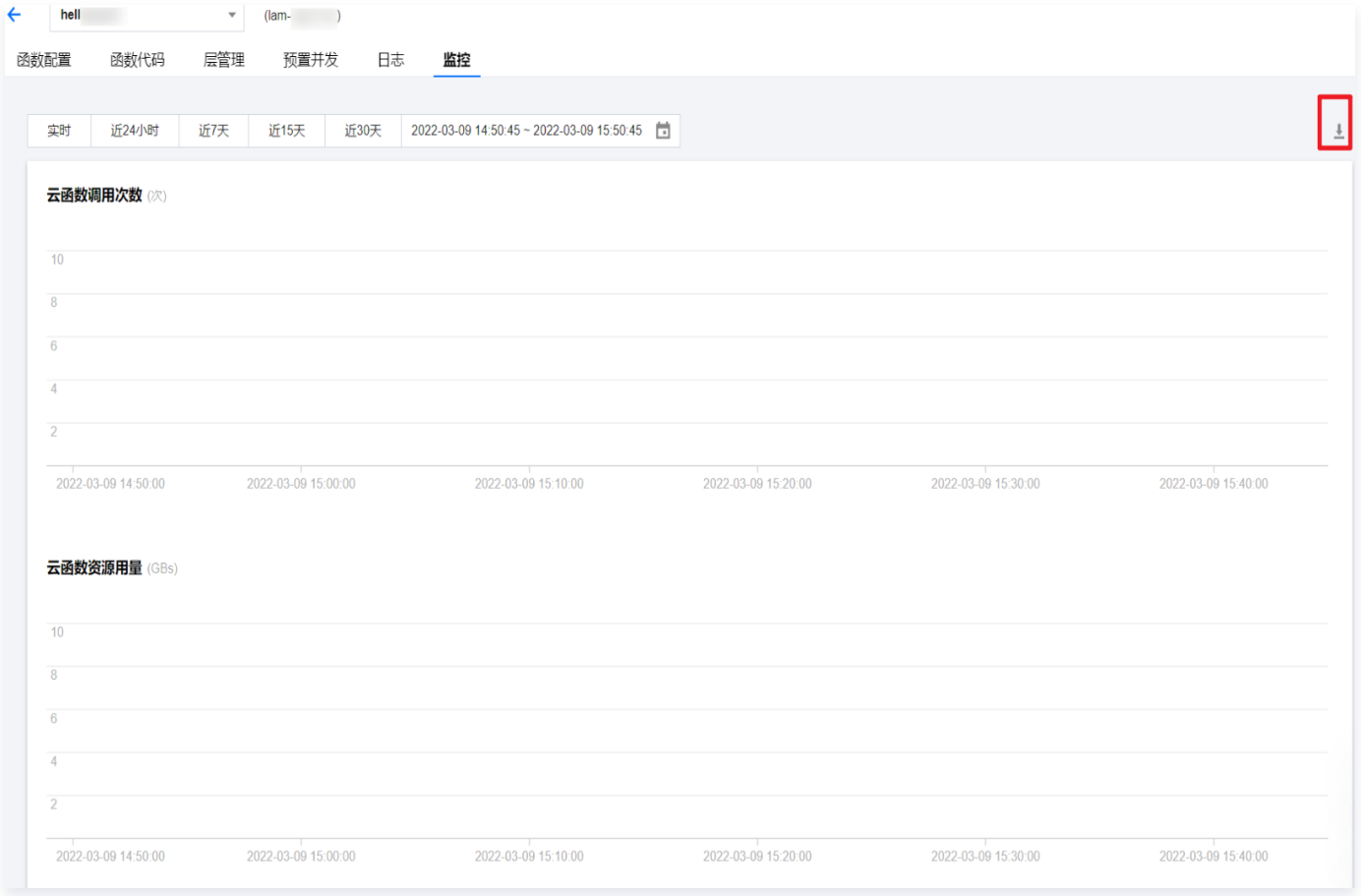
微信开发者工具



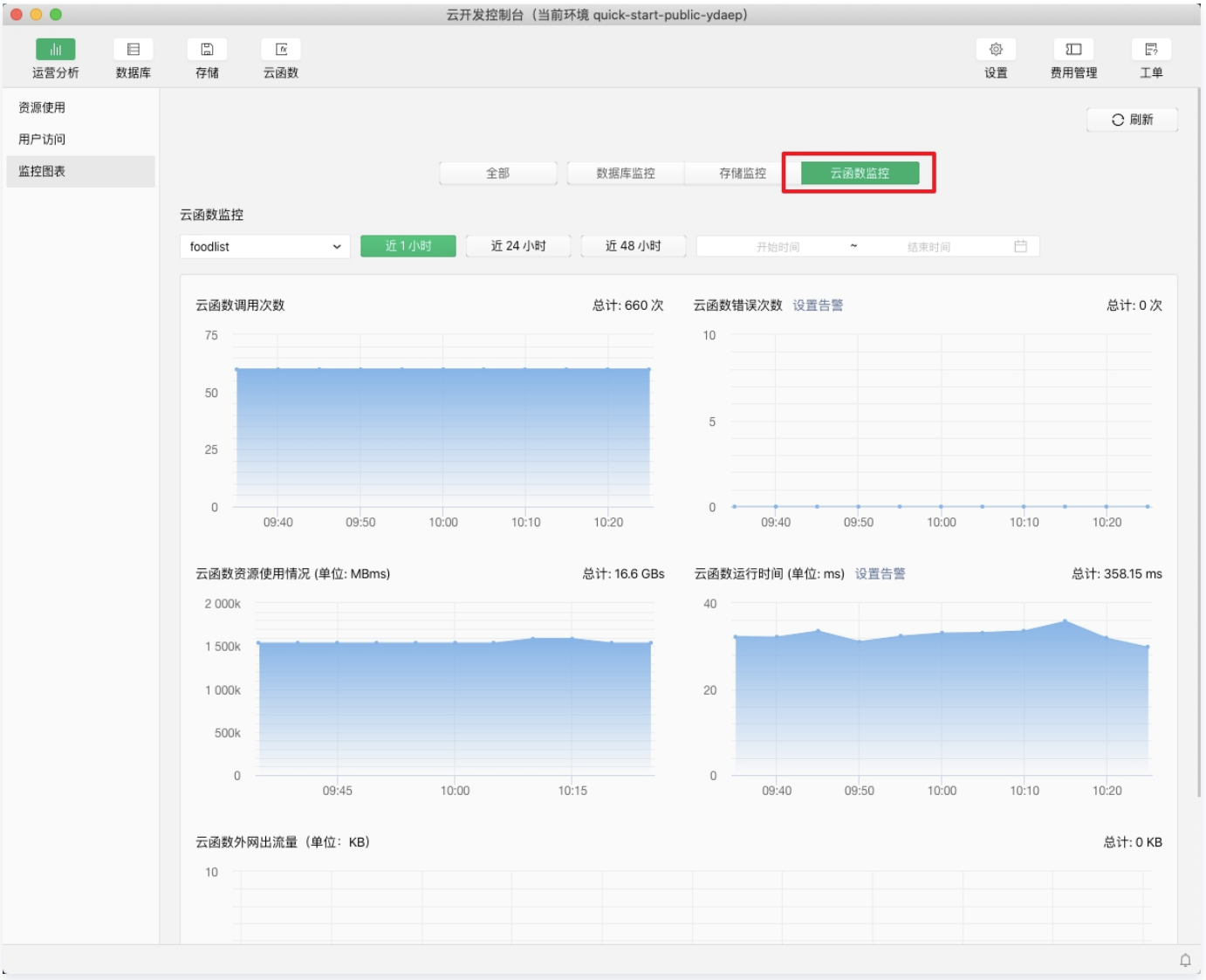
监控

1. 进入 云开发平台 > **计算** > **云函数**页面，选择并进入对应云函数的管理面板。
2. 选择**监控**，进入监控页面，您可以查看云函数的调用次数、运行时间、错误次数。
3. 单击下载图标，您可以将这些数据全部导出。

云开发 CloudBase 控制台


hell
(lam-
函数配置
函数代码
层管理
预置并发
日志
监控
实时
近24小时
近7天
近15天
近30天
2022-03-09 14:50:45 ~ 2022-03-09 15:50:45
云函数调用次数 (次)
10
8
6
4
2
2022-03-09 14:50:00
2022-03-09 15:00:00
2022-03-09 15:10:00
2022-03-09 15:20:00
2022-03-09 15:30:00
2022-03-09 15:40:00
云函数资源用量 (GBs)
10
8
6
4
2
2022-03-09 14:50:00
2022-03-09 15:00:00
2022-03-09 15:10:00
2022-03-09 15:20:00
2022-03-09 15:30:00
2022-03-09 15:40:00

微信开发者工具


云开发控制台（当前环境 quick-start-public-ydaep）
运营分析
数据库
存储
云函数
设置
费用管理
工单
资源使用
用户访问
监控图表
刷新
全部
数据库监控
存储监控
云函数监控
云函数监控
foodlist
近 1 小时
近 24 小时
近 48 小时
开始时间
~
结束时间
云函数调用次数
总计: 660 次
云函数错误次数
设置告警
总计: 0 次
75
50
25
0
09:40
09:50
10:00
10:10
10:20
10
5
0
09:40
09:50
10:00
10:10
10:20
云函数资源使用情况 (单位: MBms)
总计: 16.6 GBs
云函数运行时间 (单位: ms)
设置告警
总计: 358.15 ms
2 000k
1 500k
1 000k
500k
0
09:45
10:00
10:15
40
20
0
09:40
09:50
10:00
10:10
10:20
云函数外网出流量（单位：KB）
总计: 0 KB
10

深入理解云函数

最近更新时间：2024-11-14 11:46:02

云函数中，Node.js 的运行机制和本地 Node.js 运行的行为有些差异。

启动时间

云函数存在两种启动：

- **冷启动**：需要平台分配计算资源、加载代码、启动 Node.js 进程，**耗时较长。**
- **热启动**：函数实例、执行进程都被复用（即下文提到的**实例复用**），**耗时很短**。

如果请求到来时没有正在运行的实例，请求会触发实例的启动。实例启动通常需要一些时间，这会使触发实例启动的调用增加一些耗时。通常首次调用函数、更新函数或长时间未调用时重新调用函数时才会触发实例启动。
在实例处理完函数请求后会根据平台的实际情况存留一段时间以用于下次调用，在此时间段内的调用会优先复用该实例。

> **说明：**
> CloudBase 会根据您云函数长期的访问情况，自动调度、调配实例的数量，保证足够好的性能的同时，节省您的资源。

实例复用

考虑下面这个云函数：

```
let i = 0;
exports.main = async (event = {}) => {
  i++;
  console.log(i);
  return i;
};
```

在第一次调用该云函数的时候函数返回值为 1，这是符合预期的。
但如果连续调用这个云函数，**其返回值有可能是从 2 递增，也有可能变成 1**，这便是实例复用的结果：

- 当**热启动**时，执行函数的 Node.js 进程被复用，进程的上下文也得到了保留，所以**变量** `i` **自增**。
- 当**冷启动**时，Node.js 进程是全新的, 代码会从头完整的执行一遍，**此时返回 1**。

所以，**开发者在编写云函数时，应注意保证云函数是无状态的、幂等的**，即当次云函数的执行不依赖上一次云函数执行过程中在运行环境中残留的信息。

> **说明：**
> 关于实例复用更加详细的说明，可以参考此文档中 实例复用 章节。

时区

云函数中的时区默认是 UTC+0，在函数中获取本地时间会和北京时间有 8 个小时的差异。
定义环境变量 `TZ` 可以改变函数运行时的时区，例如设置 `TZ` 为 `Asia/Shanghai` 可以指定函数的时区为北京时间。
更多关于时区的信息，请参见 Time Zone Database 。

> **说明：**

> 在服务端处理时间（包括云数据库和云函数）应尽量**避免使用受到时区影响的本地时间**，而是**使用 Unix 时间戳这样的绝对值**，这样可以避免服务端和客户端时区差异带来的众多问题。

Node.js 8 的异步行为

考虑下面这段代码：

```
exports.main = async (event = {}, context) => {
  setTimeout(() => {
    console.log("rid: ", context.request_id);
  }, 0);
  return "ok";
};
```

在本地调用时，函数返回 **ok**，并且随后可以看到 requestId 的打印。
但如果在 Node.js 8 的云函数中运行此代码，函数依然返回 **ok**，**但是** `setTimeout` **中异步函数打印的内容却不会在这次调用日志里看到**，例如：

日志内容 Request ID: 85ee137c-0cfd-11ea-8141-525400235f2a　　执行时间: 1.1ms　内存使用: 23.84 MB

返回结果

```
"ok"
```

日志

```
START RequestId: 85ee137c-0cfd-11ea-8141-525400235f2a
Event RequestId: 85ee137c-0cfd-11ea-8141-525400235f2a

END RequestId: 85ee137c-0cfd-11ea-8141-525400235f2a
Report RequestId: 85ee137c-0cfd-11ea-8141-525400235f2a Duration:1ms Memory:256MB MaxMemoryUsed:23.843750MB
```

如果继续调用第二次，您可能会在调用日志里看到上次 `setTimeout` **中打印：**

日志内容 Request ID: 87e7ae24-0cfd-11ea-a4a6-525400192d0e　　执行时间: 0.27ms　内存使用: 23.59 MB

返回结果

```
"ok"
```

日志

```
START RequestId: 87e7ae24-0cfd-11ea-a4a6-525400192d0e
Event RequestId: 87e7ae24-0cfd-11ea-a4a6-525400192d0e
2019-11-22T07:56:03.459Z        85ee137c-0cfd-11ea-8141-525400235f2a    rid:  85ee137c-0cfd-11ea-8141-525400235f2a

END RequestId: 87e7ae24-0cfd-11ea-a4a6-525400192d0e
Report RequestId: 87e7ae24-0cfd-11ea-a4a6-525400192d0e Duration:0ms Memory:256MB MaxMemoryUsed:23.593750MB
```

这一点是很多开发者感到困惑的地方。
对于异步函数，主流程（例如示例中的 `await main(event, context)`）执行完成后，函数实例进程会**被冻结**，进程中的所有异步任务会暂停执行，直到这个进程被再次唤起。
另一方面，如果函数实例进程由于某些原因**没被复用**（例如更新了函数代码），这个异步流程中的代码就**永远不会被执行**。

> **说明：**

> 开发者**不应将关键代码放入云函数异步流程中**。
> 如果想让关键代码稳定运行，请将其**放到函数主流程中**，或者**使用 Node.js v10 或 Node.js v12 版本的云函数**。

示例

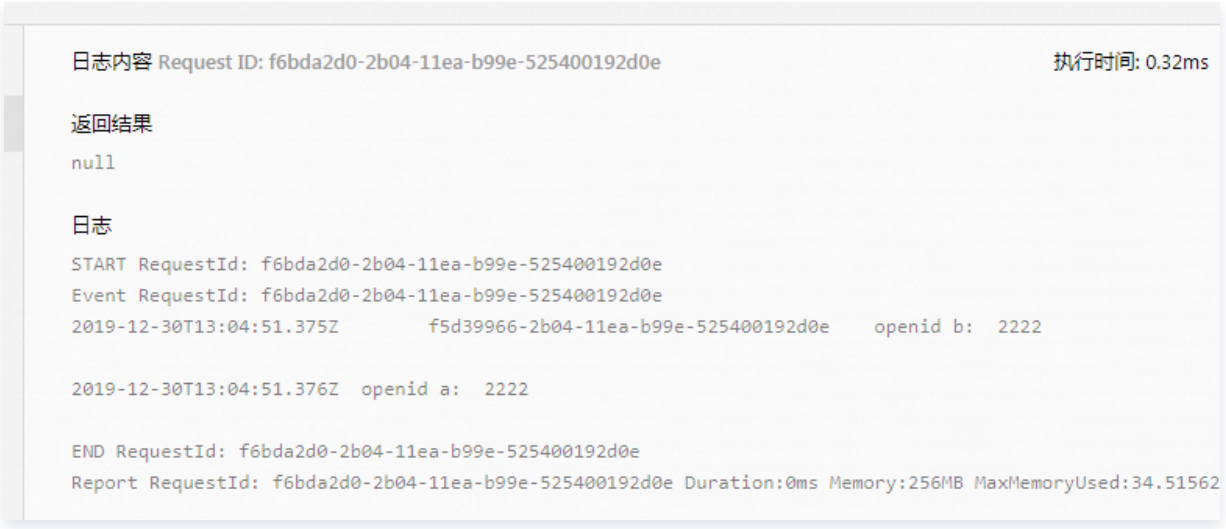
Node.js 8 云函数的异步特性可能会带来某些预期之外的行为，例如：
小程序 `wx-server-sdk` 提供了 `getWXContext` 方法来获取函数的一些上下文信息（appId，openId，unionId）来方便开发者，该方法本质是从环境变量中读取若干参数。
如果在异步流程中使用该方法获取 `openId` 或 `unionId`，可能会产生**身份漂移**的异常情况。
考虑这样的一段代码：

```
const cloud = require("wx-server-sdk");

exports.main = async (event) => {
  console.log("openid a: ", cloud.getWXContext().OPENID);
  setTimeout(() => {
    const { OPENID } = cloud.getWXContext();
    console.log("openid b: ", OPENID);
  }, 0);
};
```

假如两个不同的用户通过小程序访问该函数，用户 A 的 openid 为 1111，用户 B 的 openid 为 2222。
他们两次调用的日志将分别为：

- 日志1：

- 日志2：



可以看到，第一次调用打印出了用户 A 的 openid，这是符合预期的，但**异步流程的逻辑当次调用并没有执行**。
第二次调用时，函数实例进程得到复用，**第一次调用产生的异步逻辑继续运行**，打印出了用户 B 的 openid，而这次调用其实是**属于用户 A 调用函数的异步流程**。

云函数安全规则

最近更新时间：2025-02-14 11:41:12

身份验证

云函数安全规则是基于用户身份的权限管理系统，可以根据当前登录用户的身份进行函数调用权限管理控制。云函数安全规则适用于普通的 C 端身份调用，例如各种客户端的 `callFunction` 操作，管理端触发、HTTP 触发以及触发器触发不在安全规则的管控范围内。

操作步骤

1. 登录云开发平台，进入 云函数 页面，单击**权限控制**。

2. 编写完成**安全规则**后单击**确定**。

> **说明：**
> - 云控制台和服务端始终有所有云函数调用权限，以下配置仅针对客户端发起的云函数调用有效。
> - 权限修改后现网生效需要1-3分钟，请耐心等待。

编写安全规则

函数安全规则配置在环境级别，即环境内所有函数共享一个配置，配置中通过配置层级控制单个函数行为。
与所有安全规则相同，规则的配置基于 JSON 整体配置，但云函数安全规则配置层级更多：

- 顶级 key 表征函数名，特殊的 `*` 表征所有的函数名通配，匹配时优先匹配函数名，当未匹配时将使用 `*` 的配置。value 为每个函数单独的调用规则子配置。

- 每个子配置中，key 表示操作名（当前只支持 `invoke`），value 为 boolean 值或安全规则表达式字符串，例如：

```
{
  "*": {
    "invoke": "auth != null"
  },
  "function1": {
    "invoke": false
  }
}
```

限制

- 安全规则顶级配置**必须**包含 key 为 `*` 的配置。
- 每个函数下的配置中**必须**包含 invoke 配置。
- 云函数安全规则暂时只支持有限的 3 种配置，`true`、`false`、`"auth!=null"` 分别表示允许调用、不允许调用、以及登录后可调用。默认情况下为登录后可调用，即：

```
{
  "*": {
    "invoke": "auth != null"
  }
}
```

时区设置

最近更新时间：2024-12-27 16:25:12

云函数的运行环境内保持的是 UTC 时间，即 0 时区时间，和北京时间有 8 小时的时间差。可以通过语言的时间处理相关库或代码包（如 moment-timezone ），识别 UTC 时间并转换为+8区北京时间。

> **注意**
> 请注意云函数运行时的 Node 版本，仅 Node 15+ 版本支持设置环境变量 TZ=Asia/Shanghai 指定时区。

参考代码：

```
const moment = require("moment-timezone"); // 需在 package.json 中指定并安装依赖

exports.main = async (event, context) => {
  // javascript date
  console.log(new Date()); // 2021-03-16T08:04:07.441Z (UTC+0)
  console.log(moment().tz("Asia/Shanghai").format()); // 2021-03-16T16:04:07+08:00
(UTC+8)
};
```

预置并发

最近更新时间：2025-04-23 18:18:43

并发概述

并发是云函数在某个时刻同时处理的请求数。在业务其他服务可以支撑的情况下，您可以通过简单的配置实现云函数从几个并发到数以万计并发的拓展。

并发运行原理

在调用函数时，云函数会分配一个并发实例处理请求或事件。函数代码运行完毕返回后，该实例会处理其他请求。如果在请求到来时，所有实例都在运行中，云函数则会分配一个新的并发实例。

> **说明：**
> 云函数遵循一个并发实例同一时刻仅处理一个事件的运行逻辑，保障每个事件的处理效率和稳定性。

并发的计算

云函数的并发指的是函数代码同时处理请求或调用的数量，您可以通过以下公式估算：
并发数 = 请求速率 × 函数运行时间 = 每秒请求次数 × 每个请求的平均耗时

> **说明：**
> 您可以在监控信息中的**运行时间**看到每个请求的平均耗时。
> 例如：某业务 QPS 为2000，每个请求的平均耗时为0.02s，则每个时刻的并发数为2000qps × 0.02s = 40

并发实例复用与回收

当并发实例处理完事件请求后，不会立刻被回收，而是会保留一段时间以便复用。在保留期内，如有新的请求事件需要处理，将会优先使用保留中的并发实例，从而实现事件的快速处理，无需重新启动并发实例。

> **注意：**
> - 保留期过后，如果没有请求需要该实例处理，云函数平台则会回收该实例。对于低并发的场景，不再设置保留期，平台将启动智能回收机制进行回收。
> - 并发保留的时间由云函数平台根据情况动态调整，故函数业务代码中不能假设某个特定保留时间进行程序编写。

并发扩容

如果请求到来时，没有该版本的并发实例可以处理该请求，云函数平台会启动新的并发实例来处理。新启动的并发实例在初始化的过程后，便可以处理事件，我们称之为由弹性并发带来的扩容。
在地域维度，每个账号的弹性并发的扩容速度默认限制为500个/分钟，即在1分钟内，最多可以启动500个新的并发实例。如在1分钟内已经达到了当前限制，则将无法再启动新的并发实例，持续到下1分钟。在此期间有新的并发扩容请求，将会产生扩容受限错误（429 ResourceLimit）。
例如，广州地域的账号默认并发额度可以支撑128MB 函数的1000个并发实例。有大量请求到来时，第一分钟可以从0个并发实例启动到500个并发实例。如果还有请求需要处理，第二分钟可以从500个并发实例启动到1000个并发实例，直至并发实例可以满足请求的需要或达到并发上限。

> **说明：**
> 500个/分钟的弹性并发扩容速度可以满足多数业务场景。如果您的业务遇到该扩容速度的限制，您可以选择使用预置并发进行预热或购买 预留资源套餐 以提高弹性并发扩容速度限制。

预置并发

云函数平台弹性并发扩容的并发实例需要经历初始化的过程：包括运行环境初始化及业务代码初始化等过程。您可以使用预置并发功能，预先配置并发实例。云函数平台将在您配置后开始启动并发实例，同时不会主动回收预置并发的实例，尽可能地保障有相应数量的并发实例。如遇到并发实例因代码内存泄漏等错误，云函数平台会将其替换为新的实例。

并发服务承诺

扩容并发限制

限制类型	速度
弹性扩容并发限制	500并发/分钟
预置扩容并发限制	100并发/分钟

在地域维度，弹性并发的扩容速度默认限制为500并发/分钟。例如客户有10w 并发的诉求，按照最大弹性并发的扩容速度，需要10w/500 = 200分钟就能完成扩容操作，需要提升配额可以购买 预留资源套餐 。

> **说明：**
> 并发额度配额：单个环境下最大为256,000MB。

预置并发

预置并发支持并发实例按配置预先启动，同时云函数平台不会主动回收这些实例，会尽可能地保障有相应数量的可以处理请求的并发实例。您可通过此功能，为函数的指定版本设定预置并发额度。通过配置预置并发，可预先进行计算资源的准备，降低冷启动、运行环境初始化及业务代码初始化引起的耗时。
预置并发是在版本维度上解决请求到来时遇到并发实例初始化的问题。当您为一个函数版本配置预置并发之后，将会有以下效果：

- 云函数平台立刻开始启动并发实例，直至达到配置值。
- 云函数平台不会主动回收预置并发实例，同时会尽可能地保障预置并发实例数。
- 预置并发与弹性调用的并发实例启动速度是分开的，预置并发的启动不会占用地域维度500个/分钟的弹性扩容速度。云函数平台会根据您业务的情况调整预置并发的启动速度，默认为100个/分钟。

> **注意：**
> - 云函数平台不会主动回收预置并发实例，但并发实例可能由于进程退出、内存超限等问题不可用。一旦有不可用的实例，云函数平台会回收同时准备新的并发实例以达到预置并发实例的配置。期间可能出现短暂的实际并发实例数小于预置并发实例的情况，未启动的并发实例不会纳入计费范围。您可以在函数的监控信息“并发执行个数和预置并发”图中查看预置并发启动情况。
> - 预置并发只能配置在已发布的版本上，无法配置在 $LATEST 版本上。$LATEST 版本处于可编辑态，而预置并发需要在请求到来前启动并发实例。为了保障业务的稳定，避免因代码和配置编辑带来的版本不一致问题，预置并发只能配置在已发布的版本上。已发布版本的代码和配置无法修改，适合生产环境使用。

针对函数已发布的版本，可以设定期望数量的预置并发数。登录 云开发平台 ，进入**云函数**页面。您可通过每个云函数的**预置管理**配置预置并发。

实践教程

并发是云函数在某个时刻同时处理的请求数。在业务其他服务可以支撑的情况下，您可以通过简单的配置实现云函数从几个并发到数以万计并发的拓展。

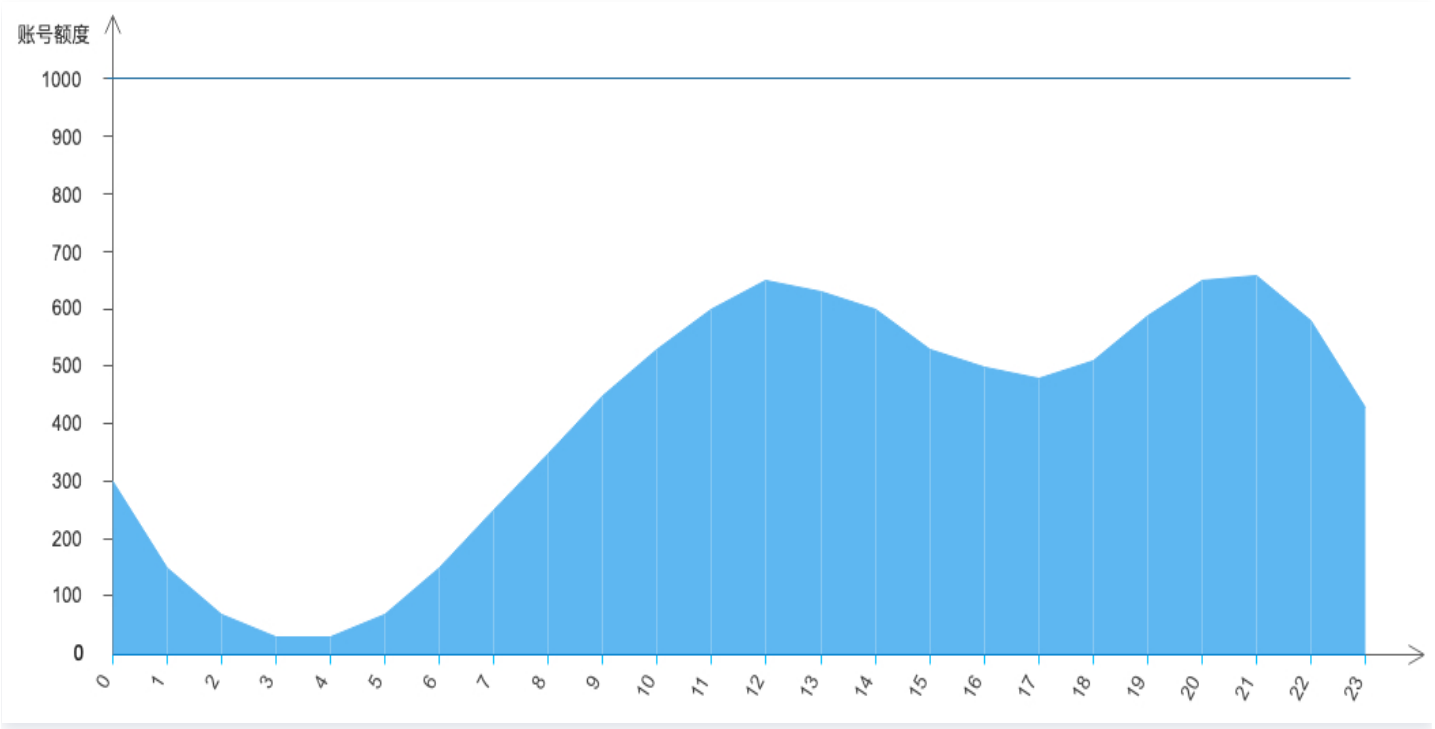
应用场景

高 QPS 短运行时长

使用云函数进行简单的数据、文件处理，例如云存储触发云函数进行信息上报、对文件处理等。此类场景下单次请求运行时间较短。

实践建议

一个账号下有多个业务同时使用云函数进行支撑时，云函数的并发配额则需要进行按需调度。例如，根据客户端业务的特点进行分析配置：客户端业务，会随着用户流量存在波峰波谷，为了保障用户体验，要求加载速度快，可以有一定的错误容忍度。
针对上述业务，可以为函数配置一定量的预置额度。如按最大使用量的60%来设置，但同时不配置函数的最大独占配额，确保在高峰到来时能充分利用总配额。云函数额度变化如下图所示：



云托管

概述

最近更新时间：2024-02-02 14:52:21

如需了解更多，请参见 云托管文档 。

HTTP 访问服务

概述

最近更新时间：2023-09-13 16:55:06

CloudBase 提供统一的 HTTP 访问服务，开发者可以通过标准的 HTTP 协议访问环境内的各种资源，无需使用 SDK 接入即可访问后端云资源。
HTTP 访问支持自定义资源的访问路径（URL Path）、自定义配置访问服务域名（CNAME）等能力，满足不同开发场景需要。
目前提供 HTTP 访问的资源有：

- 云函数
- 云托管
- 静态网站托管

功能和优势

- 每个环境拥有免费的默认域名，并原生支持 HTTPS。
- 高性能的 HTTP 协议访问。
- 支持针对域名、路径配置路由规则。
- 自定义绑定/解绑域名，可通过自定义域名访问环境服务。

使用限制

自定义域名仅能绑定 5 个。

快速开始

最近更新时间：2023-09-13 16:55:06

准备工作

1. 安装 CLI 工具。
2. 登录 CLI 工具。

初始化目录

```
mkdir my-cloudbase-service && cd my-cloudbase-service
mkdir functions && mkdir functions/hello
touch cloudbaserc.json functions/hello/index.js
```

然后我们获得了一个如下结构的目录：

```
.
├── cloudbaserc.json
└── functions
    └── hello
        └── index.js
```

`cloudbaserc.json` 内，填入环境 ID：

```
// cloudbaserc.json
{
  "envId": "your-env-id"
}
```

`functions/hello/index.js` 内，我们写入一个简单的 Hello World：

```
// functions/hello/index.js
exports.main = async function () {
  return "Hello World!";
};
```

发布云函数

执行以下命令：

```
cloudbase functions:deploy hello
```

等待之后，云函数便发布成功：

```
cloudbase functions:deploy hello
```

```
? 未找到函数发布配置，使用默认配置？ Yes
✔ [hello] 函数部署成功！
```

为云函数创建 HTTP 路由

执行以下命令创建一条HTTP 路由，路径为 `/hello` ，指向的云函数为 `hello` ：

```
cloudbase service:create -p /hello -f hello
```

通过 HTTP 访问云函数

随后便可以通过 `https://${env}.ap-shanghai.app.tcloudbase.com/hello` 直接访问函数，其中 `${env}` 是环境 ID，`ap-shanghai` 为地域，根据环境所在地域不同填写对应地域。

```
curl https://${env}.ap-shanghai.app.tcloudbase.com/hello
hello world!
```

也可以直接在浏览器内打开 `https://${env}.ap-shanghai.app.tcloudbase.com/hello` 。

HTTP 访问服务鉴权

最近更新时间：2025-04-23 18:18:43

HTTP 访问支持鉴定云开发的用户登录凭证，开发者在开启鉴权之后，可以在 HTTP 头部中加入 `<登录凭证>` 的方式调用云函数。

登录凭证添加方式：

- 采用 V2版本的方式登录，添加请求头：`Authorization: `Bearer <credentials>``
- 采用 V1版本的方式登录，添加请求头：`X-Cloudbase-Credentials: `<credentials>``

开启 HTTP 访问鉴权

在云开发平台的 HTTP 访问服务管理页面，可以为各条路径启动或关闭鉴权：

> **说明：**
> 开启访问鉴权之后，没有鉴权信息，或者鉴权信息非法的请求，都会请求失败。

使用 SDK 获取 HTTP 鉴权头部

以 Web 端 SDK 为例，使用云开发支持的登录方式登录后，可以使用 `Auth.getAuthHeader()` 获取鉴权头部：

```
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "xxxx"
});

/**
    执行登录流程，此处省略.....
*/

const authHeader = cloudbase.auth().getAuthHeader();
// { 'x-cloudbase-credentials': '<credentials>' }
```

在请求中加入鉴权头部

我们使用 axios 向 HTTP 服务的 URL 发起一个 HTTP 请求，其中加入鉴权头部：

```
const axios = require("axios");
const cloudbase = require("@cloudbase/js-sdk");
const app = cloudbase.init({
  env: "xxxx"
});

/**
    执行登录流程，此处省略……
*/

const authHeader = cloudbase.auth().getAuthHeader();

axios({
  method: "post",
  url: "https://env-id.service.tcloudbase.com/xxxx",
  data: {
    /* ... */
  },
  headers: {
    ...authHeader
  }
}).then((res) => {
  //...
});
```

HTTP 请求示例：

```
POST /aaa/bbb HTTP/1.1
Host: env-id.service.tcloudbase.com
X-Cloudbase-Credentials: <credentials>
<其它头部>: <...>

<传输体...>
```

如何使用云函数处理 HTTP 请求

最近更新时间：2024-01-17 15:29:21

本文档描述了 如何使用云函数处理HTTP请求，即 HTTP请求如何转换为云函数的入参，云函数的返回值如何转换为HTTP响应。阅读该文档前，请先阅读 快速开始 文档，了解如何创建云函数并通过 HTTP 访问云函数。另外，阅读本文档需要对云函数有一定的了解。

云函数代码

以下为一个简单的云函数代码片段：

```
exports.main = async (event, context) => {
  console.log('event: ', event)
  console.log('context: ', context)
  return { event, context }
}
```

这段代码并没有 HTTP服务相关的内容，并不会启动一个监听某端口的 HTTP 服务。那么就出现了一个问题：这个云函数怎么接收 HTTP 请求，怎么返回 HTTP 响应。本篇文档将详细解答这个问题。

云函数的入参

通过 HTTP 调用云函数时，HTTP 请求会被转化为称之为**集成请求**的特殊的结构体，并通过云函数入参 **event** 传递给云函数入口函数。
集成请求，就是通过 JSON 格式数据描述 HTTP 响应报文的一段数据。
集成请求结构如下：

```
{
  "path": "HTTP请求路径，如 /hello",
  "httpMethod": "HTTP请求方法，如 GET",
  "headers": {}, // HTTP请求头，key:value，value 为字符串
  "multiValueHeaders": {} // HTTP请求头，key:value，value 为字符串数组，用于支持存在相同请求头的情形
  "queryStringParameters": {}, // HTTP请求的Query，键值对形式
  "requestContext": {}, // 云开发相关信息
  "body": "HTTP请求体", // 普通文本 或 Base64 编码，如果是 isBase64Encoded=true，则是 Base64 编码
  "isBase64Encoded": false // true or false，表示 body 是否为 Base64 编码，如果是 Base64 编码，业务需要解码后获得元数据。如果 HTTP 请求包体是 二进制数据，则包体内容会被处理成 Base64编码的字符串，此时该值为 true。
}
```

下面是该结构的一个示例：

```
{
  "path": "/",
  "httpMethod": "GET",
```

```
  "headers": {
    "host": "env-id.service.tcloudbase.com",
    "connection": "close",
    "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36"
  },
  "multiValueHeaders": {
    "host": ["env-id.service.tcloudbase.com"],
    "connection": ["close"],
    "user-agent": ["Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"]
  },
  "queryStringParameters": {},
  "isBase64Encoded": false,
  "body": ""
}
```

云函数的返回值

云函数可以返回 `number` 、 `string` 、 `object` 等类型的数据，或者返回 集成响应，随后返回值会被转化为正常的 HTTP 响应。
集成响应，就是通过 JSON 格式数据描述 HTTP 响应报文的一段内容。

返回字符串或数字

云函数返回**字符串或数字**，字符串将会被放在 HTTP 响应 Body 中返回，同时 HTTP 响应的 `content-type` 会被设置为 `text/plain` 。

```
module.exports.main = async function () {
  return "hello gateway"
}
```

最终 HTTP 响应为：

```
HTTP/1.1 200 OK
date: Mon, 16 Dec 2019 08:35:31 GMT
content-type: text/plain; charset=utf-8
content-length: 13

hello gateway
```

返回 Object

云函数返回的 JS 对象会被转换为 JSON 字符串并在 HTTP 响应 Body 中返回，同时 HTTP 响应的 `content-type` 会被设置为 `application/json` 。

```
module.exports.main = async function () {
  return {
    foo: "bar"
```

```
  }
}
```

最终 HTTP 响应为：

```
HTTP/1.1 200 OK
date: Mon, 16 Dec 2019 08:35:31 GMT
content-type: application/json; charset=utf-8
content-length: 13

{"foo":"bar"}
```

返回集成响应

云函数可以返回如下这样特殊结构的**集成响应**，来自由地控制响应体。

```
{
  "statusCode": 200, // HTTP状态码
  "headers": { "headerName": "headerValue", ... }, // HTTP 响应头
  "body": "...", // HTTP 响应体，如果需返回 二进制数据，可以将二进制数据编码成 Base64字符串，并将 isBase64Encoded 设置为 true
  "isBase64Encoded": false, // HTTP 响应体是否是 Base64 编码
}
```

使用集成响应返回 HTML

将 `content-type` 设置为 `text/html`，即可在 `body` 中返回 HTML，会被浏览器自动解析：

```
module.exports.main = async function () {
  return {
    statusCode: 200,
    headers: {
      "content-type": "text/html"
    },
    body: "<h1>Hello</h1>"
  }
}
```

最终 HTTP 响应为：

```
HTTP/1.1 200 OK
date: Mon, 16 Dec 2019 08:35:31 GMT
content-type: text/html; charset=utf-8
content-length: 14

<h1>Hello</h1>
```

使用集成响应返回 JS 文件

将 `content-type` 设置为 `application/javascript`，即可在 `body` 中返回 JavaScript 文件：

```
module.exports.main = async function () {
  return {
    statusCode: 200,
    headers: {
      "content-type": "application/javascript"
    },
    body: 'console.log("Hello!")'
  }
}
```

最终 HTTP 响应为：

```
HTTP/1.1 200 OK
date: Mon, 16 Dec 2019 08:35:31 GMT
content-type: application/javascript; charset=utf-8
content-length: 21

console.log("Hello!")
```

使用集成响应返回二进制文件

如果返回体是诸如图片、音视频这样的二进制文件，那么可以将 `isBase64Encoded` 设置为 `true`，并且将二进制文件内容转为 Base64 编码的字符串，例如：

```
module.exports.main = async function () {
  return {
    isBase64Encoded: true,
    statusCode: 200,
    headers: {
      "content-type": "image/png"
    },
    body: "iVBORw0KGgoAAAANSUhEUgAAAMgAAADICAY..."
  }
}
```

最终 HTTP 响应为一张 PNG 图片：

```
HTTP/1.1 200 OK
date: Mon, 16 Dec 2019 08:35:31 GMT
content-type: image/png
content-length: 9897

<binary payload...>
```

安全规则

概述

最近更新时间：2024-11-14 10:52:41

简介

安全规则是在基础权限控制基础上，更高级、灵活、可扩展的一种权限控制方式，可用于保护云数据以及云存储中的数据。
基础权限控制是根据开发过程中最基础常见的情况预置的权限组，可以实现简单的权限管控，但随着应用业务场景越来越复杂，原有的管控方式无法支持类似根据角色、或者登录模式进行管控的需求。基础权限控制下实现类似需求需要将所有的客户端操作收归到云函数中，通过云函数中运行校验判断逻辑，实现访问控制，但在安全规则下，开发者可以自定义管控条件，通过规则语句完成权限配置，系统会根据规则完成访问控制，甚至可以进行数据校验。
安全规则利用基于 JSON 格式可扩展的配置语言定义用户可访问数据库与存储中的哪些数据开发者可根据应用所需的细化级别，编写简单或复杂的规则来保护应用中的数据。

优势特性

- **灵活**。开发者可根据自己的业务特性自定义规则，规则支持根据应用数据本身来限制数据的可访问性。
- **细化粒度**。规则可根据需求设置，或简单或复杂。
- **安全独立**。规则在管理端进行设置，独立于应用业务逻辑之外，客户端不会获取规则，也无法绕过规则验证，保证对应用数据的访问都经过校验。
- **便捷**。通过配置规则即可完成权限控制，免除接入层管控的开发与部署，可以快速验证。

工作方式

安全规则通过如下方式工作：在访问数据时，系统读取管理端设置的完整规则配置 JSON，其key为操作类型。系　根据操作类型获取配置值，值可能为 boolean 或 表达式字符串，表达式字符串为类 Javascript 语言的逻辑表达式。当 boolean 或 表达式计算结果为 true 时则表示通过允许访问，否则拒绝访问。只有允许访问时才会进行对应的后续操作。基础的规则配置格式如下：

```
{
    // 当配置值为 true 或 条件表达式计算值为 true 时允许对数据的访问。
    "read": false,
    "write": <<condition>>
}
```

其中表达式中可使用相应的内置变量获取请求的身份及元数据信息来进行复杂逻辑判断。更多详情请参见 安全规则语言 。

计费

安全规则本身不收取任何费用，但规则中使用元数据变量（doc，resource）或获取函数（get）将对对应资源产生读取，消耗资源用量。

限制

- 安全规则配置中每个表达式（组）最大长度限制为 1024 字符。
- 表达式支持变量（不支持正则）、数组表达式（数组成员限制为数字或字符串）、逻辑表达式（&&、||）、一元表达式（+1，-1，!a）、二元表达式（ >、>=、<、<=、==、!=、in），成员表达式（a.b、a["b"]）、调用表达式（仅云数据库安全规则支持），字符串模板表达式（用户云数据库函数调用动态参数，例：`database.collection.${doc.objectId}` ）。

使用入门

最近更新时间：2023-06-12 16:00:57

安全规则为云数据库与云存储中的数据提供强大的、完全可自定义的保护。您可以按照本指南中的步骤轻松开始使用规则，确保数据安全，并保护您的应用免受恶意用户攻击。

了解安全规则语言

安全规则语言使用一种 JSON 结构，key 指代操作类型，value 为允许操作时的条件，可以为 boolean 或表达式字符串，表达式字符串语法类似 Javascript 语言，其是单个逻辑表达式，或多个逻辑表达式通过与/或方式组合，当表达式的计算值决定了操作是否被允许。

步骤1：定义规则和数据结构

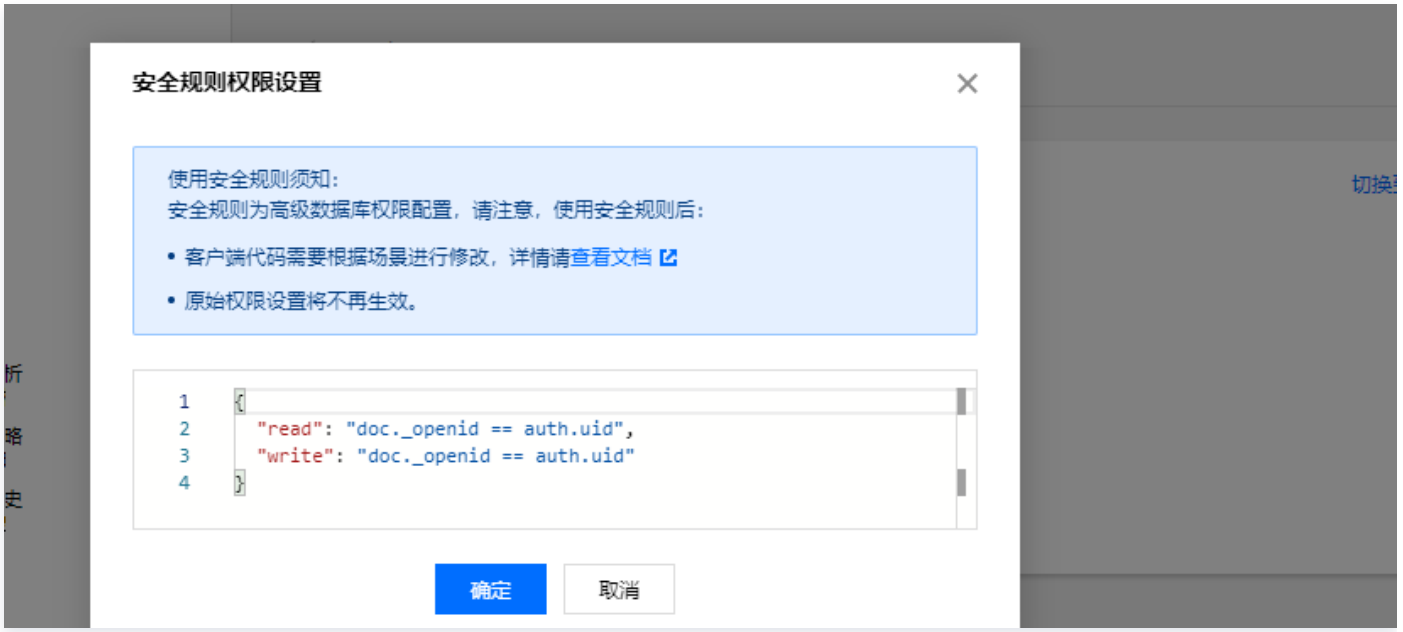
安全规则提供了内部变量获取请求数据的内容以根据数据进行访问控制，数据的组织方式可能会影响规则的写法，例如，论坛应用中需要限制所有登录用户可以浏览内容，用户只能修改、删除自己的帖子，那么论坛帖集合（posts）中记录需要存储 userID 表示记录归属，安全规则根据 userID 字段限制 update 与 delete 行为，放过所有的登录用户的 read 行为。

步骤2：获取安全规则

使用/查看现有安全规则可登录 云开发控制台，选定需要的环境，进入云数据库集合权限控制页或文件管理权限控制页，可以选择使用高级权限控制或查看已经在使用的安全规则。

1. 以云数据库集合权限管理为例，进入控制台，选择对应环境，单击左侧菜单栏**数据库**查看环境下 云数据库集合 的详情信息。
2. 单击集合名称，在详情中切换至权限设置页面，可由基础权限控制选择切换为安全规则。

3. 单击**切换到安全规则**，系统会根据已有的基础权限转化为对应的安全规则信息，开发者也可在其基础上进行修改。



4. 已经设置成功，正在使用的安全规则后续可以在权限设置中进行查看及修改。

步骤3：编辑发布安全规则

在云开发控制台权限控制页中可编辑所需的安全规则，例如上述论坛应用的例子，可改 posts 集合的权限为以下规则：

```
{
  "read": "auth != null", // 所有登录用户可读
  "create": "auth != null", // 所有登录用户可以发帖
  "update": "doc.userID == auth.openid", // 用户只能修改自己的帖子
  "delete": "doc.userID == auth.openid" // 用户只能删除自己的帖子
}
```

通过保存按　使编辑的新规则部署云端，保存成功后安全规则即时生效。

了解安全规则

最近更新时间：2023-06-12 16:01:38

安全规则语言

安全规则基于灵活、强大的自定义语言，可以支持各种复杂性和细化程度。我们可以按照自己应用的具体需要，设置特定的规则或者一般性的规则。安全规则语言使用一种 JSON 结构，key 指代操作类型，value 为允许操作时的条件，可以为 boolean 或表达式字符串，表达式字符串语法类似 Javascript 语言，其是单个逻辑表达式，或多个逻辑表达式通过与/或方式组合，当表达式的计算值决定了操作是否被允许。
由于安全规则是新的自定义语言，请阅读本指南，在您深入了解更复杂的规则时，可以更好地了解此规则语言。

基本结构

云数据库

安全规则基于 JSON 结构，value 为 boolean 或类似 JavaScript 的表达式：

```
{
    "read": true,
    "write": <<condition>>,
    "create": true,
    "update": false,
    "delete": <<condition>>
}
```

其中包含几个关键概念：

- **操作类型**： JSON key 为支持权限控制的操作类型，create、delete、read、update为集合的增删查改操作，write 为简便的控制集合的写权限配置，若没有配置 create、delete、update 操作，则按照 write 配置处理。
- **规则**： JSON value 为规则条件，可以为 boolean 值或逻辑表达式（组），当条件值为 true 时表示允许进行操作。

默认情况下，遇到没有定义的操作类型，则拒绝该操作访问。

云存储

安全规则基于 JSON 结构，value 为 boolean 或类似 JavaScript 的表达式：

```
{
    "read": true,
    "write": <<condition>>,
}
```

其中包含几个关键概念：

- **操作类型**： JSON key 为支持权限控制的操作类型，read、write为云存储文件的读写操作。
- **条件**： JSON value 为规则条件，可以为 boolean 值或逻辑表达式（组），当条件值为 true 时表示允许进行操作。

默认情况下，遇到没有定义的操作类型，则拒绝该操作访问。

条件构造

条件可以为 boolean 或表达式字符串，表达式字符串语法类似 Javascript 语言，其是单个逻辑表达式，或多个逻辑表达式通过与/或方式组合，当表达式的计算值决定了操作是否被允许。

云数据库

预置变量

变量	类型	说明
now	number	当前时间，以从 Linux 计时原点开始计算的毫秒数表示。
auth	Auth	在用户登录后，提供 uid（用户的唯一 ID）和 login
doc	Object	代表请求数据时当前记录的对象，在条件内引用 doc 将导致最多一次从服务中读取该值。此查询将计入资源的所有与服务相关的配额。
request	Request	代表请求对象，可以获取请求相关的值

Auth

字段名	类型	说明
loginType	string	登录方式 公众平台登录，开放平台登录，自定义登录，匿名登录等。
uid	string	用户唯一 ID，微信小程序的请求没有此值。
openid	string	用户 openid，仅在微信登录方式下存在值。

Request

字段名	类型	说明
data	object	请求数据，操作类型为 create，update 时存在，代表请求是传入 data 对象

变量可用于规则表达式中，通过 doc 与 request.data 可以获取数据当前的值与请求传入的值，例如在订单集合中，保护订单的金额值不被篡改，可使用规则：

```
{
    // ... //
    "update": "doc.price == request.data.price || request.data.price == undefined"
    // ... //
}
```

内置方法

目前仅支持 get 函数，唯一的参数必须为 `database.集合名称.文档id` 。通过访问其它文档的数据来判断用户操作是否符合安全规则。

云函数

预置变量

变量	类型	说明
now	number	当前时间，以从 Linux 计时原点开始计算的毫秒数表示。
auth	Auth	在用户登录后，提供 uid（用户的唯一 ID）和 loginType（登录类型）。如果用户未登录，则为 null。
resource	Resource	代表请求文件元数据，在条件内引用 resource 将导致最多一次从服务中读取该值。

Auth

字段名	类型	说明
loginType	string	登录方式 公众平台登录，开放平台登录，自定义登录，匿名登录等。
uid	string	用户唯一 ID，微信小程序的请求没有此值。
openid	string	用户 openid，仅在微信登录方式下存在值。

Resource

字段名	类型	说明
openid	string	文件私有归属标识，标记所有者 ID

变量可用于规则表达式中，例如限制用户写入的数据归属于自己，并且只有自己可修改。

```
{
  {
    "write": "resource.openid == auth.uid"
}
```

运算符

运算符	说明	示例	示例解释(集合查询)
==	等于	auth.uid == 'zzz'	用户的 uid 为 zzz
!=	不等于	auth.uid != 'zzz'	用户的 uid 不为 zzz
>	大于	doc.age > 10	查询条件的 age 属性大于 10
>=	大于等于	doc.age >= 10	查询条件的 age 属性大于等于 10
<	小于	doc.age < 10	查询条件的 age 属性小于 10
<=	小于等于	doc.age <= 10	查询条件的 age 属性小于等于 10
in	存在于集合中	auth.uid in ['zzz','aaa']	用户的 uid 是['zzz','aaa']中的一个

!(xx in [])	不存在于集合中，使用 in 的方式描述 !(a in [1,2,3])	!(auth.uid in ['zzz','aaa'])	用户的 uid 不是['zzz','aaa']中的任何一个
&&	与	auth.uid == 'zzz' && doc.age>10	用户的 uid 为 zzz 并且查询条件的 age 属性大于 10
||	或者	auth.uid == 'zzz' || doc.age>10	用户的 uid 为 zzz 或者查询条件的 age 属性大于 10
.	对象元素访问符	auth.uid	用户的 uid
[]	数组访问符属性	get('database.collection_a.user')[auth.uid] == 'zzz'	collection_a 集合中 id 为 user 的文档，key 为用户 uid 的属性值为 zzz

编写安全规则

最近更新时间：2023-09-13 16:55:06

安全规则允许您控制数据的访问权限。通过灵活的规则语法可以在集合或存储桶上实现不同粒度的读写控制组合。

默认规则

默认情况下，安全规则拒绝所有的数据访问。

```
{}
```

用户身份认证

经过身份认证的用户发起请求时，系统会使用用户唯一 ID `uid` 及用户登录方式 loginType 填充 auth 变量。当未经身份验证的用户发出请求时，auth 变量值为 null。

> **说明**
> 通过 auth 变量，可以用以下常用方式来根据身份对文件访问进行控制：
> - 公开：不判断 auth 值。
> - 只对已登录用户公开：检查 auth 不为 null。
> - 用户私有：检查 auth.uid 是否等于资源 openid。
> - 仅对某种特殊的登录方式进行判断，限制匿名登录用户访问，检查 auth.loginType 不为 ANONYMOUS。

公开

任何不考虑 auth 的规则均可被视为 public 规则，因为他不考虑用户的身份验证上下文，这些规则在呈现公开数据（静态资源内容）的场景下是很适用。

云数据库

```
{
  "read": "doc._openid != null"
}
```

云存储

```
{
  "read": "resource.openid != null"
}
```

对登录的用户开放

在某些情况下，可能希望限制只有登录用户才可以访问用户数据。例如，登录用户才可以查看论坛中的讨论。由于所有未登录用户的 auth 变量为 null，因此可以设置如下规则：

```
{
```

```
  "read": "auth != null"
}
```

所有用户可读，仅创建者及管理员可写

云数据库

数据通过 _openid 记录当前数据的归属用户 ID。

```
{
  "read": true,
  "write": "doc._openid == auth.openid", // 登录方式为微信
  "write": "doc._openid == auth.uid" // 登录方式为非微信
}
```

云存储

```
{
  "read": true,
  "write": "resource.openid == auth.openid", // 登录方式为微信
  "write": "resource.openid == auth.uid" // 登录方式为非微信
}
```

仅创建者及管理员可读写

云数据库

数据通过 _openid 记录当前数据的归属用户 ID。

```
{
  "read": "doc._openid == auth.openid", //登录方式为微信
  "read": "doc._openid == auth.uid", // 登录方式为非微信
  "write": "doc._openid == auth.openid", //登录方式为微信
  "write": "doc._openid == auth.uid" // 登录方式为非微信
}
```

云存储

```
{
  "read": "resource.openid == auth.openid", //登录方式为微信
  "read": "resource.openid == auth.uid", // 登录方式为非微信
  "write": "resource.openid == auth.openid", //登录方式为微信
  "write": "resource.openid == auth.uid" // 登录方式为非微信
}
```

所有用户可读，仅管理员可写

```
{
  "read": true,
  "write": false
}
```

仅管理员可读写

```
{
  "read": false,
  "write": false
}
```

数据验证

您可以根据数据库或存储分区中的现有数据，使用安全规则有条件地写入新数据。您还可以编写用来强制执行数据验证的规则，方法是根据写入的新数据来限制编写操作。

对新数据的限制

云数据库

如果要确保包含特定字段的文档尚未创建。例如，如果要拒绝创建包含 ranking 字段的所有文档，您可以在 create 条件中禁止它。

```
{
    // ... //
    "create": "request.data.ranking == undefined"
}
```

云存储

上传的文件都必须有归属值，且归属当前用户。

```
{
    // ... //
    "write": "resource.openid == auth.openid"
}
```

使用现有数据

许多应用都将访问权限控制信息以字段形式存储在数据库中的文档内。安全规则可以根据文档数据属性控制访问：
要实现此权限控制，需要在用户数据中定义对应的属性。安全规则根据数据属性检查请求，校验允许或拒绝。例如，我们在存储成绩时，则可以向不同的用户组分配不同的访问权限级别：向“学生”组分配内容只读权限，向“教师”组分配教师所教科目的读写权限.
用户表(user 集合)则可以设计如下结构：

```
{
    userID: string, // 用户唯一id
    role: string, // STUDENT, TEACHER,
```

```
    projects: string[] // 教师所教科目
}
```

成绩表（score）设计如下：

```
{
    studentID: string,
    project: string,
    score: number
}
```

成绩表规则可设置为：

```
{
  "read": "get(`database.user.${auth.uid}`).role == 'STUDENT' ||
(get(`database.user.${auth.uid}`).role == 'TEACHER' && doc.project in
get(`database.user.${auth.uid}`).projects)",
  "write": "get(`database.user.${auth.uid}`).role == 'TEACHER' && doc.project in
get(`database.user.${auth.uid}`).projects"
}
```

系统函数

最近更新时间：2023-09-13 16:55:06

安全规则内置一些系统级函数，系统函数提供强大的通用能力完成一些操作，开发者可以在规则表达式中直接调用实现对应的功能，以便更灵活地控制资源的访问权限。

get

```
static

get(path) returns document 对象
```

根据参数获取指定 doc 内容。

输入参数

参数	类型	说明
path	string	非空，格式为 `database.集合名称.文档id` 的字符串，值可以通过多种计算方式得到，例如使用字符串模板进行拼接（`database.${doc.collction}.${doc._id}`）

返回值

undefined or null 表征 doc 不存在，否则为 document 对象，表征查询得到的数据。

示例

1. 用户的权限是写在一个独立的文档，用一个数值表示用户的权限范围：

```
{
  "read": "get('database.test.123')[auth.uid] in [1,2,3]",
  "delete": "get('xxxx')[auth.uid] == 1 && doc.user in ['ersed','sfsdf'] "
}
```

2. 集合 A 包含 shopId、orderId 关联关系，集合 B 包含 owner，shopId 关联关系，对集合 A 查询，希望限制只查到当前用户有权限的 shop 的订单。

```
{
  "read:": "auth.openid in get(`database.B.${doc.shopId}`).owner"
}
```

限制

- get 参数中存在的变量 doc 需要在 query 条件中以 == 或 in 方式出现，若以 in 方式出现，只允许 in 唯一值， 即 doc.shopId in array, array.length == 1
- 一个表达式最多可以有 3 个 get 函数，最多可以访问 10 个不同的文档。
- get 函数的嵌套深度最多为 2， 即 get（get（path））。

说明

在未使用变量的情况下，每个 get 会产生一次读操作，在使用变量时，每个 get，每个变量值会产生一次读操作。
例如：规则 `get(&#180;database.collection.${doc._id&#180;}).test` ，在查询 `_.or([{_id:1},{_id:2},{_id:3},{_id:4},{_id:5}])` 下会产生 5 次读取。系统会对同 doc，同 field 的读取进行缓存（在尽可能的情况下，安全规则会合并部分同 doc 不同 field 的读取，在一次读取中获取多个 field，以减少对数据库资源的消耗，合并程度与规则的复杂程度与写法有关）。

日志管理

概述

最近更新时间：2023-09-13 16:55:06

云开发为开发者提供了更高级的日志检索及管理能力，开发者可以使用云开发提供的 SDK 自定义 打印日志 并可用多种方式检索日志，例如 全文检索 、键值检索 、模糊关键字检索 以及支持多种 查询语法 。

功能概览

打印日志

相比原有的 console 对象打印日志，开发者可使用云开发自定义日志，输出同样提供四个日志等级的日志，例如 `log` 、 `info` 、 `warn` 、 `error` 。开发者自定义打印的日志内容会自动增加日志中的字段并建立键值索引，详情请参见 打印日志 。

检索日志

相比原有单一的日志查看方式，云开发为开发者提供多种检索日志内容的方式，既可以精确检索也可以模糊匹配关键词，提供全文内容的关键词检索以及 key:value 形式的键值对检索，并支持多种查询语法，详情请参见 检索日志 。

更多说明

原始日志文本将根据**分词符**分成多个关键词，以及使用**大小写敏感**来精确定位到用户输入的日志信息。

分词符

原始日志文本将根据分词符切分成多个关键词，云开发默认分词符为 `!@#%^&*()_="', <>/?|\;:\n\t\r[]{}` ，以下面某条日志内容为例：

```
10002345987;write;ERROR;code=400;topic does not exist;
```

上述日志内容中 `;` ， `=` 两个默认分词符会将日志内容切分成6个单词 `10002345987` ， `write` ， `ERROR` ， `code` ， `400` ， `topic does not exist` 。在精准搜索中，用户只需输入任意一个单词，即可搜索到此条日志内容。

> **说明**
> - 因 日志默认系统字段 中 requestid 包含 - ，故 - 不作为默认分词符。
> - 用户日志根据分词符进行分词之后的日志内容，单个分词后的内容不能超过32KB。

大小写敏感

云开发日志检索默认**大小写敏感**，例如在上述日志内容例子中，日志内容已被切分成6个单词 `10002345987` ， `write` ， `ERROR` ， `code` ， `400` ， `topic does not exist` 。若需要精确搜索到 ERROR 关键词的相关日志，需在控制台输入 `ERROR` 。

> **注意**
> 若在控制台输入 `error` 或 `Error` ，则无法检索到 ERROR 的相关日志。

打印日志

最近更新时间：2023-09-13 16:55:06

操作场景

用户可以使用原有日志打印方式（console 对象打印日志），也可以使用 tcb-admin-node 封装的自定义打印日志。

- 使用 console 打印日志：可使用日志等级 `log` 、 `info` 、 `warn` 、 `error` 不会自动建立 键值索引，其中日志内容会封装至日志内容中 msg 字段的值。
- 自定义打印日志：需要导入 tcb-admin-node，可使用 `log` 、 `info` 、 `warn` 、 `error` ，日志内容会自动增加日志字段并建立 键值索引。

导入 SDK

```
const admin = require('tcb-admin-node')
```

> **注意**
> 因导入的 SDK 为 nodejs 语言的 SDK，故下述的操作示例均为 nodejs 语言。

操作用法

log

用例 `log`：

```
admin.logger().log({content: "this is a log"})
```

日志打印：

```
{
    "level": "log",
    "timestamp": "1565864885000002",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "content": "this is a log"
}
```

```
console.log("this is a log")
```

日志打印：

```
{
    "level": "log",
    "timestamp": "1565864885000002",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "msg": "this is a log"
}
```

> **说明**
> SDK 方式需要用户输入 object 类型参数。

info

用例 `info`:

```
admin.logger().info({content: "this is an info"})
```

日志打印:

```
{
    "level": "info",
    "timestamp": "1565864885000003",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "content": "this is an info"
}
```

```
console.info("this is an info")
```

日志打印:

```
{
   "level": "info",
   "timestamp": "1565864885000002",
   "function": "functionName",
   "requestid": "123345-123123-213123123-444",
   "src": "app",
   "msg": "this is an info"
}
```

> **说明**
> SDK 方式需要用户输入 object 类型参数。

warn

用例 `warn`:

```
admin.logger().warn({content: "this is a warn"})
```

日志打印:

```
{
    "level": "warn",
    "timestamp": "1565864885000004",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "content": "this is a warn"
}
```

```
console.warn("this is a warn")
```

日志打印：

```
{
    "level": "warn",
    "timestamp": "1565864885000002",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "msg": "this is a warn"
}
```

> **说明**
> SDK 方式需要用户输入 object 类型参数。

error

用例 error：

```
admin.logger().error({content: "this is an error"})
```

日志打印：

```
{
    "level": "error",
    "timestamp": "1565864885000005",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "content": "this is an error"
}
```

```
console.error("this is an error")
```

日志打印：

```
{
    "level": "error",
    "timestamp": "1565864885000002",
    "function": "functionName",
    "requestid": "123345-123123-213123123-444",
    "src": "app",
    "msg": "this is an error"
}
```

> **说明**
> SDK 方式需要用户输入 object 类型参数。

日志格式

日志打印后的格式会自带系统默认的字段，其中默认的系统的字段如下：

字段	类型	默认	说明
level	string	是	（log/info/warn/error）
timestamp	string	是	打印日志的时间戳，精度到微秒
function	string	是	当次调用的函数名称
requestId	string	是	当次请求的 ID
src	string	是	system/app
msg	string	否	简单日志内容

注意

- 用户自定义打印了日志对象内容则会增加自定义的日志字段，并建立 键值索引 。
- 用户自定义日志字段若出现系统默认字段， 检索日志 时则会优先自定义日志内容。
- 限制用户自定义日志字段不能出现以下关键字： `"\_\_FILENAME\_\_"` ，`"\_\_TIMESTAMP\_\_","\_\_LOGSETID\_\_","\_\_TOPICID\_\_"` 。

键值索引

云开发日志会默认创建键值索引，键值索引可使用于**键值检索**中，其中默认创建的键值索引包括**上述日志默认系统参数索引**，以及**用户自定义的日志对象属性键值索引**，如下：

```
var array = ["tcb", 123434, true, false, "hello tcb", null, undefined, (new
Date).getTime(), true && false, 0 > 1 ? "0<1" : "1大于0"];

var logShort = {
    name: "tcb",
    timestamp: (new Date).getTime(),
    randnumber: Math.random(),
    intlog: 1236753171,
    stringlog: "testlog",
    floatlog: 0.5234562,
    arraylog: array,
    booleanlog: true,
    operationlog: 0 > 1 ? "0<1" : "1大于0",
    nulllog: null,
};

admin.logger().log(logShort)
```

日志打印如下：

```
{
    "level": "log",
```

```
    "timestamp": "1568281919939",
    "function": "PqWC-i16eK",
    "requestId": "f51346ea-d542-11e9-9950-525400edfec1",
    "src": "app",
    "name": "tcb",
    "randnumber": "0.4922406878066756",
    "intlog": "1236753171",
    "stringlog": "testlog",
    "floatlog": "0.5234562",
    "arraylog": "["tcb",\n 123434,\n true,\n false,\n "hello tcb",\n null,\n null,\n
1568281919939,\n false,\n "1大于0"]",
    "booleanlog": "true",
    "operationlog": "1大于0",
    "nulllog": "null"
}
```

则默认可以用以上输出的键值索引，例如： `level` 、`timestamp` 、`function` 、`requestid` 、`src` 、`name` ，`randnumber` 、`intlog` 、`stringlog` 、`floatlog` 、`arrarylog` 、`booleanlog` 、`operationlog` ，`nulllog` 。

检索日志

最近更新时间：2024-08-13 16:08:42

操作场景

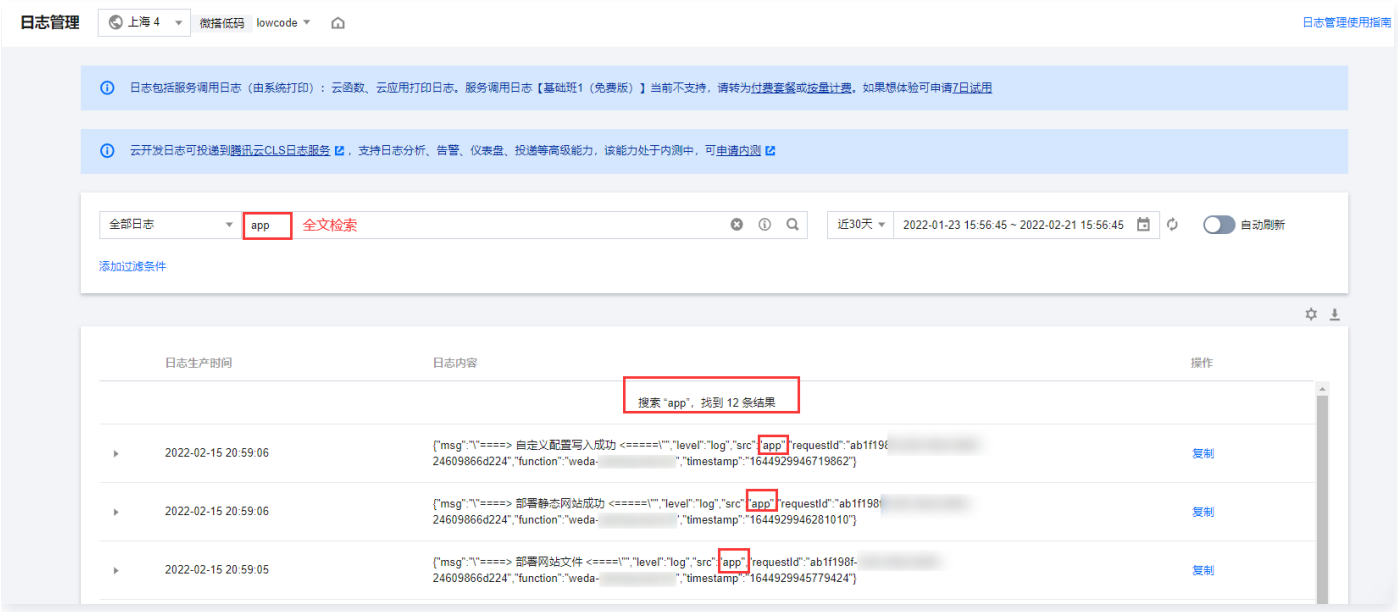
本文档主要指导您如何在云开发控制台进行日志检索。

操作步骤

1. 登录云开发控制台 - 日志检索 。
2. 进入日志检索页面，单击**点击开通**，开通日志检索功能。
3. 选择检索的时间范围，然后在输入框填写检索语法（支持全文检索、模糊关键字检索、键值检索）。

全文检索

日志内容会根据分词符拆分为多个词组，用户可以输入特定的关键词精确检索到相关日志，也支持模糊关键字匹配检索。



键值检索

日志内容以 JSON 对象格式返回，类似于键值对即 `key:value` 形式，其中 key 作为用户可以自定义输出日志内容对象属性字段，value 则为日志内容，在控制台中同样支持 `key:value` 格式定位日志，value 支持模糊关键字检索。模糊关键字检索如下：

模糊关键字检索

日志服务提供模糊查询的能力，通过特殊的模糊关键字进行日志检索，具体说明如下：

元字符	描述
*	模糊查询关键字，匹配零个、单个或多个任意字符。不支持开头 `*`，例如，输入 `abc*`，会返回以 `abc` 开头的所有命中日志
?	模糊查询关键字，特定位置匹配单个字符。例如，输入 `ab?c`，会返回以 `ab` 为开头，以 `c` 为结尾字符，且两者之间且有一个字符的所有命中日志

查询语法

检索支持以下查询语法：

语法	语义
key:value	键值搜索格式，其中 value 支持 `?` 、 `*` 模糊搜索
A and B	“与”逻辑，返回 A 与 B 的交集结果，若多个关键词，使用空格分隔则默认为 and
A or B	“或”逻辑，返回 A 或 B 的并集结果
not B	“非”逻辑，返回不包含 B 的结果
A not B	“减”逻辑，返回符合 A 但不符合 B 的结果，即 A - B
A:B	key-value 对的查询格式，若 key 或者 value 中包含空格、冒号等关键字符，需要用引号包括。例如 "error level":high
'a'	字符a将被视为普通字符，不会当作语法关键词处理
"A"	A 中的所有关键词都将被视为普通字符，不会当作语法关键词处理
\	转义字符，转义后的字符表示符号本身，例如转义引号 `\"` ，转义冒号 `\:` 等
*	模糊查询关键字，匹配零个、单个或多个任意字符，不支持开头 `*` ，例如：输入 `abc*` ，会返回以 `abc` 开头的所有命中日志
?	模糊查询关键字，特定位置匹配单个字符。例如，输入 `ab?c` ，会返回以 `ab` 为开头，以 `c` 为结尾字符，且两者之间且有一个字符的所有命中日志

说明

- 运算符的优先级由高到低排序为 `:` > `"` > `and` > `not` > `or` 。
- 若 b 是文本， `a=b` 与 `a:b` 的区别在于前者是 a 全等于 b，后者是 a 包含 b（按分词逻辑处理，支持模糊搜索）。
- 语法关键词不区分大小写。

监控告警

概述

最近更新时间：2023-09-13 16:55:07

当您需要云开发的监控告警能力时，可在环境中监控统计页查看到具体的监控告警能力。您可查看到当前环境云函数、数据库、云存储等不同资源的监控指标。可查看自己业务的使用情况，并观察业务是否有异常存在。

> **说明**
> 在某些指标改变时，云开发还支持创建告警来及时通知您采取措施。告警在一定周期内监控某些特定指标，并根据给定的阈值，每隔若干个时间段发送告警。

告警包含以下几个组成部分：

- 告警触发条件（什么条件下发送告警）：具体触发告警的规则，每个告警策略中可以创建多个触发条件。
- 告警对象（哪个对象发出告警）：被告警的资源的对象，目前支持的对象为函数。
- 告警渠道（谁通过什么方式收到告警）：用户创建告警的最小单元，用户可以创建告警策略绑定在需要告警的对象上。一个告警策略可以绑定到多个对象。

管理告警策略

最近更新时间：2024-09-03 10:41:22

操作场景

当用户需要针对云开发某个资源对象的某个状态发送告警时，需要先创建告警策略。本文档主要指导您如何创建和查看告警策略。

操作步骤

创建告警策略

1. 登录 云开发控制台，单击左侧菜单**环境**。
2. 单击您需要创建告警策略的环境名称，进入环境页面，单击左侧栏**监控统计 > 告警策略**。
3. 进入告警策略页面，单击**新建**。
4. 进入新建策略页面，配置说明如下：
   - **策略名称**：填写策略名称，1 - 20个字符。
   - **备注**：填写策略备注，0 - 100个字符。
   - **产品类型**：所需告警的云开发产品资源类型，目前仅支持函数。
   - **配置告警对象**：
     - 选中全部对象，则该告警策略绑定当前环境的全部资源。
     - 选中选择部分对象，则该告警策略绑定用户选中的资源。
   - **设置告警触发条件**：
     告警触发条件是指标、比较关系、阈值、统计周期和持续周期组成的一个有语义的条件。
     - 告警指标：目前仅支持云函数错误次数和云函数运行时间。
     - 统计周期：目前的统计周期仅支持5分钟统计一次。
     - 比较方式：目前提供的比较方式包括 >、>=、<、<=、= 以及 !=。
     - 持续周期：目前支持的持续周期包括持续1个周期、持续2个周期、持续3个周期、持续4个周期以及持续5个周期。
     - 告警频率：目前支持的告警频率包括每小时告警一次和每12个小时告警一次。

> **说明：**
> 例如指标为运行时间 、比较关系为 > 、阈值为500 、统计周期为5分钟 、持续周期为2个周期 ，告警频率为每小时告警一次。则表示：每5分钟收集一次运行时间数据，若函数的运行时间连续两次大于500毫秒则触发告警。告警每小时触发一次。

   - **配置告警渠道**：根据需求，配置告警接收对象、有效时段、接收渠道（邮件、短信、微信、站内信）。

> **说明：**

> 告警接收对象可以为用户，也可以为用户组，请前往 访问管理 创建用户组。

查看告警历史

当您的告警条件被触发后，您可以前往统一工作台**运维监控** > **告警历史**中查看到某个时间段发送的告警。

API 和 SDK 使用指引
云 API 使用指引

最近更新时间：2024-10-09 14:32:21

简介

云开发（Tencent CloudBase，TCB）是腾讯云提供的云原生一体化开发环境和工具平台，为开发者提供高可用、自动弹性扩缩的后端云服务，包含计算、存储、托管等 serverless 化能力，可用于云端一体化开发多种终端应用（小程序、公众号、Web 应用等），帮助开发者统一构建和管理后端服务和云资源，避免了应用开发过程中繁琐的服务器搭建及运维，开发者可以专注于业务逻辑的实现，开发门槛更低，效率更高。

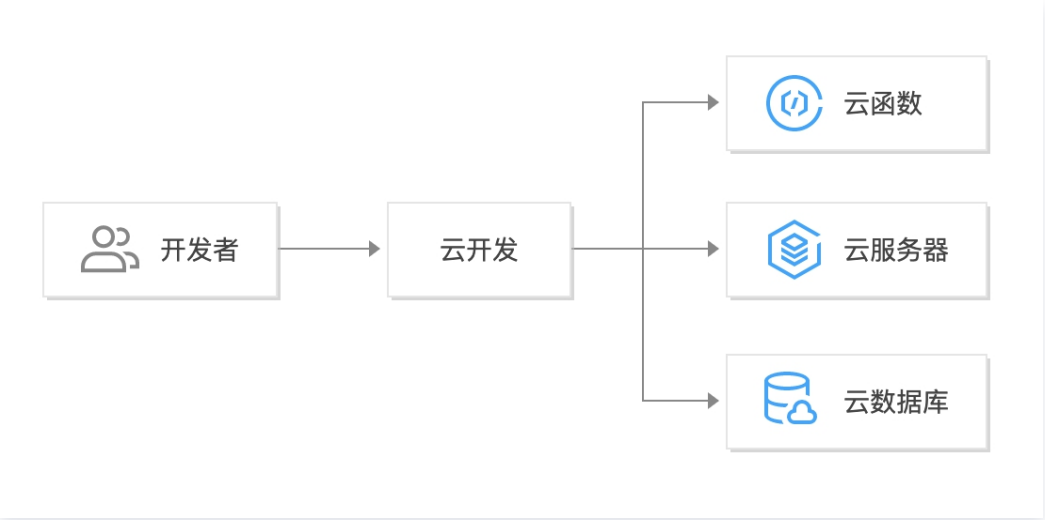
> **说明**
> 云开发的云 API 是云开发提供的管理端能力 API，开发者可使用云 API 自定义管理云开发资源，个性化搭建自有的控制台，或者在云 API 上二次封装更多能力对外开放，以满足更丰富的需求场景。更多云 API 规范请参见 腾讯云云 API 。

云 API 架构

云开发提供的一站式后端服务中，包含以下能力版块：

- 云函数，为云开发提供底层云函数管理及计算能力。
- 云数据库，为云开发提供底层数据库能力。
- 云存储，为云开发提供底层存储能力。

云开发对外提供服务，整体架构如下：



如需了解整个产品概况，单击 了解更多 。
针对云开发的云 API 包含两大部分：

- 由云开发产品提供的云 API 能力。
- 由云函数、云数据库、云存储提供的 API 能力。

以上两大部分均通过云开发 API 服务统一对外提供服务。

> **说明**
> - 针对云开发提供的 API，可参见 API 概览 。

- 针对云函数、云数据库、云存储提供的 API 能力，可通过以下云开发提供的 API Center 能力调用。

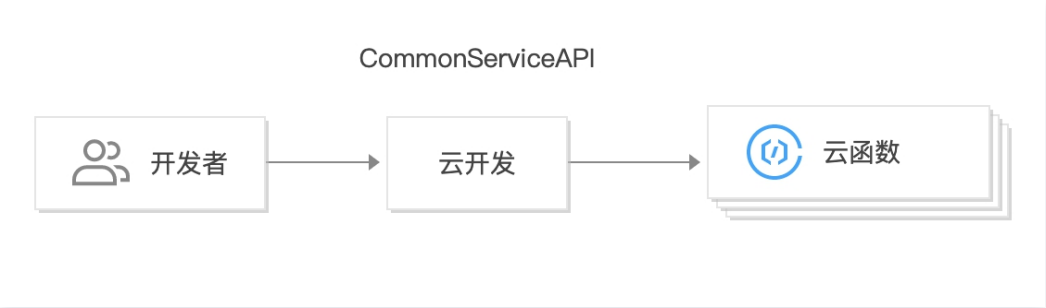
API Center 简介

主要功能

API Center 是云开发提供通过云开发 API 网管服务访问**云函数、云数据库、云存储**的 API 能力，主要有以下好处：

- API 请求入口统一化。
- 业务只需关注云开发即可。

调用情况如下图所示：



云开发提供一个通用的云 API 接口 **CommonServiceAPI**，可查看 文档说明，具体的使用方式如下：

- 云函数、云数据库、云存储**各自的 API 能力**。
- 通过 **CommonServiceAPI** 接口调用以上服务。

示例说明

以下使用云数据库的 DescribeRestoreTables 接口举例。

正常的调用方式

接口名称：**DescribeRestoreTables**。
输入参数：除公共参数外（可参见 公共请求参数 ），输入参数还包含：

参数	类型	说明
InstanceId	String	数据库实例 ID
Time	String	回档时间

请求示例：

```
{
    "Uin":"123456",
    "SubAccountUin":"12345678",
    "Version":"2018-06-08",
    "Timestamp":"1575446253",
    "Region":"ap-shanghai",
    "Action":"DescribeRestoreTables", // 调□接□
    "InstanceId": "tnt-123456789",     // 参数1
    "Time": "2019-12-01 00:00:00"     // 参数2
```

```
}
```

响应如下：

参数	类型	说明
Tables	List	可回档表格

```
{
    "RequestId":"4a39f1ce-e11b-4954-881d-561504e1ab4e",
    "Tables":[]
}
```

通过 CommonServiceAPI 的调用方式

1. 将原调　公共参数中的 Action，设置为 CommonServiceAPI 接口的 Service 参数。
2. 将原调　方 request 参数序列化为 json 的字符串，设置为 CommonServiceAPI 接口的 JSONData 参数。

接口名称：CommonServiceAPI
参数：除公共参数外（可参见 公共请求参数 ），输入参数还有：

参数	类型	说明
Service	String	转发云 API 的 Action 名
JSONData	String	转发云 API 参数，Json 格式字符串

请求示例：

```
{
    "Uin" :"123456",
    "SubAccountUin":"12345678",
    "Version":"2018-06-08",
    "Timestamp":"1575446253",
    "Region":"ap-shanghai",
    "Action":"CommonServiceAPI",
    "Service":"DescribeRestoreTables",
    "JSONData":"{
        \"InstanceId\": \"tnt-123456789\",
        \"Time\": \"2019-12-01 00:00:00\"
    }"
}
```

响应如下：

参数	类型	说明
JSONResp	String	云 API 响应，Json 格式

```
{
    "JSONResp": "{
        \"Response\":{
            \"RequestId\":\"4a39f1ce-e11b-4954-881d- 561504e1ab4e\",
            \"Tables\":[]
         }
     }",
     "RequestId": "427c6660-d389-4da4-aaa2-ca8c3626ba0e"
}
```