



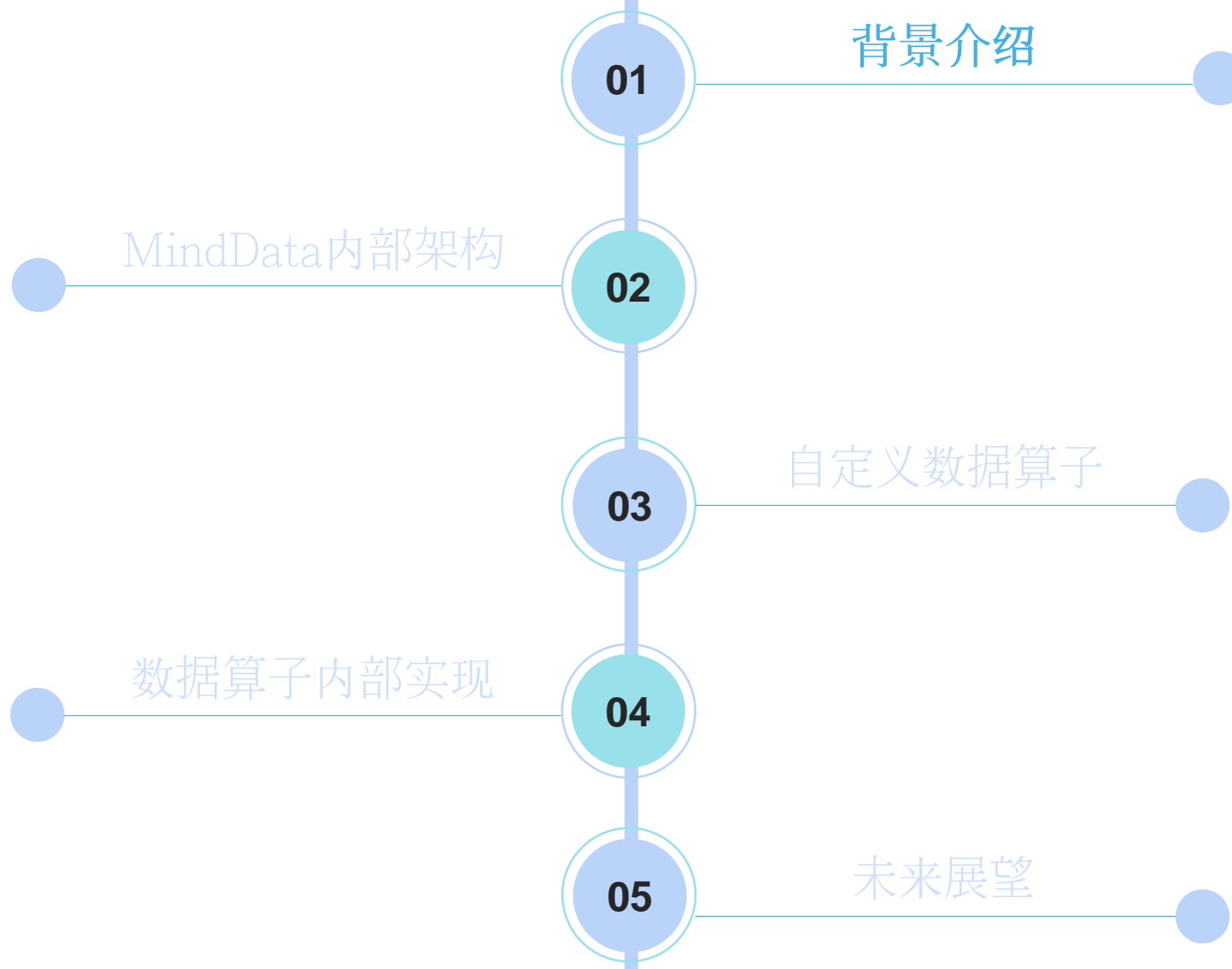
MindSpore

MindSpore数据算子内部实现

作者: Xiao Tianci

目录

contents

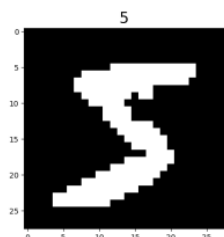


背景介绍

数据加载

将磁盘或数据库中的原始数据集信息加载到内存空间，构建成框架通用格式Tensor的形式

图像



文本

Welcome to Beijing!
北京欢迎您!
我喜欢English!

音频



数据加载

数据增强

模型训练

数据增强

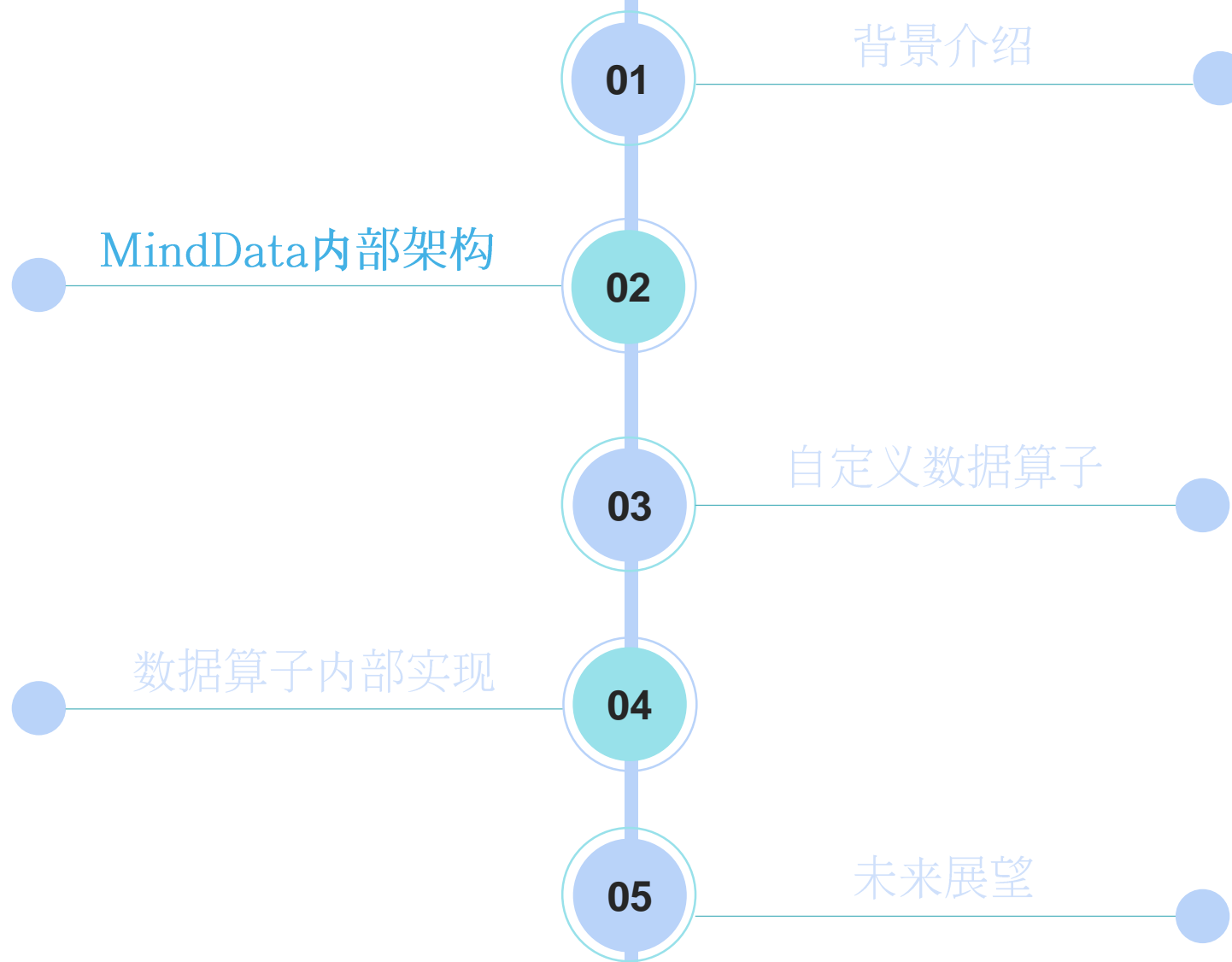
对数据Tensor进行处理变化，使其更易于学习，提升模型训练效果，如对图像进行**旋转**、**缩放**、**裁剪**；对文本进行**分词**；对音频进行**滤波**、**重采样**、**频谱变换**等。

数据增强算子： 提供对数据Tensor进行特定处理变化能力的API接口，如[RandomResize](#)、[RandomCrop](#)等。

数据集加载算子： 提供将原始数据集信息加载到内存空间能力的API接口，如[Cifar10Dataset](#)、[CocoDataset](#)等。

目录

contents



MindData内部架构

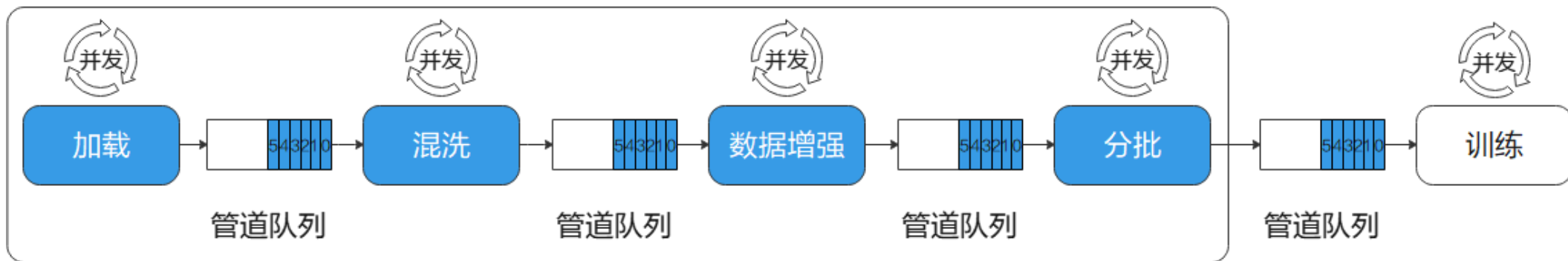
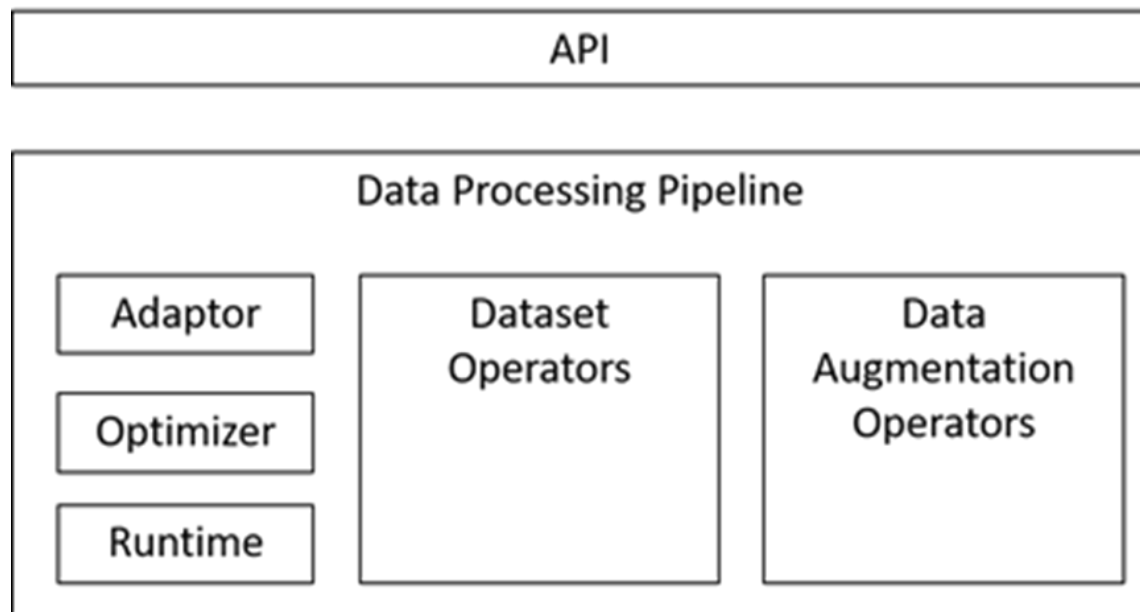
定义数据管道

```
dataset = ds.ImageFolderDataset(path)
```

```
dataset = dataset.shuffle(10)
```

```
dataset = dataset.map(transforms)
```

```
dataset = dataset.batch(32)
```




数据处理流水线

MindData内部架构

➤ 数据集加载算子

Vision	
<code>mindspore.dataset.CelebADataset</code>	A source dataset for reading and parsing CelebA dataset.
<code>mindspore.dataset.Cifar100Dataset</code>	A source dataset for reading and parsing Cifar100 dataset.
<code>mindspore.dataset.Cifar10Dataset</code>	A source dataset for reading and parsing Cifar10 dataset.
<code>mindspore.dataset.CocoDataset</code>	A source dataset for reading and parsing COCO dataset.
<code>mindspore.dataset.ImageFolderDataset</code>	A source dataset that reads images from a tree of directories.
<code>mindspore.dataset.MnistDataset</code>	A source dataset for reading and parsing the MNIST dataset.
<code>mindspore.dataset.VOCDataset</code>	A source dataset for reading and parsing VOC dataset.
Text	
<code>mindspore.dataset.CLUEDataset</code>	A source dataset that reads and parses CLUE datasets.
Graph	
<code>mindspore.dataset.GraphData</code>	Reads the graph dataset used for GNN training from the shared file and database.

Standard Format 	
<code>mindspore.dataset.CSVDataset</code>	A source dataset that reads and parses comma-separated values (CSV) datasets.
<code>mindspore.dataset.ManifestDataset</code>	A source dataset for reading images from a Manifest file.
<code>mindspore.dataset.MindDataset</code>	A source dataset for reading and parsing MindRecord dataset.
<code>mindspore.dataset.TextFileDataset</code>	A source dataset that reads and parses datasets stored on disk in text format.
<code>mindspore.dataset.TFRecordDataset</code>	A source dataset for reading and parsing datasets stored on disk in TFData format.
User Defined	
<code>mindspore.dataset.GeneratorDataset</code>	A source dataset that generates data from Python by invoking Python data source each epoch.
<code>mindspore.dataset.NumpySlicesDataset</code>	Creates a dataset with given data slices, mainly for loading Python data into dataset.
<code>mindspore.dataset.PaddedDataset</code>	Creates a dataset with filler data provided by user.

MindData内部架构

➤ 数据增强算子

mindspore.dataset.vision.c_transforms

<code>mindspore.dataset.vision.c_transforms.AutoContrast</code>	Apply automatic contrast on input image.
<code>mindspore.dataset.vision.c_transforms.BoundingBoxAugment</code>	Apply a given image transform on a random selection of bounding box regions of a given image.
<code>mindspore.dataset.vision.c_transforms.CenterCrop</code>	Crop the input image at the center to the given size.
<code>mindspore.dataset.vision.c_transforms.CutMixBatch</code>	Apply CutMix transformation on input batch of images and labels.

mindspore.dataset.vision.py_transforms

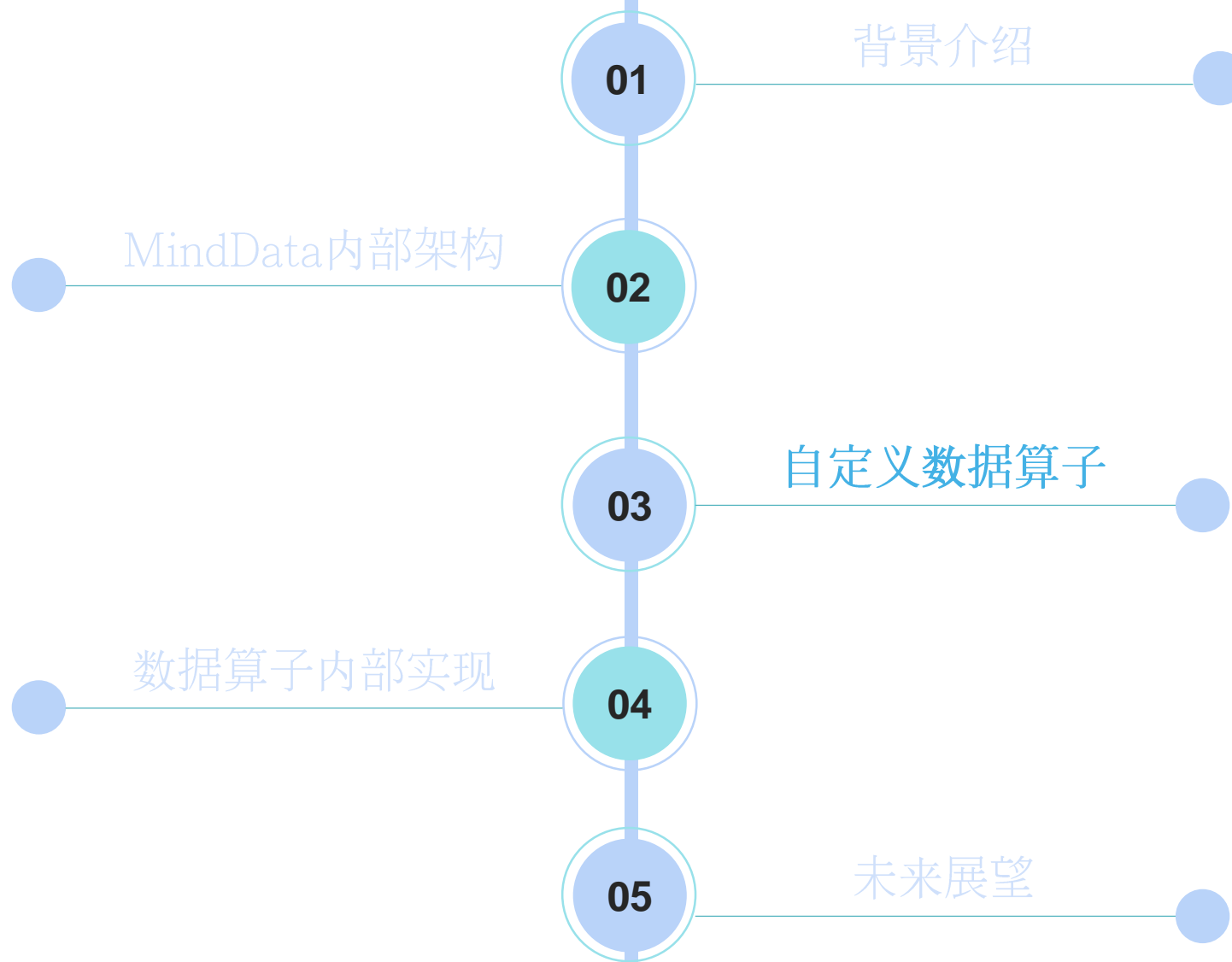
<code>mindspore.dataset.vision.py_transforms.AutoContrast</code>	Automatically maximize the contrast of the input PIL image.
<code>mindspore.dataset.vision.py_transforms.CenterCrop</code>	Crop the central region of the input PIL image to the given size.
<code>mindspore.dataset.vision.py_transforms.Cutout</code>	Randomly cut (mask) out a given number of square patches from the input NumPy image array of shape (C, H, W).
<code>mindspore.dataset.vision.py_transforms.Decode</code>	Decode the input image to PIL image format in RGB mode.

mindspore.dataset.text.transforms

API Name	Description
<code>mindspore.dataset.text.transforms.BasicTokenizer</code>	Tokenize a scalar tensor of UTF-8 string by specific rules.
<code>mindspore.dataset.text.transforms.BertTokenizer</code>	Tokenizer used for Bert text process.
<code>mindspore.dataset.text.transforms.CaseFold</code>	Apply case fold operation on UTF-8 string tensor, which is aggressive that can convert more characters into lower case.

目录

contents



自定义数据算子

➤ 自定义数据集加载

- ✓ 对于目前MindSpore不支持直接加载的数据集，可以构造自定义数据集类
- ✓ 然后通过GeneratorDataset接口实现自定义方式的数据加载。

```
import numpy as np

np.random.seed(58)

class DatasetGenerator:
    def __init__(self):
        self.data = np.random.sample((5, 2))
        self.label = np.random.sample((5, 1))

    def __getitem__(self, index):
        return self.data[index], self.label[index]

    def __len__(self):
        return len(self.data)
```

- `__init__`: 数据集对象实例化/初始化，用户可以在此进行变量初始化等操作
- `__getitem__`: 重写`__getitem__`函数以支持随机访问，能够获取指定index索引的数据并返回
- `__len__`: 重写`__len__`函数，能够获取数据集的样本数量

```
dataset_generator = DatasetGenerator()
dataset = ds.GeneratorDataset(dataset_generator, ["data", "label"], shuffle=False)

for data in dataset.create_dict_iterator():
    print('{}{}'.format(data["data"], '{}'.format(data["label"])))

[0.36510558 0.45120592] [0.78888122]
[0.49606035 0.07562207] [0.38068183]
[0.57176158 0.28963401] [0.16271622]
[0.30880446 0.37487617] [0.54738768]
[0.81585667 0.96883469] [0.77994068]
```

自定义数据算子

➤ 自定义数据集增强

对于目前MindSpore不支持的数据增强方法，可以构造自定义数据增强函数，然后通过map接口插入数据处理管道。

- `generator_func()`: 自定义数据集方法，生成5条数据
- `pyfunc()`: 自定义数据增强方法，将数据数值乘以2

```
import numpy as np
import mindspore.dataset as ds

def generator_func():
    for i in range(5):
        yield (np.array([i, i+1, i+2]),)

def pyfunc(x):
    return x*2

dataset = ds.GeneratorDataset(generator_func, ["data"])

for data in dataset.create_dict_iterator():
    print(data)
```

✓ 数据处理前:

```
{'data': Tensor(shape=[3], dtype=Int64, value= [0, 1, 2])}
{'data': Tensor(shape=[3], dtype=Int64, value= [1, 2, 3])}
{'data': Tensor(shape=[3], dtype=Int64, value= [2, 3, 4])}
{'data': Tensor(shape=[3], dtype=Int64, value= [3, 4, 5])}
{'data': Tensor(shape=[3], dtype=Int64, value= [4, 5, 6])}
```

✓ 数据处理操作:

```
dataset = dataset.map(operations=pyfunc, input_columns=["data"])

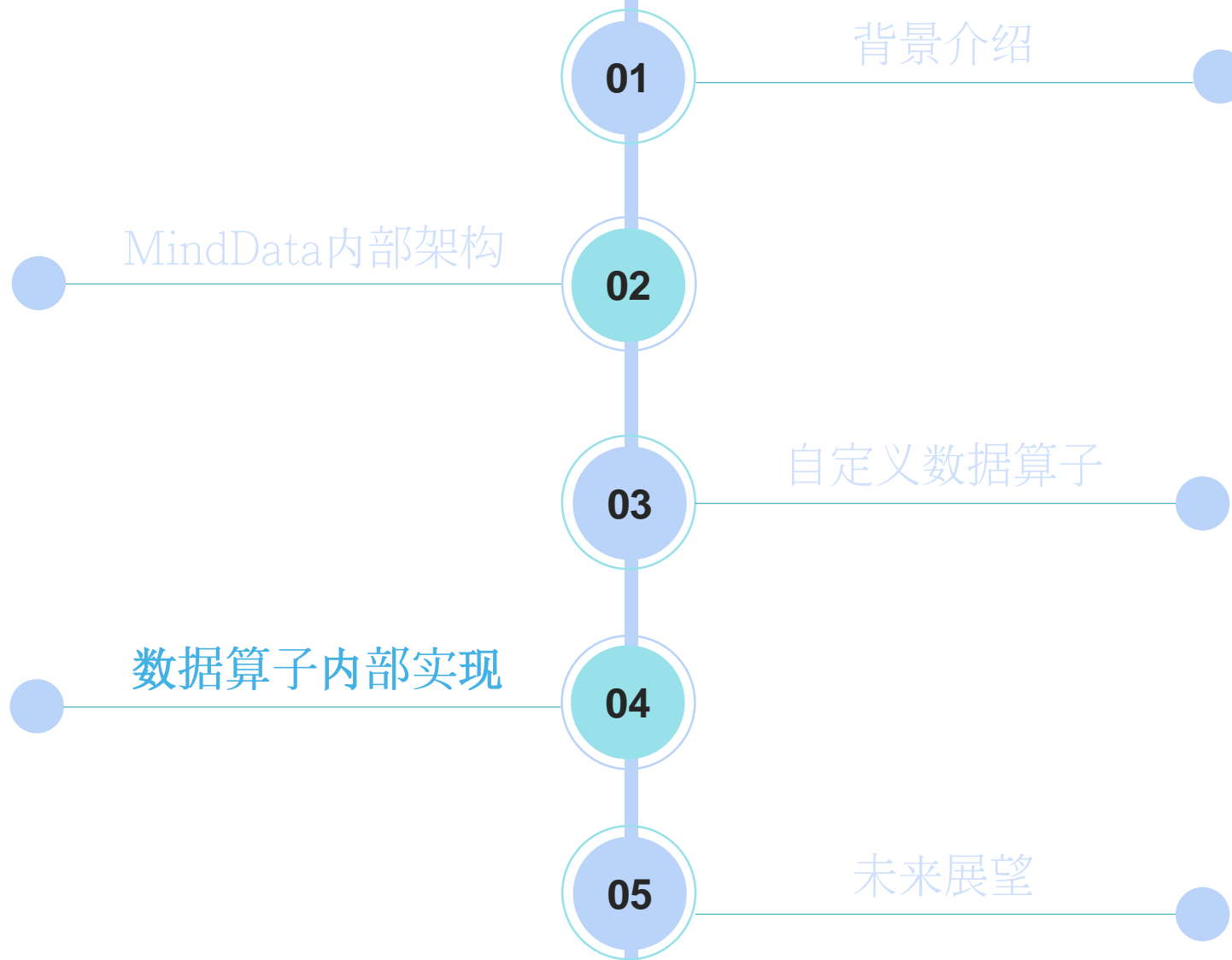
for data in dataset.create_dict_iterator():
    print(data)
```

✓ 数据处理后:

```
{'data': Tensor(shape=[3], dtype=Int64, value= [0, 2, 4])}
{'data': Tensor(shape=[3], dtype=Int64, value= [2, 4, 6])}
{'data': Tensor(shape=[3], dtype=Int64, value= [4, 6, 8])}
{'data': Tensor(shape=[3], dtype=Int64, value= [ 6,  8, 10])}
{'data': Tensor(shape=[3], dtype=Int64, value= [ 8, 10, 12])}
```

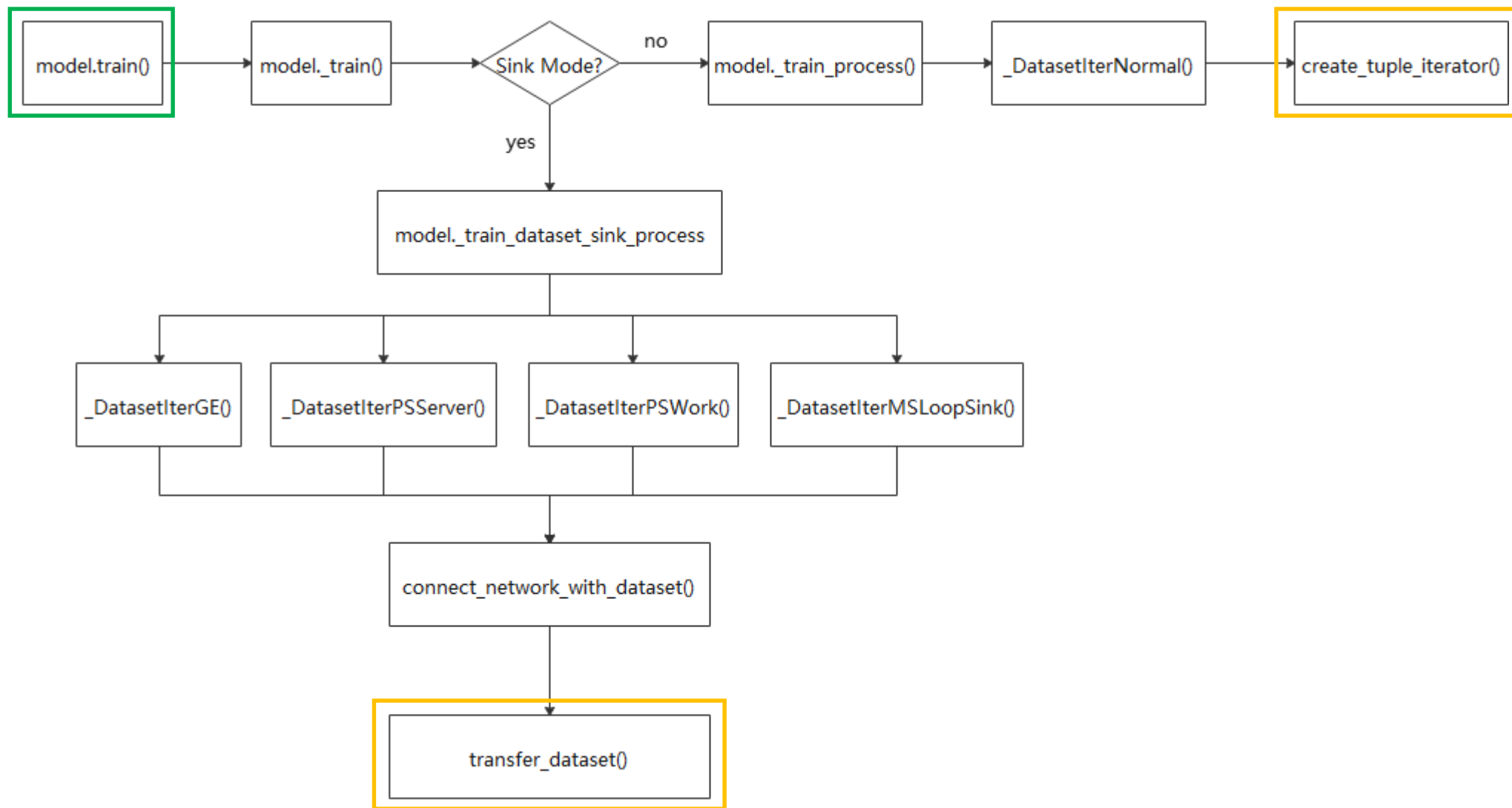
目录

contents



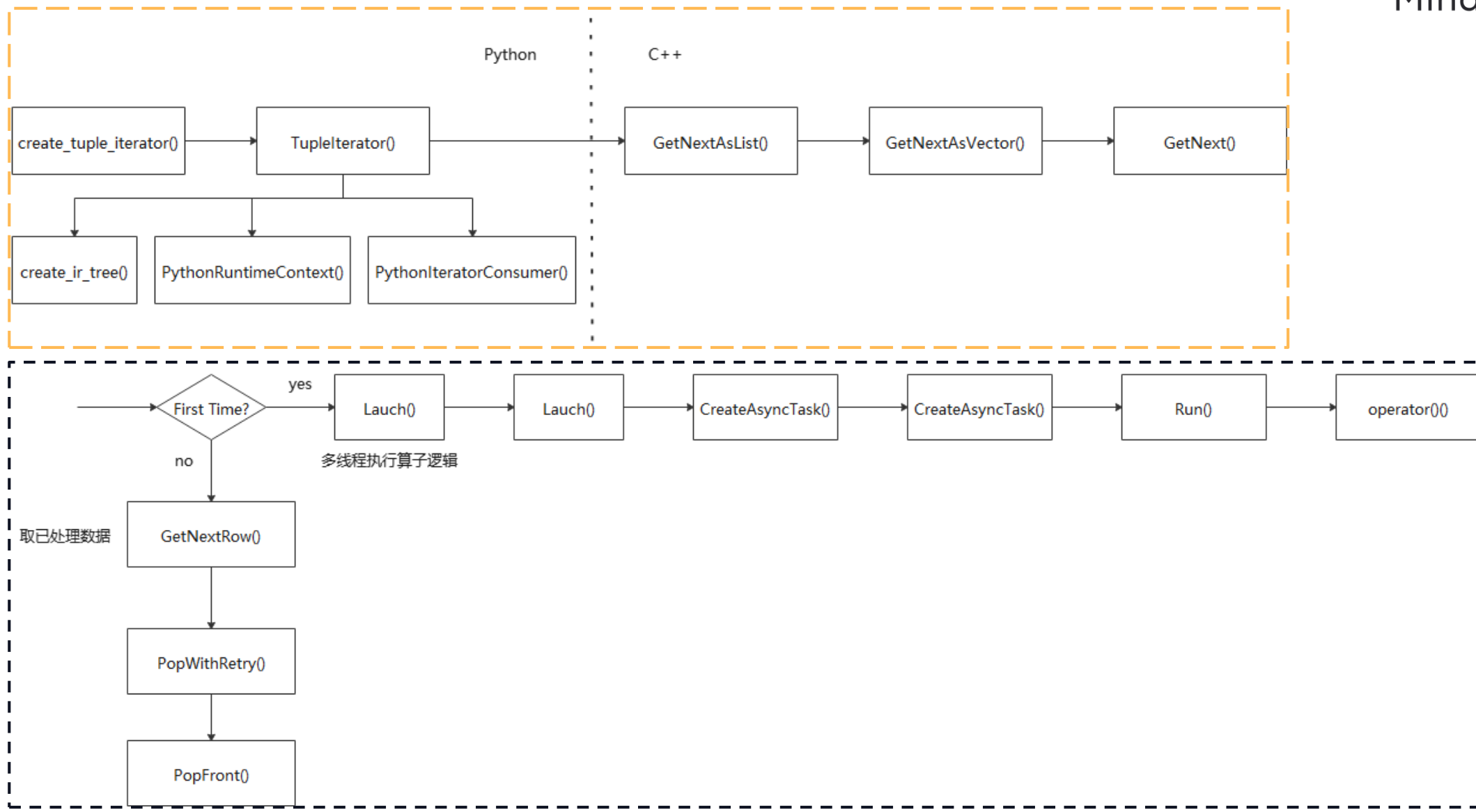
数据算子内部实现

➤ 网络训练调用数据处理流程



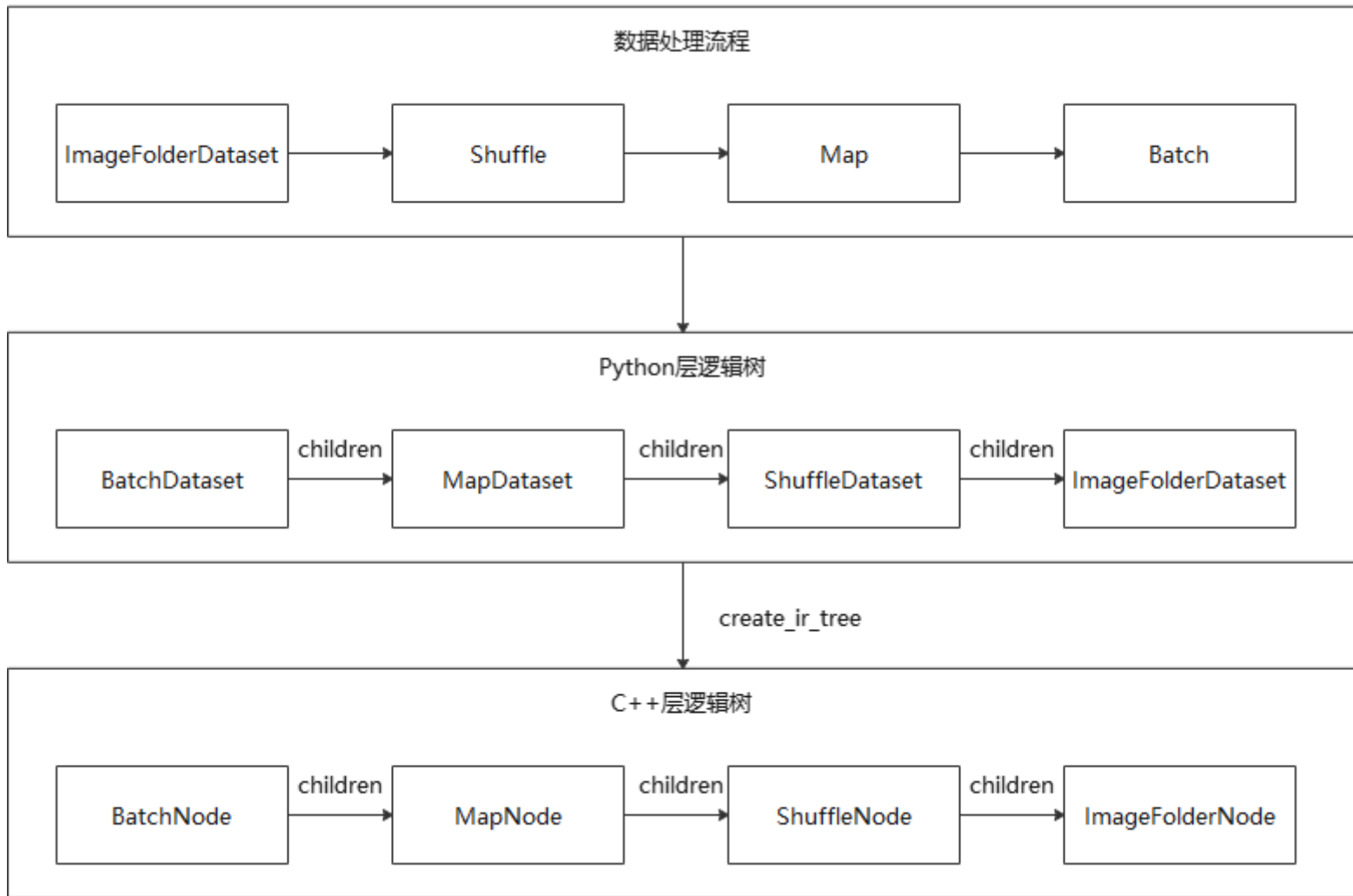
数据算子内部实现

➤ 数据迭代调用流程



数据算子内部实现

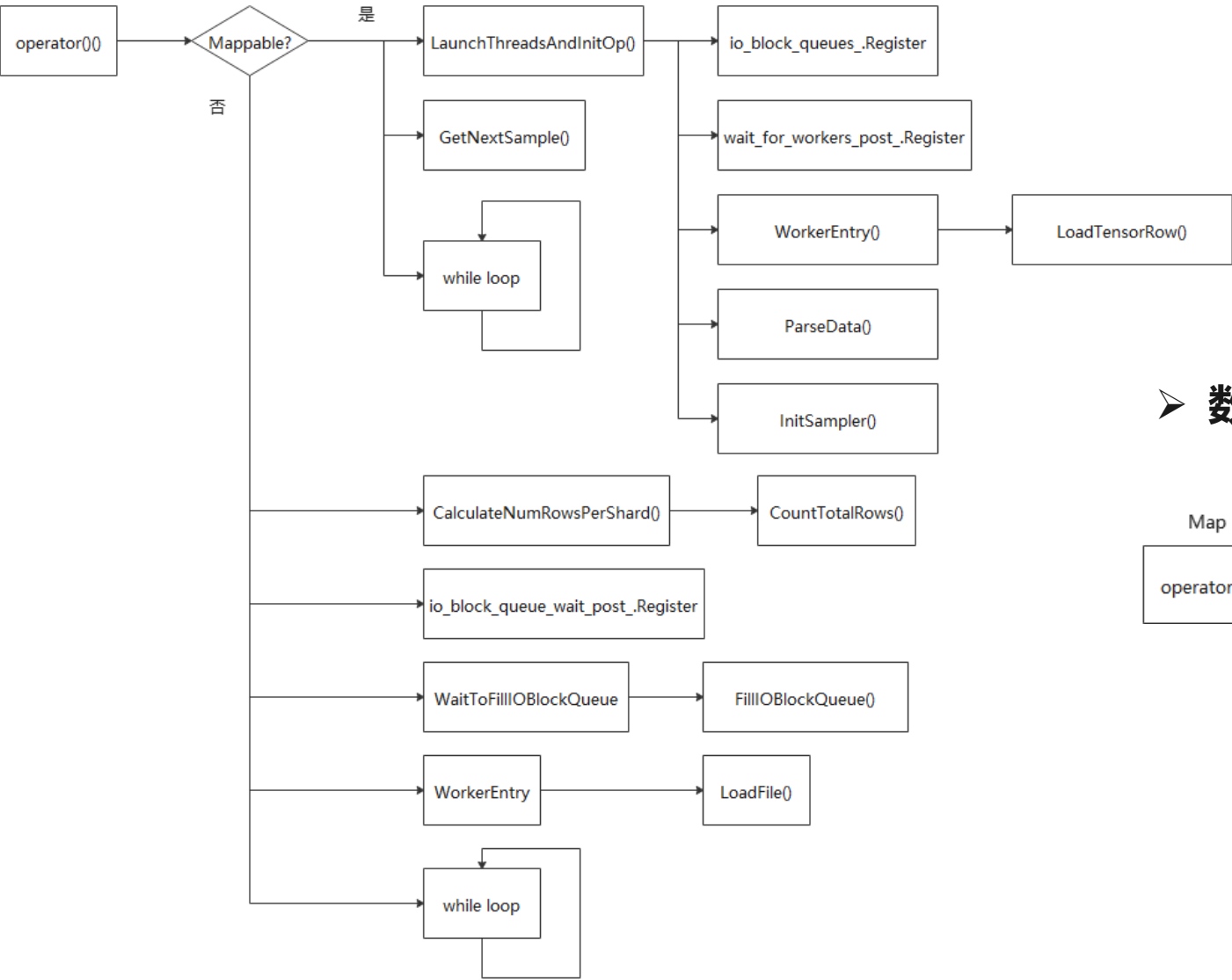
➤ 构建执行树 (pipeline)



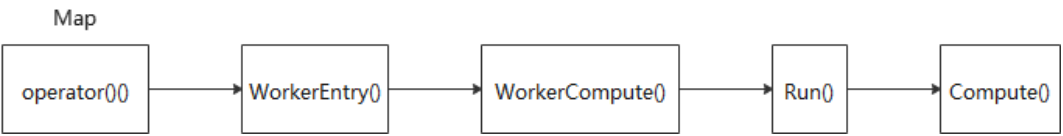
数据算子内部实现



数据集加载算子



数据增强算子



数据算子内部实现



算子接口定义

Python API

C++ API

Java API

.....

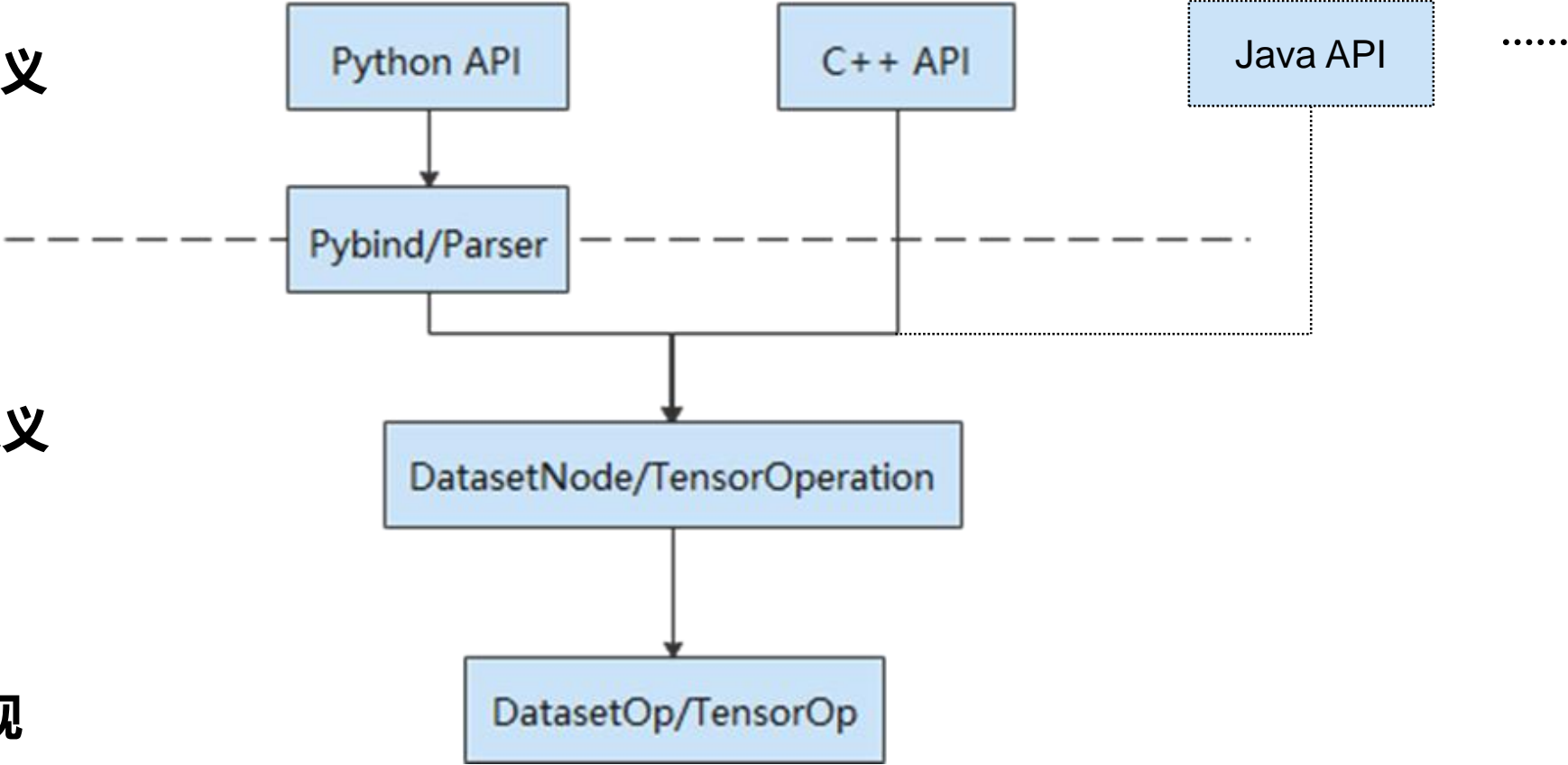
Pybind/Parser

算子IR层定义

DatasetNode/TensorOperation

算子Op实现

DatasetOp/TensorOp



数据算子内部实现

➤ 数据增强算子

• 算子Op实现

算子的主要代码逻辑在算子Op实现文件中完成，按需重写Compute、OutputShape、OutputType等函数

✓ Compute函数（必选）

```
Status ExampleOp::Compute(const std::shared_ptr<Tensor> &input, std::shared_ptr<Tensor> *output)
```

将输入Tensor处理变换后赋值给输出Tensor，如果函数过大，可将其拆解为多个工具函数统一放在data_utils.cc文件中

✓ OutputShape函数（可选）

若算子改变了Tensor的shape，则需重写此函数

✓ OutputType函数（可选）

若算子改变了Tensor中数据的Type，则需重写此函数

数据算子内部实现

➤ 数据增强算子

- **算子IR层定义**

提供算子的中间表示，以对接不同语言下的算子接口，使得各语言下算子的表现一致，需实现Build、ValidateParams等函数

✓ **Build函数**

```
std::shared_ptr<TensorOp> ExampleOperation::Build() {  
    std::shared_ptr<ExampleOp> tensor_op = std::make_shared<ExampleOp>(Arg1_,Arg2_);  
    return tensor_op;  
}
```

用于创建底层Op实现类的对象，并返回其指针

✓ **ValidateParams函数**

用于对各输入参数进行校验，判断其是否合法，并返回状态码

数据算子内部实现

➤ 数据增强算子

- **算子接口定义**

分别实现C++和Python的接口定义，创建中间表示层对象并返回

✓ C++接口

```
Example::Example(arg1, arg2) : arg1_(arg1), arg2_(arg2) {}
```

```
std::shared_ptr<TensorOperation> Example::Parse() {  
    return std::make_shared<ExampleOperation>(arg1_, arg2_);  
}
```

✓ Python接口

需先按模板编写Pybind文件，将Python与C++绑定

```
class Example(TensorOperation):  
    def __init__(self, arg1, arg2):  
        self.arg1 = arg1  
        self.arg2 = arg2  
    def parse(self):  
        return cde.ExampleOperation(self.arg1, self.arg2)
```

数据算子内部实现

➤ 数据集加载算子

• 算子Op实现

算子的主要代码逻辑在算子Op实现文件中完成，operator()函数已统一在父类实现，只需重写LaunchThreadsAndInitOp、LoadTensorRow、ComputeColMap等函数

✓ **LaunchThreadsAndInitOp函数**

Status LaunchThreadsAndInitOp()

启动多线程，执行数据文件扫描及计数、数据加载、采样器初始化等操作

✓ **LoadTensorRow函数**

Status LoadTensorRow(row_id_type row_id, TensorRow *row)

将数据和标签包装成Tensor结构

✓ **ComputeColMap函数**

构造数据集列名与索引的映射

数据算子内部实现

➤ 数据集加载算子

• 算子IR层定义

提供算子的中间表示，以对接不同语言下的算子接口，使得各语言下算子的表现一致，需实现Build、ValidateParams、GetDatasetSize等函数

✓ Build函数

```
Status ExampleNode::Build(std::vector<std::shared_ptr<DatasetOp>> *const node_ops)
```

用于创建底层Op实现类的对象，将其插入数据处理管道，并返回状态码

✓ ValidateParams函数

用于对各输入参数进行校验，判断其是否合法，并返回状态码

✓ GetDatasetSize函数

调用算子Op层函数，获取数据集样本数量

数据算子内部实现

➤ 数据集加载算子

• 算子接口定义

分别实现C++和Python的接口定义，创建中间表示层对象并返回

✓ C++接口

```
Example::Example(arg1, arg2) : {  
    auto sampler_obj = sampler ? sampler->Parse() : nullptr;  
    auto ds = std::make_shared<ExampleNode>(arg1, arg2);  
    ir_node_ = std::static_pointer_cast<DatasetNode>(ds);  
}
```

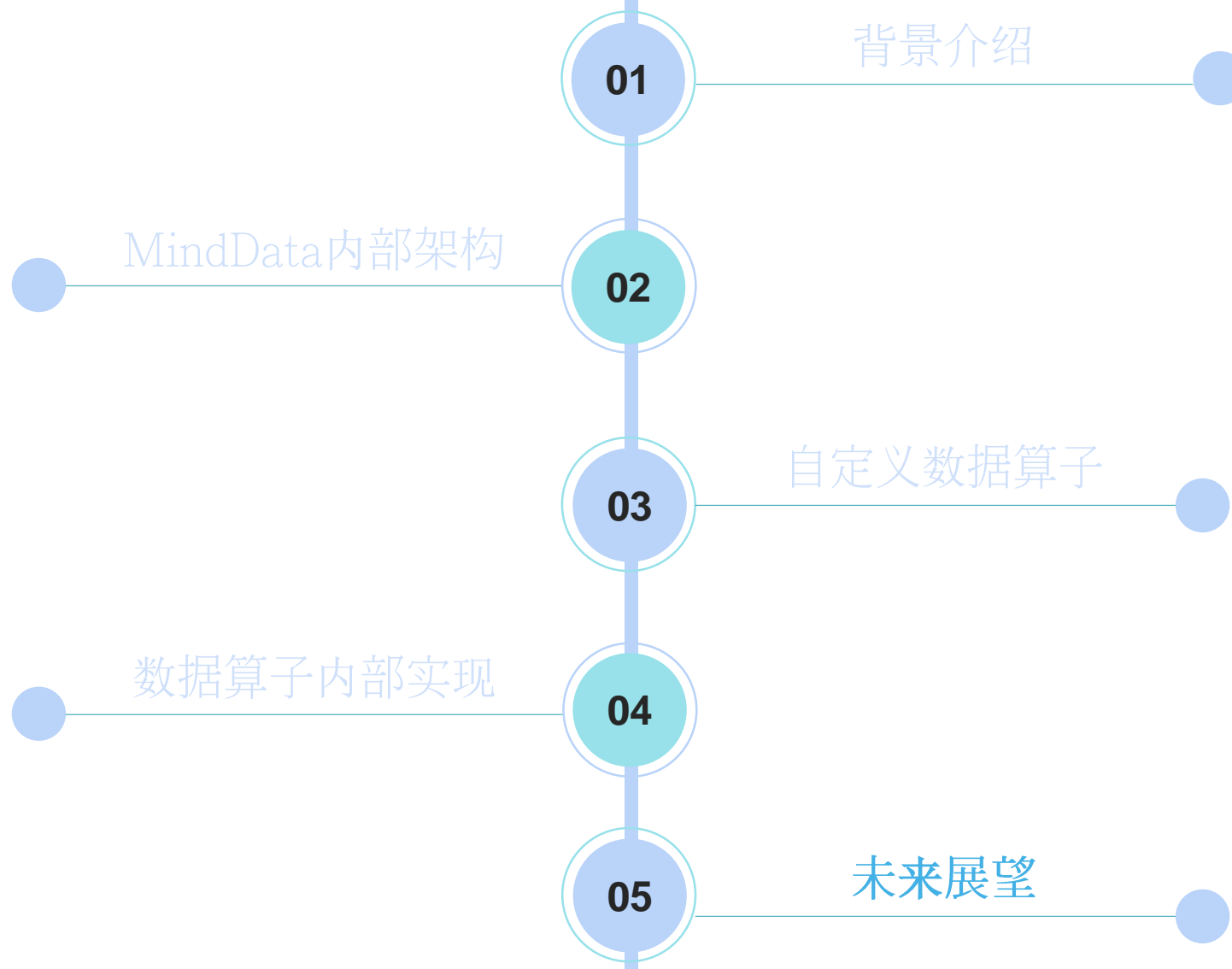
✓ Python接口

需先按模板编写Pybind文件，将Python与C++绑定

```
class ExampleDataset(MappableDataset):  
    def __init__(self, arg1, arg2):  
        self.arg1 = arg1  
        self.arg2 = arg2  
    def parse(self):  
        return cde.ExampleNode(self.arg1, self.arg2)
```

目录

contents



未来展望

- 提供Audio、Feature类数据处理算子
- 补充现有CV、NLP类数据处理算子
- 提供更多编程语言API接口
- 支持数据处理异构硬件加速

加入MindSpore SIG组，获取更多最新动态！

<https://mailweb.mindspore.cn/postorius/lists/mindspore-discuss.mindspore.cn/>

THANK YOU