

AgentSphere

SDK 参考

文档版本	01
发布日期	2026-09-29



版权所有 © 华为云计算技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址：贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址：<https://www.huaweicloud.com/>

目 录

1 配置本地 E2B SDK 执行环境.....	1
2 沙箱实例管理.....	4
3 文件操作.....	9
4 执行命令.....	15
5 运行代码.....	18
6 进程管理.....	20

1 配置本地 E2B SDK 执行环境

AgentSphere提供了安全隔离的云上沙箱环境，支持代码执行、文件管理和命令运行等能力，广泛应用于AI Agent代码解释器、自动化测试、数据处理等场景。为了降低迁移成本，AgentSphere兼容了E2B SDK，您可以复用已有的E2B工作流和示例代码，仅需修改环境变量即可无缝切换到AgentSphere沙箱服务。

前提条件

- 已开通AgentSphere服务，并获取API Key。获取方式请参见[创建API Key](#)。
- 已创建沙箱模板，并获取模板ID。创建方式请参见[创建沙箱模板](#)。
- 已安装Python，且Python版本满足E2B SDK要求，版本要求详见[e2b](#)及[e2b-code-interpreter](#)。
- 已安装E2B SDK，安装命令如下：

```
pip install e2b==2.42.0
pip install e2b-code-interpreter==2.9.1
```

说明

建议使用e2b和e2b-code-interpreter指定版本。

- 给E2B SDK安装Patch文件，操作如下：

```
wget https://raw.githubusercontent.com/skyao/agentsphere-e2b-patch-python/main/dist/agentsphere_e2b_patch-0.1.0-py3-none-any.whl
pip install ./agentsphere_e2b_patch-0.1.0-py3-none-any.whl
```

配置环境变量

AgentSphere兼容E2B SDK，您需要配置以下环境变量，将SDK的请求指向AgentSphere服务：

环境变量	说明	获取方式
E2B_API_URL	AgentSphere管理面的访问地址，即API Server通过APIGW对外暴露的地址。	不同区域的访问地址不同，请通过 API访问地址 查询。

环境变量	说明	获取方式
E2B_SANDBOX_URL	AgentSphere数据面的访问地址，即模板所使用的Agent Gateway实例的公网域名。	在AgentSphere控制台中找到对应的“沙箱模板”，然后单击模板名称进入详情页，在“网络配置”中查看其入站网关的“公网域名”。
E2B_API_KEY	访问AgentSphere管理面的API Key，作为认证凭据。	在AgentSphere控制台的“API Key管理”页面创建并获取，详情请参见 创建API Key 。

⚠ 注意

- E2B_SANDBOX_URL必须与您创建沙箱实例时所使用的模板绑定的Agent Gateway域名一致，否则数据面请求将无法到达沙箱实例。
- 请妥善保管API Key，切勿硬编码在代码中或提交到版本控制系统，建议通过环境变量或密钥管理服务注入。

您可以通过以下三种方式设置环境变量：

方式一：dotenv 文件

在项目根目录下创建.env文件，配置环境变量：

```
E2B_API_URL=https://agentsphere.<region>.myhuaweicloud.com
E2B_SANDBOX_URL=https://<your-sandbox-url>.huaweicloud-agentnetwork.com
E2B_API_KEY=<your_api_key>
```

然后在Python代码中使用dotenv库加载环境变量，该方式下环境变量仅在当前代码运行范围内生效：

```
from dotenv import load_dotenv
load_dotenv()
```

方式二：终端环境变量

在执行python代码的主机中设置环境变量

```
export E2B_API_URL=https://agentsphere.<region>.myhuaweicloud.com
export E2B_SANDBOX_URL=https://<your-sandbox-url>.huaweicloud-agentnetwork.com
export E2B_API_KEY=<your_api_key>
```

您也可以将以上命令添加到终端配置文件（如 ~/.bashrc ）中，使其在所有终端会话中全局生效。

方式三：内嵌环境变量

在代码中通过os模块直接设置环境变量，该方式下环境变量仅在当前代码运行范围内生效：

```
import os
os.environ["E2B_API_URL"] = "https://agentsphere.<region>.myhuaweicloud.com"
```

```
os.environ["E2B_SANDBOX_URL"] = "https://<your-sandbox-url>.huaweicloud-agentnetwork.com"
os.environ["E2B_API_KEY"] = "<your_api_key>"
```

 说明

使用内嵌环境变量时，请确保在导入E2B模块之前完成环境变量设置，否则SDK可能在初始化时读取到空值。

API 访问地址

不同区域调用AgentSphere API的访问地址不同，如下表所示，请您根据业务需要选择对应区域的访问地址。

表 1-1 表 1 AgentSphere API 访问地址

区域名称	区域	API访问地址
华北-北京四	cn-north-4	- 智能体沙箱API: https://agentsphere.cn-north-4.myhuaweicloud.com - 智能体身份API: https://agent-identity.cn-north-4.myhuaweicloud.com
华南-广州	cn-south-1	- 智能体沙箱API: https://agentsphere.cn-south-1.myhuaweicloud.com - 智能体身份API: https://agent-identity.cn-south-1.myhuaweicloud.com
西南-贵阳一	cn-southwest-2	- 智能体沙箱API: https://agentsphere.cn-southwest-2.myhuaweicloud.com - 智能体身份API: https://agent-identity.cn-southwest-2.myhuaweicloud.com

相关文档

- [E2B SDK官方文档](#)：了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。

2 沙箱实例管理

沙箱实例是AgentSphere的核心运行单元。本节介绍如何创建、获取列表、获取信息、连接、暂停和删除沙箱实例。

前提条件

以下示例均基于E2B SDK，请确保已完成[配置本地E2B SDK执行环境](#)。

沙箱实例管理

创建沙箱实例

您可以使用*Sandbox.create*方法创建沙箱实例，创建时需指定模板名称，也可以指定超时时间、环境变量等参数。

```
from e2b import Sandbox

# template字段的值为已有的template_id
sandbox = Sandbox.create(template="<template_id>")

# 您可以在创建时指定超时时间，单位为秒
# 当沙箱的运行时间超过超时时间时，沙箱将会被自动删除
sandbox = Sandbox.create(
    template="<template_id>",
    api_key="<your_api_key>",
    envs={"MY_ENV": "value"},
    timeout=300
)
```

参数	说明
template	沙箱模板ID，指定沙箱实例的配置模板。
timeout	沙箱实例的超时时间，单位为秒。超时后沙箱实例将自动终止。
api_key	访问API Key。
envs	沙箱环境变量，以字典形式传入。

获取沙箱实例信息

您可以通过 `sandbox.get_info` 方法获取沙箱实例的详细信息，包括实例ID、模板ID、运行状态、启动时间、结束时间等。

```
from e2b import Sandbox

# 创建沙箱实例
sandbox = Sandbox.create(template="<template_id>")

# 获取沙箱实例信息
info = sandbox.get_info()
print(info)
```

获取沙箱实例指标数据

您可以通过 `sandbox._envd_api.get("/metrics")` 方法查看沙箱实例的监控数据指标。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 获取沙箱实例的监控数据指标
resp = sandbox._envd_api.get("/metrics")
metrics_data = resp.json()
# 返回字段包括:
# cpu_count, cpu_used_pct, mem_total, mem_used, mem_cache, disk_total, disk_used
print(metrics_data)
```

检查沙箱运行状态

您可以通过 `sandbox.is_running` 方法检查沙箱实例是否处于运行状态。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 检查沙箱是否处于运行状态
running = sandbox.is_running()
print(f"沙箱运行状态: {running}")
```

修改沙箱超时时间

您可以通过 `sandbox.set_timeout` 方法修改沙箱实例的超时时间，避免长时间运行的任务被意外终止。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 修改超时时间为600秒
sandbox.set_timeout(600)
```

连接已有沙箱实例

您可以通过 `Sandbox.connect` 方法，使用沙箱ID连接到已存在的沙箱实例，适用于跨会话复用沙箱的场景。

```
import os
from e2b import Sandbox

E2B_API_KEY = os.environ.get("E2B_API_KEY", "")

# 通过沙箱ID连接到已存在的沙箱实例
sandbox = Sandbox.connect("<your_sandbox_id>", api_key=E2B_API_KEY)
print(f"已连接沙箱实例, ID: {sandbox.sandbox_id}")
```

查询沙箱实例列表

您可以通过 *Sandbox.list* 方法，返回沙箱实例列表。

```
import os
from getpass import getpass
from e2b import Sandbox
API_URL = "https://agentsphere.<region>.myhuaweicloud.com" # 根据真实情况修改
E2B_API_KEY = os.environ.get("E2B_API_KEY", "")

paginator = Sandbox.list(
    limit=10,
    api_key=E2B_API_KEY,
    api_url=API_URL,
    domain="agentsphere.<region>.myhuaweicloud.com", # 根据真实情况修改
    request_timeout=30,
)
sandboxes = paginator.next_items()
print(f'count={len(sandboxes)}, has_next={paginator.has_next}')
for sandbox in sandboxes:
    print(
        f'sandbox_id={sandbox.sandbox_id}, "
        f'template_id={sandbox.template_id}, state={sandbox.state.value}"
    )
```

暂停沙箱实例

您可以通过 *sandbox.pause* 方法暂停沙箱实例以节省资源，暂停后可通过 *Sandbox.connect* 方法重新连接并恢复。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 暂停沙箱实例
sandbox.pause()
print("沙箱实例已暂停")
```



注意

沙箱最大运行时长不能超过24小时。

删除沙箱实例



注意

删除操作不可逆，沙箱实例内的所有数据将被清除且无法恢复。请在删除前确认已保存必要数据。

您可以通过 *sandbox.kill* 方法删除沙箱实例。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 删除沙箱实例
sandbox.kill()
print("沙箱实例已删除")
```

完整代码示例

通过指定模板创建沙箱实例，创建成功后返回沙箱对象，您可以基于该对象执行后续操作。

```
import os
import time
from e2b import Sandbox

# 从环境变量中读取API Key
E2B_API_KEY = os.environ.get("E2B_API_KEY", "")

# 使用模板创建沙箱实例，超时时间300秒
def test_create_sandbox():
    sandbox = Sandbox.create(
        template="<template_id>",
        api_key=E2B_API_KEY,
        timeout=300,
    )
    assert sandbox.sandbox_id is not None and sandbox.sandbox_id != ""
    print(f"[PASS] create sandbox: {sandbox.sandbox_id}")
    return sandbox

# 获取沙箱实例的详细信息
def test_get_info(sandbox):
    info = sandbox.get_info()
    assert info is not None
    print(f"[PASS] get_info: {info}")

# 检查沙箱实例是否处于运行状态
def test_is_running(sandbox):
    running = sandbox.is_running()
    assert running is True
    print(f"[PASS] is_running: {running}")

# 修改沙箱实例的超时时间为600秒
def test_set_timeout(sandbox):
    sandbox.set_timeout(600)
    print("[PASS] set_timeout(600)")

# 查询当前账号下所有沙箱实例
def test_list_sandboxes():
    E2B_API_KEY = os.environ.get("E2B_API_KEY", "")
    API_URL = os.environ.get("API_URL", "")
    paginator = Sandbox.list(
        limit=10,
        api_key=E2B_API_KEY,
        api_url=API_URL,
        domain="agentsphere.<region>.myhuaweicloud.com", # 根据实际情况修改
        request_timeout=30,
    )
    sandboxes = paginator.next_items()
    print(f"count={len(sandboxes)}, has_next={paginator.has_next}")
    for sandbox in sandboxes:
        print(
            f"sandbox_id={sandbox.sandbox_id}, "
            f"template_id={sandbox.template_id}, state={sandbox.state.value}"
        )

# 通过沙箱ID连接到已存在的沙箱实例
def test_connect_sandbox(sandbox_id):
    sandbox = Sandbox.connect(sandbox_id, api_key=E2B_API_KEY)
    assert sandbox.sandbox_id == sandbox_id
    print(f"[PASS] connect sandbox: {sandbox.sandbox_id}")
    return sandbox

# 暂停沙箱实例以节省资源
def test_pause_sandbox(sandbox):
    try:
```

```
sandbox.pause()
print("[PASS] pause sandbox")
except Exception as e:
    print(f"[SKIP] pause not supported or error: {e}")

# 删除沙箱实例，操作不可逆
def test_kill_sandbox(sandbox):
    sandbox.kill()
    print(f"[PASS] kill sandbox: {sandbox.sandbox_id}")
```

相关文档

- [创建沙箱模板](#)：了解如何创建自定义沙箱模板，预装所需的运行环境和依赖包。
- [创建API Key](#)：了解如何创建和管理API Key，获取访问凭据。
- [创建网关实例](#)：了解Agent网关的配置和管理，获取数据面访问地址。
- [E2B SDK官方文档](#)：了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。

3 文件操作

AgentSphere沙箱实例提供了独立的文件系统，您可以通过SDK进行文件读写、目录管理、文件查询等操作。

前提条件

以下示例均基于E2B SDK，请确保已完成[配置本地E2B SDK执行环境](#)。

文件操作

写入和读取文件

您可以使用 *files.write* 方法向沙箱实例写入文件，使用 *files.read* 方法读取文件内容。支持写入文本和JSON等格式的数据。

```
from e2b import Sandbox

# template字段的值为已有的template_id
sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token，可选，如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config._ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 写入文本文件
sandbox.files.write("/tmp/test.txt", "Hello AgentSphere!")

# 读取文件内容
content = sandbox.files.read("/tmp/test.txt")
print(content) # 输出: Hello AgentSphere!

# 写入JSON文件
import json
sandbox.files.write("/tmp/data.json", json.dumps({"key": "value", "count": 42}))

# 读取并解析JSON
raw = sandbox.files.read("/tmp/data.json")
parsed = json.loads(raw)
print(parsed) # 输出: {'key': 'value', 'count': 42}
```

检查文件是否存在

您可以使用 *files.exists* 方法检查沙箱实例中指定路径的文件是否存在，返回布尔值。

```
from e2b import Sandbox
```

```
sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token, 可选, 如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 写入文件后检查
sandbox.files.write("/tmp/check.txt", "exists!")
print(sandbox.files.exists("/tmp/check.txt")) # 输出: True

# 检查不存在的文件
print(sandbox.files.exists("/tmp/nonexistent.txt")) # 输出: False
```

创建目录

您可以使用 *files.make_dir* 方法在沙箱实例中创建目录, 支持递归创建多级目录。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token, 可选, 如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 递归创建多级目录
sandbox.files.make_dir("/tmp/nested/deep/dir")
print(sandbox.files.exists("/tmp/nested/deep/dir")) # 输出: True
```

列出目录内容

您可以使用 *files.list* 方法列出指定目录下的文件和子目录, 返回文件信息列表。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token, 可选, 如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 创建目录并写入文件
sandbox.files.make_dir("/tmp/list_test")
sandbox.files.write("/tmp/list_test/a.txt", "aaa")
sandbox.files.write("/tmp/list_test/b.txt", "bbb")

# 列出目录内容
items = sandbox.files.list("/tmp/list_test")
for item in items:
    print(f"文件名: {item.name}")
# 输出:
# 文件名: a.txt
# 文件名: b.txt
```

重命名文件或目录

您可以使用 *files.rename* 方法重命名沙箱实例中的文件或目录。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token, 可选, 如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 写入文件后重命名
```

```
sandbox.files.write("/tmp/src.txt", "rename me")
sandbox.files.rename("/tmp/src.txt", "/tmp/dst.txt")
print(sandbox.files.exists("/tmp/dst.txt")) # 输出: True
print(sandbox.files.exists("/tmp/src.txt")) # 输出: False
```

删除文件

您可以使用 *files.remove* 方法删除沙箱实例中的指定文件。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token，可选，如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 写入文件后删除
sandbox.files.write("/tmp/remove_me.txt", "delete me")
sandbox.files.remove("/tmp/remove_me.txt")
print(sandbox.files.exists("/tmp/remove_me.txt")) # 输出: False
```

删除目录



注意

删除目录操作为递归删除，将同时删除目录下的所有子目录和文件，且不可恢复。

您可以使用 *files.remove* 方法递归删除目录及其下的所有内容。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token，可选，如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 创建目录和文件
sandbox.files.make_dir("/tmp/rmdir/sub")
sandbox.files.write("/tmp/rmdir/sub/file.txt", "data")

# 递归删除目录
sandbox.files.remove("/tmp/rmdir")
print(sandbox.files.exists("/tmp/rmdir")) # 输出: False
```

查看文件属性

您可以使用 *files.get_info* 方法获取文件的大小、修改时间等属性信息。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token，可选，如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 写入文件后获取属性
sandbox.files.write("/tmp/info.txt", "info test")
info = sandbox.files.get_info("/tmp/info.txt")
print(info)
```

生成上传/下载的 url 链接

您可以使用 `upload_url`/`download_url` 生成临时访问链接，可以在浏览器或外部工具中直接上传或下载文件。

```
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 上传文件
url = sandbox.upload_url("/tmp/e2b_upload_test.txt")
assert url is not None and url != ""
print(f"[PASS] upload_url: {url}")

# 下载文件
sandbox.files.write("/tmp/e2b_download_test.txt", "download me")
url = sandbox.download_url("/tmp/e2b_download_test.txt")
assert url is not None and url != ""
print(f"[PASS] download_url: {url}")

sandbox.kill()
```

文件上传/下载/解压缩

您需要以执行命令的方式进行解压缩文件。

```
import time
from e2b_code_interpreter import Sandbox
from e2b.sandbox.filesystem.filesystem import WriteEntry

sbx = Sandbox.create(
    template="<template_id>",
    validate_api_key=False,
    timeout=300,
)
time.sleep(5)

# 批量写入文件
sbx.files.write_files([
    WriteEntry(path="/home/user/multi_a.txt", data="file A content"),
    WriteEntry(path="/home/user/multi_b.txt", data="file B content"),
])
logger.info("write_files (batch upload) OK")

# 读取批量写入的文件
a_content = sbx.files.read("/home/user/multi_a.txt", format="text")
assert a_content == "file A content"
logger.info("batch upload read back OK")

# 流式读取文件
sbx.files.write("/home/user/download_me.txt", "download this file")
stream = sbx.files.read("/home/user/download_me.txt", format="stream")
downloaded = b"".join(stream)
assert b"download this file" in downloaded
logger.info("stream download OK")

# 压缩文件
sbx.commands.run("cd /home/user && echo 'compress test' > file1.txt && echo 'more data' > file2.txt")
sbx.commands.run("cd /home/user && zip archive.zip file1.txt file2.txt")
assert sbx.files.exists("/home/user/archive.zip") is True
logger.info("zip compress OK")

# 解压文件
sbx.commands.run("cd /home/user && mkdir -p extracted && unzip archive.zip -d extracted")
f1 = sbx.files.read("/home/user/extracted/file1.txt", format="text")
assert "compress test" in f1
logger.info("unzip extract OK")
```

```
# 清理文件
sbx.commands.run("rm -rf /home/user/multi_a.txt /home/user/multi_b.txt /home/user/download_me.txt /
home/user/file1.txt /home/user/file2.txt /home/user/archive.zip /home/user/extracted")

sbx.kill()
```

完整代码示例

以下示例基于已创建的Sandbox对象。

```
import os
import json
from e2b import Sandbox

sandbox = Sandbox.create(template="<template_id>")

# 设置沙箱实例的traffic_access_token，可选，如果已经安装前提条件中的patch文件则无需此设置
sandbox.connection_config.ConnectionConfig__extra_sandbox_headers["E2b-Traffic-Access-Token"] =
sandbox.traffic_access_token

# 读写文本文件
def test_write_and_read_text(sandbox):
    sandbox.files.write("/tmp/e2b_test.txt", "Hello E2B Filesystem!")
    content = sandbox.files.read("/tmp/e2b_test.txt")
    assert "Hello E2B Filesystem!" in content
    print(f"[PASS] write & read text: {content.strip()}")

# 读写JSON文件
def test_write_json(sandbox):
    payload = {"key": "value", "count": 42}
    sandbox.files.write("/tmp/e2b_data.json", json.dumps(payload))
    raw = sandbox.files.read("/tmp/e2b_data.json")
    parsed = json.loads(raw)
    assert parsed["key"] == "value"
    assert parsed["count"] == 42
    print(f"[PASS] write & read JSON: {parsed}")

# 检查文件是否存在
def test_exists(sandbox):
    sandbox.files.write("/tmp/e2b_exists_check.txt", "exists!")
    assert sandbox.files.exists("/tmp/e2b_exists_check.txt") is True
    assert sandbox.files.exists("/tmp/nonexistent_file_abc123.txt") is False
    print("[PASS] exists check: True/False correct")

# 递归创建多级目录
def test_make_dir(sandbox):
    sandbox.files.make_dir("/tmp/e2b_nested/deep/dir")
    assert sandbox.files.exists("/tmp/e2b_nested/deep/dir") is True
    print("[PASS] make_dir nested")

# 列出目录下的文件和子目录
def test_list(sandbox):
    sandbox.files.make_dir("/tmp/e2b_list_test")
    sandbox.files.write("/tmp/e2b_list_test/a.txt", "aaa")
    sandbox.files.write("/tmp/e2b_list_test/b.txt", "bbb")
    items = sandbox.files.list("/tmp/e2b_list_test")
    names = [item.name for item in items]
    assert "a.txt" in names
    assert "b.txt" in names
    print(f"[PASS] list directory: {names}")

# 重命名文件或目录
def test_rename(sandbox):
    sandbox.files.write("/tmp/e2b_rename_src.txt", "rename me")
    sandbox.files.rename("/tmp/e2b_rename_src.txt", "/tmp/e2b_rename_dst.txt")
    assert sandbox.files.exists("/tmp/e2b_rename_dst.txt") is True
    assert sandbox.files.exists("/tmp/e2b_rename_src.txt") is False
    print("[PASS] rename file")
```

```
# 删除指定文件
def test_remove(sandbox):
    sandbox.files.write("/tmp/e2b_remove_me.txt", "delete me")
    sandbox.files.remove("/tmp/e2b_remove_me.txt")
    assert sandbox.files.exists("/tmp/e2b_remove_me.txt") is False
    print("[PASS] remove file")

# 递归删除目录及其下所有内容
def test_remove_directory(sandbox):
    sandbox.files.make_dir("/tmp/e2b_rmdir/sub")
    sandbox.files.write("/tmp/e2b_rmdir/sub/file.txt", "data")
    sandbox.files.remove("/tmp/e2b_rmdir")
    assert sandbox.files.exists("/tmp/e2b_rmdir") is False
    print("[PASS] remove directory (recursive)")

# 获取文件的大小、修改时间等属性
def test_get_info(sandbox):
    sandbox.files.write("/tmp/e2b_info.txt", "info test")
    try:
        info = sandbox.files.get_info("/tmp/e2b_info.txt")
        assert info is not None
        print(f"[PASS] get_info: {info}")
    except AttributeError:
        info = sandbox.files.get_info("/tmp/e2b_info.txt")
        print(f"[PASS] get_info (alternate): {info}")
```

相关文档

- [创建沙箱模板](#): 了解如何创建自定义沙箱模板，预装所需的运行环境和依赖包。
- [创建API Key](#): 了解如何创建和管理API Key，获取访问凭据。
- [创建网关实例](#): 了解Agent网关的配置和管理，获取数据面访问地址。
- [E2B SDK官方文档](#): 了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。

4 执行命令

您可以在沙箱实例中执行终端命令，支持指定执行用户、设置环境变量、流式输出和超时控制。

前提条件

以下示例均基于E2B SDK，请确保已完成[配置本地E2B SDK执行环境](#)。

执行命令

您可以使用`commands.run`方法在沙箱实例中执行终端命令，同时`commands.run`方法支持以下扩展参数：

参数	说明
<code>user</code>	执行命令的用户身份。
<code>on_stdout</code>	标准输出回调函数，接收实时输出数据。
<code>on_stderr</code>	标准错误回调函数，接收实时错误数据。
<code>envs</code>	命令执行的环境变量，以字典形式传入。
<code>timeout</code>	命令执行的超时时间，单位为秒。超时后命令将被中断。

基本命令执行

您可以在沙箱实例中执行终端命令并获取输出结果，并且可以通过`user`参数指定执行命令的身份。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

# 执行终端命令
response = sandbox.commands.run("ls")
print(response.stdout)

# 以root身份执行命令
response = sandbox.commands.run("ls", user="root")
print(response.stdout)
```

流式输出

对于长时间运行的命令，您可以通过 `on_stdout` 和 `on_stderr` 回调函数实时获取输出，无需等待命令执行完毕。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

command = """
echo foo
sleep 1
echo bar
"""

# 通过回调函数实时获取标准输出和标准错误
sandbox.commands.run(
    command,
    on_stdout=lambda data: print(data),
    on_stderr=lambda data: print(data)
)
```

查看环境变量

您可以通过 `envs` 字段为沙箱设置环境变量。

```
from e2b_code_interpreter import Sandbox

# 创建沙箱时设置环境变量
sandbox = Sandbox.create(
    template="<template_id>",
    envs={
        "MY_VAR": "hello",
        "MY_SECRET": "12345",
    },
)

# 设置环境变量并执行命令
result = sandbox.commands.run("echo $MY_VAR")
print(result.stdout) # 输出: hello
```

设置命令超时时间

您可以通过 `timeout` 参数为命令执行设置超时时间，单位为秒。超时后命令将被中断。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

# 设置超时时间为3秒，超时后命令将被中断
result = sandbox.commands.run("sleep 100", timeout=3)
```

完整代码示例

```
from e2b_code_interpreter import Sandbox

# 创建沙箱实例
sandbox = Sandbox.create(
    template="<template_id>",
    envs={
        "MY_VAR": "hello",
        "MY_SECRET": "12345",
    },
)

# 执行终端命令
response = sandbox.commands.run("ls")
```

```
# 可以通过user参数指定执行命令的身份，支持user和root
response = sandbox.commands.run("ls",user="root")

# 支持流式返回
command = """
echo foo
sleep 1
echo bar
"""
# 您可以根据您的需要传递on_stdout和on_stderr回调
sandbox.commands.run(
    command,
    on_stdout=lambda data: print(data),
    on_stderr=lambda data: print(data)
)

# 执行命令并设置所需环境变量
result = sandbox.commands.run("echo $MY_VAR")
assert "hello" in result.stdout
print(f"[PASS] run command with envs: {result.stdout.strip()}")

# 执行命令并设置命令运行超时时间
result = sandbox.commands.run("sleep 100", timeout=3)
```

相关文档

- [创建沙箱模板](#)：了解如何创建自定义沙箱模板，预装所需的运行环境和依赖包。
- [创建API Key](#)：了解如何创建和管理API Key，获取访问凭据。
- [创建网关实例](#)：了解Agent网关的配置和管理，获取数据面访问地址。
- [E2B SDK官方文档](#)：了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。

5 运行代码

使用代码解释器沙箱（code-interpreter）模板来创建沙箱，该模板预装了代码执行环境，支持多种编程语言，支持代码解释和结果返回，您可以直接在沙箱实例中运行代码。

📖 说明

使用代码解释器沙箱（code-interpreter）模板，需要把健康检查的端口设置为49999，协议为HTTP，检查路径为/health。

前提条件

以下示例均基于E2B SDK，请确保已完成[配置本地E2B SDK执行环境](#)。

运行代码

您可以使用`run_code`方法在沙箱实例中运行代码。

运行 Python 代码

`run_code`方法默认语言为Python，直接传入代码字符串即可。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

# 默认执行Python代码
response = sandbox.run_code("print(\"hello\")")
print(response)
```

运行其他语言代码

通过第二个位置参数指定代码的编程语言，如javascript。

📖 说明

当前支持的语言以代码解释器沙箱模板预装的环境为准。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

# 指定语言为javascript
```

```
response = sandbox.run_code("console.log(\"hello\")", "javascript")
print(response)
```

设置代码执行超时时间

通过 *timeout* 参数设置代码执行的超时时间，单位为秒。超时后代码运行将被终止。

```
from e2b_code_interpreter import Sandbox

sandbox = Sandbox.create()

# 设置超时时间为60秒
response = sandbox.run_code("print(\"hello\")", timeout=60) # 单位:秒
```

相关文档

- [创建沙箱模板](#)：了解如何创建自定义沙箱模板，预装所需的运行环境和依赖包。
- [创建API Key](#)：了解如何创建和管理API Key，获取访问凭据。
- [创建网关实例](#)：了解Agent网关的配置和管理，获取数据面访问地址。
- [E2B SDK官方文档](#)：了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。

6 进程管理

AgentSphere沙箱实例支持通过PTY（伪终端）创建交互式终端会话，适用于需要实时交互的场景，如运行交互式程序、模拟终端操作、自动化Shell脚本测试等。与 `commands.run`不同，PTY会话支持持续的输入输出流，可以模拟真实终端的行为，包括终端窗口大小调整、TTY属性检测等。

前提条件

以下示例均基于E2B SDK，请确保已完成[配置本地E2B SDK执行环境](#)。

交互式进程/PTY 会话

创建 PTY 终端会话

您可以使用`pty.create`方法在沙箱实例中创建一个PTY伪终端会话，创建时需指定终端窗口大小和超时时间。返回的终端对象包含`pid`属性，用于后续操作。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template="<template_id>")

# 创建PTY终端会话，指定窗口大小为24行80列
terminal = sandbox.pty.create(
    PtySize(rows=24, cols=80),
    timeout=30,
)
print(f"终端PID: {terminal.pid}")
```

`pty.create`方法支持以下参数：

参数	是否必选	说明
size	是	终端窗口大小，通过PtySize(rows, cols)指定行数和列数。
timeout	否	PTY终端会话的超时时间，单位为秒。超时后终端将被终止。

发送输入数据

您可以使用 `pty.send_stdin` 方法向 PTY 终端的标准输入发送数据，数据格式为 bytes。支持多次发送，模拟用户逐行输入。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template="<template_id>")

terminal = sandbox.pty.create(PtySize(rows=24, cols=80), timeout=30)

# 向终端发送命令
sandbox.pty.send_stdin(terminal.pid, b"echo 'hello from PTY'\n")
# 发送exit命令结束终端会话
sandbox.pty.send_stdin(terminal.pid, b"exit\n")
```

`pty.send_stdin`方法支持以下参数：

参数	是否必选	说明
pid	是	目标PTY终端的进程ID。
data	是	发送到标准输入的数据，类型为bytes。

等待终端结束

您可以使用终端对象的 `wait` 方法等待 PTY 终端会话结束，并通过 `on_pty` 回调函数实时接收终端输出数据。`wait` 方法会阻塞直到终端进程退出，返回包含退出码的结果对象。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template="<template_id>")

terminal = sandbox.pty.create(PtySize(rows=24, cols=80), timeout=30)
sandbox.pty.send_stdin(terminal.pid, b"echo 'hello'\n")
sandbox.pty.send_stdin(terminal.pid, b"exit\n")

# 通过回调函数实时收集终端输出
output_parts = []
def collect(data):
    output_parts.append(data.decode(errors="replace"))

# 等待终端结束，获取退出码
result = terminal.wait(on_pty=collect)
print(f"退出码: {result.exit_code}")
print(f"完整输出: {''.join(output_parts)}")
```

`terminal.wait`方法支持以下参数：

参数	是否必选	说明
on_pty	否	终端输出回调函数，接收实时输出的 bytes 数据。

调整终端窗口大小

您可以使用 `pty.resize` 方法动态调整 PTY 终端的窗口大小，适用于模拟不同终端尺寸的场景。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template=" <template_id>")

terminal = sandbox.pty.create(PtySize(rows=24, cols=80), timeout=30)

# 将终端窗口调整为40行120列
sandbox.pty.resize(terminal.pid, PtySize(rows=40, cols=120))
print("终端窗口已调整为 40x120")
```

*pty.resize*方法支持以下参数：

参数	是否必选	说明
pid	是	目标PTY终端的进程ID。
size	是	新的终端窗口大小，通过PtySize(rows, cols)指定。

终止 PTY 终端会话

您可以使用*pty.kill*方法终止指定的PTY终端会话。

 **注意**

终止PTY终端会话后，终端进程将被强制结束，未读取的输出数据将丢失。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template=" <template_id>")

terminal = sandbox.pty.create(PtySize(rows=24, cols=80), timeout=30)

# 终止PTY终端会话
sandbox.pty.kill(terminal.pid)
print(f"终端已终止: pid={terminal.pid}")
```

*pty.kill*方法支持以下参数：

参数	是否必选	说明
pid	是	要终止的PTY终端的进程ID。

重新连接 PTY 终端

您可以使用*pty.connect*方法通过进程ID重新连接到已存在的PTY终端会话，适用于跨会话复用终端的场景。

```
from e2b import Sandbox, PtySize

sandbox = Sandbox.create(template=" <template_id>")

terminal = sandbox.pty.create(PtySize(rows=24, cols=80), timeout=30)
pid = terminal.pid

# 通过PID重新连接到已存在的PTY终端
sandbox.pty.connect(pid)
```

```
print(f"已重新连接到终端: pid={pid}")
```

```
sandbox.pty.kill(pid)
```

pty.connect方法支持以下参数：

参数	是否必选	说明
pid	是	要重新连接的PTY终端的进程ID。

完整代码示例

```
import os
import time
from e2b import Sandbox, PtySize

# 从环境变量中读取API Key
E2B_API_KEY = os.environ.get("E2B_API_KEY", "")

# 创建PTY终端会话
def test_pty_create(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    assert terminal.pid is not None
    print(f"[PASS] pty create: pid={terminal.pid}")
    sandbox.pty.kill(terminal.pid)

# 发送输入并等待终端结束
def test_pty_send_stdin_and_wait(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    sandbox.pty.send_stdin(
        terminal.pid,
        b"echo 'hello from PTY'\n",
    )
    sandbox.pty.send_stdin(terminal.pid, b"exit\n")
    result = terminal.wait(on_pty=lambda data: None)
    print(f"[PASS] pty send_stdin & wait: exit_code={result.exit_code}")

# 验证PTY是否为真实终端（isatty检查）
def test_pty_isatty_check(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    sandbox.pty.send_stdin(
        terminal.pid,
        b"python3 -c \"import sys; print(sys.stdout.isatty())\"\n",
    )
    sandbox.pty.send_stdin(terminal.pid, b"exit\n")
    output_parts = []
    def collect(data):
        output_parts.append(data.decode(errors="replace"))
    result = terminal.wait(on_pty=collect)
    full_output = "".join(output_parts)
    print(f"[PASS] pty isatty check: contains 'True' = {'True' in full_output}")

# 调整终端窗口大小
def test_pty_resize(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
```

```
)
sandbox.pty.resize(terminal.pid, PtySize(rows=40, cols=120))
print("[PASS] pty resize to 40x120")
sandbox.pty.kill(terminal.pid)

# 终止PTY终端会话
def test_pty_kill(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    sandbox.pty.kill(terminal.pid)
    print(f"[PASS] pty kill: pid={terminal.pid}")

# 重新连接到已存在的PTY终端
def test_pty_connect(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    pid = terminal.pid
    try:
        sandbox.pty.connect(pid)
        print(f"[PASS] pty connect: reconnected to pid={pid}")
    except Exception as e:
        print(f"[SKIP] pty connect: {e}")
    sandbox.pty.kill(pid)

# 模拟Python交互式解释器会话
def test_pty_python_interactive(sandbox):
    terminal = sandbox.pty.create(
        PtySize(rows=24, cols=80),
        timeout=30,
    )
    sandbox.pty.send_stdin(terminal.pid, b"python3\n")
    time.sleep(1)
    sandbox.pty.send_stdin(terminal.pid, b"x = 2 + 3\n")
    time.sleep(1)
    sandbox.pty.send_stdin(terminal.pid, b"print(x)\n")
    time.sleep(1)
    sandbox.pty.send_stdin(terminal.pid, b"exit()\n")
    sandbox.pty.send_stdin(terminal.pid, b"exit\n")
    output_parts = []
    def collect(data):
        output_parts.append(data.decode(errors="replace"))
    result = terminal.wait(on_pty=collect)
    full_output = "".join(output_parts)
    print(f"[PASS] pty python interactive: output len={len(full_output)}")

def main():
    sandbox = None
    try:
        sandbox = Sandbox.create(template="<template_id>", api_key=E2B_API_KEY, timeout=300)
        print(f"Sandbox created: {sandbox.sandbox_id}")
        time.sleep(1)
        test_pty_create(sandbox)
        test_pty_send_stdin_and_wait(sandbox)
        test_pty_isatty_check(sandbox)
        test_pty_resize(sandbox)
        test_pty_kill(sandbox)
        test_pty_connect(sandbox)
        test_pty_python_interactive(sandbox)
    finally:
        if sandbox is not None:
            sandbox.kill()
    print("\n=== Sandbox 操作测试完成 ===")
```

```
if __name__ == "__main__":  
    main()
```

相关文档

- [创建沙箱模板](#)：了解如何创建自定义沙箱模板，预装所需的运行环境和依赖包。
- [创建API Key](#)：了解如何创建和管理API Key，获取访问凭据。
- [创建网关实例](#)：了解Agent网关的配置和管理，获取数据面访问地址。
- [E2B SDK官方文档](#)：了解E2B SDK的完整接口说明和更多用法示例。
- 如果在使用SDK时遇到连接失败、认证错误等问题，请先检查环境变量配置是否正确，特别是E2B_API_URL、E2B_SANDBOX_URL和E2B_API_KEY是否与控制台中的信息一致。