

- 基于 **Bigtable** 的数据存储与检索应用实验
- 实验目标
- **Bigtable**学习
 - 架构
 - 读写过程
- 实验内容
- 连接步骤
 - 操作分析
- 实验心得
- 总结

基于 **Bigtable** 的数据存储与检索应用实验

实验目标

了解 **Bigtable** 的基本概念和特点。掌握使用 **Bigtable** 存储和检索数据的基本操作。通过实验理解 **Bigtable** 在实际应用中的使用场景。

Bigtable学习

架构

一张大表肯定是不能存在一个服务器上的，而是被分成多份存在多个服务器上，一份就是一个逻辑单位——tablet。bigtable架构中最核心的概念是tablet。存放tablet的节点在bigtable体系中叫做tablet server，一个tablet server中存放多个tablet。bigtable在最底层把数据按照key进行排列后，进行分区，一个分区就是一个tablet，而一个tablet就是GFS中的一个文件。

读写过程

- 读取过程：客户端发起读请求：客户端应用程序指定要读取的表名、行键（**Row Key**）以及（可选的）列族（**Column Family**）、列限定符（**Qualifier**）、时间戳范围等参数，构造一个读请求。

查找 Tablet 位置： 客户端将读请求发送给 Bigtable 的 Master 节点。Master 节点根据行键在 Tablet 分布图中查找对应的 Tablet 信息（包括 Tablet ID 和负责的 Tablet Server 地址）。

转发读请求： Master 节点将查找到的 Tablet 位置信息返回给客户端。客户端直接将读请求发送给对应的 Tablet Server。

Tablet Server 处理读请求： Tablet Server 接收到读请求后，根据请求参数在本地存储的 SSTable 文件和 Memtable 中查找数据。若数据存在于 SSTable 文件：Tablet Server 通过 GFS API 查询 SSTable 文件的元数据，获取其内部数据块（chunk）在 GFS 集群中的分布信息。根据数据块位置信息，通过 GFS API 从相应的 DataNode 读取所需数据块内容。将读取到的数据块内容拼接成完整的数据项，返回给客户端。如果数据存在于多个版本（不同时间戳），按需选择合适的版本返回。如果数据跨越多个 SSTable 或 Memtable，可能需要进行多版本合并或筛选。

响应客户端： Tablet Server 将查询结果打包成响应消息，发送回客户端。客户端接收到响应后，解析并使用读取到的数据。

- 写入过程：

客户端发起写请求： 客户端应用程序指定要写入的表名、行键、列族、列限定符以及值（Cell Value）和时间戳（默认为当前时间），构造一个写请求

查找 Tablet 位置： 类似于读取过程，客户端首先将写请求发送给 Master 节点。Master 节点查找对应的 Tablet 信息并返回给客户端。

转发写请求： 客户端直接将写请求发送给对应的 Tablet Server。

Tablet Server 处理写请求： Tablet Server 接收到写请求后，将其写入内存中的 Memtable。**Memtable 刷写到 SSTable：** 当 Memtable 达到一定大小或达到其他触发条件，Tablet Server 会触发 Memtable 刷写到本地磁盘，生成新的 SSTable 文件。**生成 SSTable 文件：** Tablet Server 通过 GFS API 创建一个新的 SSTable 文件，并写入文件头、索引等元数据。将 Memtable 中的数据按需排序，并组织成 SSTable 文件格式的数据块。**分散存储数据块：** 将 SSTable 文件内部数据块（chunk）分散存储在 GFS 集群中：Tablet Server 通过 GFS API 将 SSTable 文件的数据块上传到 GFS 集群中的多个 DataNode。GFS 根据其数据分布策略（如复制因子）自动将数据块复制到其他 DataNode，确保数据冗余和高可用。

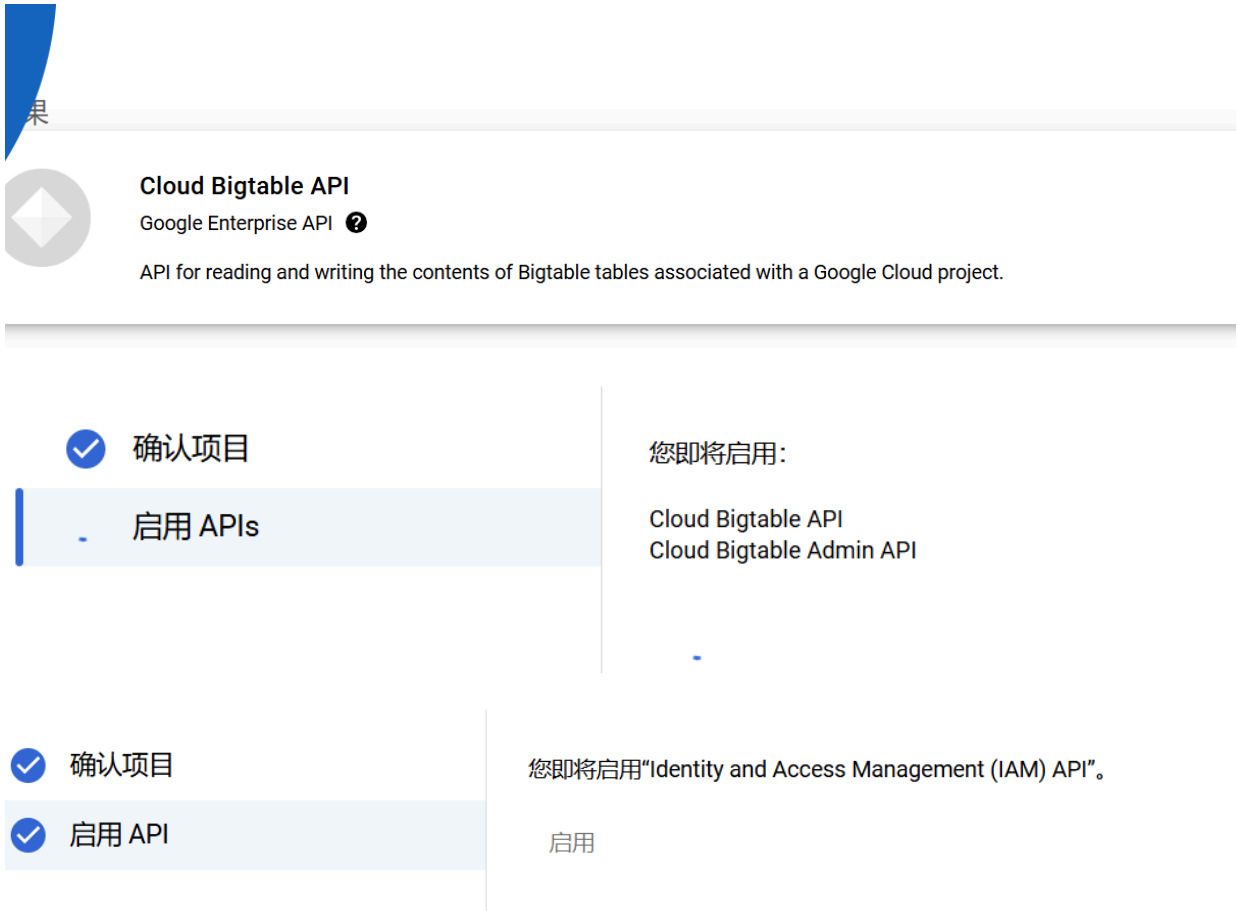
响应客户端： Tablet Server 完成写入操作后，向客户端发送确认消息，表示写入成功

实验内容

- 搭建 Bigtable 环境： 在 Google Cloud Platform 上创建一个 Bigtable 实例，并配置好必要的参数。


<https://console.cloud.google.com/>

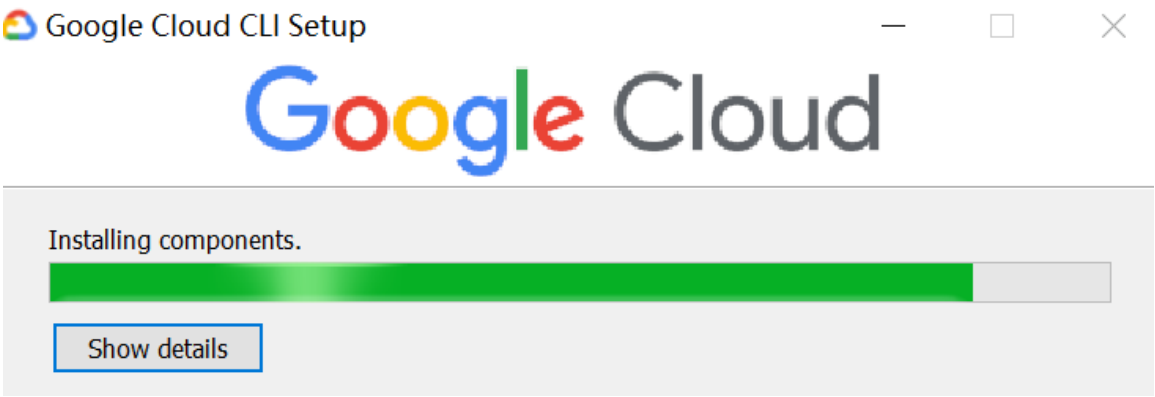
- 创建新项目.按照提示填写项目名称、项目 ID 等信息，并创建新项目。
- 启用 Bigtable API



- 创建 Bigtable 实例

1. 安装 Google Cloud CLI

 Google Cloud CLI Setup





“Google Cloud SDK”想要访问您的 Google 账号

3243106232@qq.com

这将允许“Google Cloud SDK”:

- 查看、修改、配置和删除您的 Google Cloud 数据, 和查看您 Google 账号的电子邮件地址。
- 查看和登录您的 Google Cloud SQL 实例
- 查看和管理您的 Google 计算引擎资源
- 查看和管理您在 Google App 引擎中部署的应用

```
You are logged in as: [3243106232@qq.com].

Pick cloud project to use:
[1] school-project-425811
[2] Enter a project ID
[3] Create a new project
Please enter numeric choice or text value (must exactly match list item): 1

Your current project has been set to: [school-project-425811].
```

2. 创建实例



指定实例名称



选择存储空间类型



配置您的第一个集群

- 填写实例配置 实例 ID: 指定一个唯一的实例 ID。实例类型: 选择实例的类型, 通常有开发型 (Development) 和生产型 (Production) 两种。存储类型: 选择存储类型, 通常有 SSD 和 HDD 两种。区域: 选择实例所在的地理位置, 应根据你的应用需求选择合适的区域。集群 ID: 设置集群的 ID。节点数: 设置集群的节点数目。
 - 配置其他参数: 根据需求配置其他参数, 比如选择是否启用数据加密、设置访问权限等。
 - 确认并创建: 完成配置后, 点击“创建”按钮, 等待一段时间直到实例创建完成。
 - 连接到 Bigtable 实例
- 编写数据存储与检索程序: 编写一个简单的程序, 实现以下功能: 连接到 Bigtable 实例。创建一个数据表。向数据表中插入一些样本数据。根据行键 (Row Key) 或者范围检索数据。更新或删除数据。关闭与 Bigtable 的连接。实验验证与分析: 运行你编写的程序, 验证数据的存储和检索功能是否正常工作, 并进行性能分析和比较。可以尝试在不同规模的数据量下测试程序的表现。

连接步骤

```
from google.cloud import bigtable
from google.cloud.bigtable import column_family, row_filters

# 配置连接参数
PROJECT_ID = 'school-project-425811'
INSTANCE_ID = 'ls-ls-ls'
TABLE_ID = 'sample-table'

# 创建 Bigtable 客户端
client = bigtable.Client(project=PROJECT_ID, admin=True)
instance = client.instance(INSTANCE_ID)

# 创建数据表
table = instance.table(TABLE_ID)
if not table.exists():
    print(f"Creating table {TABLE_ID}...")
    column_families = {
        'cf1': column_family.MaxVersionsGCRule(2)
    }
    table.create(column_families=column_families)

# 插入样本数据
rows = []
row_key = 'row-key-1'
row = table.direct_row(row_key)
row.set_cell('cf1', 'field1', 'value1')
row.set_cell('cf1', 'field2', 'value2')
rows.append(row)

row_key = 'row-key-2'
row = table.direct_row(row_key)
row.set_cell('cf1', 'field1', 'value3')
row.set_cell('cf1', 'field2', 'value4')
rows.append(row)

table.mutate_rows(rows)
print("Inserted sample data.")

# 根据行键检索数据
key_to_retrieve = 'row-key-1'
row = table.read_row(key_to_retrieve)
if row:
    print(f"Row key: {key_to_retrieve}")
    for column_family_id, columns in row.cells.items():
        for column, cell_list in columns.items():
            for cell in cell_list:
                print(f"Column: {column}, Value: {cell.value.decode('utf-8')}")

# 更新数据
row = table.direct_row('row-key-1')
row.set_cell('cf1', 'field1', 'updated-value')
row.commit()
```

```
print("Updated data.")

# 删除数据
row = table.direct_row('row-key-2')
row.delete()
row.commit()
print("Deleted data.")

# 关闭连接
client.close()
print("Closed connection.")
```

因防火墙问题未能成功链接

操作分析

- 创建数据表

```
table = instance.table(TABLE_ID)
if not table.exists():
    print(f"Creating table {TABLE_ID}...")
    column_families = {
        'cf1': column_family.MaxVersionsGCRule(2)
    }
    table.create(column_families=column_families)
```

创建一个表对象。如果表不存在，则创建表并定义一个列族 cf1，设置其最大版本数为 2。

- 插入样本数据

```
rows = []
row_key = 'row-key-1'
row = table.direct_row(row_key)
row.set_cell('cf1', 'field1', 'value1')
row.set_cell('cf1', 'field2', 'value2')
rows.append(row)

row_key = 'row-key-2'
row = table.direct_row(row_key)
row.set_cell('cf1', 'field1', 'value3')
row.set_cell('cf1', 'field2', 'value4')
rows.append(row)
```

```
table.mutate_rows(rows)
print("Inserted sample data.")
```

创建两行数据，并为每行设置多个单元格的值，然后将这些行插入到表中。

- 根据行键检索数据

```
key_to_retrieve = 'row-key-1'
row = table.read_row(key_to_retrieve)
if row:
    print(f"Row key: {key_to_retrieve}")
    for column_family_id, cells in row.cells.items():
        for column_qualifier, cell in cells.items():
            print(f"Column Family: {column_family_id}, Column: {column_qualifier.decode('utf-8')}, Value: {cell[0].value.decode('utf-8')}")
```

根据指定的行键 `row-key-1` 检索数据。如果行存在，则打印行键和对应的列族、列限定符及其值。

- 更新数据

```
row_key = 'row-key-1'
row = table.direct_row(row_key)
row.set_cell('cf1', 'field1', 'updated-value')
row.commit()
print("Updated data for row-key-1.")
```

使用行键 `row-key-1` 更新指定单元格的值，然后提交更新。

- 删除数据

```
row_key = 'row-key-2'
row = table.direct_row(row_key)
row.delete()
row.commit()
print("Deleted data for row-key-2.")
```

使用行键 `row-key-2` 删除指定行的数据，并提交删除操作。

实验心得

环境配置：正确配置 Google Cloud 环境和服务账号是成功执行 Bigtable 操作的前提。

API 使用：通过本实验，掌握了 Bigtable Python 客户端的基本使用方法，包括表和行的操作。

数据操作：实验展示了如何在 Bigtable 中进行增删改查操作，熟悉这些基本操作是使用 Bigtable 的基础。

总结

本实验为 Google Cloud Bigtable 的基础应用提供了完整的实践流程，从环境配置到数据操作，每一步都展示了如何有效地与 Bigtable 进行交互。通过此实验，理解了 Bigtable 的基本概念和操作方法，为进一步学习和使用 Bigtable 奠定了基础。