python中的hook函数

1. 什么是Hook

经常会听到钩子函数(hook function)这个概念，最近在看目标检测开源框架mmdetection，里面也出现大量Hook的编程方式，那到底什么是hook？hook的作用是什么？

- what is hook 钩子hook，顾名思义，可以理解是一个挂钩，作用是有需要的时候挂一个东西上去。具体的解释是：钩子函数是把我们自己实现的hook函数在某一时刻挂接到目标挂载点上。

- hook函数的作用 举个例子，hook的概念在windows桌面软件开发很常见，特别是各种事件触发的机制; 比如C++的MFC程序中，要监听鼠标左键按下的时间，MFC提供了一个onLeftKeyDown的钩子函数。很显然，MFC框架并没有为我们实现onLeftKeyDown具体的操作，只是为我们提供一个钩子，当我们需要处理的时候，只要去重写这个函数，把我们需要操作挂载在这个钩子里，如果我们不挂载，MFC事件触发机制中执行的就是空的操作。

从上面可知

- hook函数是程序中预定义好的函数，这个函数处于原有程序流程当中（暴露一个钩子出来）
- 我们需要再在有流程中钩子定义的函数块中实现某个具体的细节，需要把我们的实现，挂接或者注册（register）到钩子里，使得hook函数对目标可用
- hook 是一种编程机制，和具体的语言没有直接的关系
- 如果从设计模式上看，hook模式是模板方法的扩展
- 钩子只有注册的时候，才会使用，所以原有程序的流程中，没有注册或挂载时，执行的是空（即没有执行任何操作）

本文用python来解释hook的实现方式，并展示在开源项目中hook的应用案例。hook函数和我们常听到另外一个名称：回调函数（callback function）功能是类似的，可以按照同种模式来理解。

2. hook实现例子

据我所知，hook函数最常使用在某种流程处理当中。 这个流程往往有很多步骤。hook函数常常挂载在这些步骤中，为增加额外的一些操作，提供灵活性。

下面举一个简单的例子，这个例子的目的是实现一个通用往队列中插入内容的功能。流程步骤有2个

- 需要再插入队列前，对数据进行筛选 `input_filter_fn`
- 插入队列 `insert_queue`

```
class ContentStash(object):
    """
    content stash for online operation
    pipeline is
    1. input_filter: filter some contents, no use to user
    2. insert_queue(redis or other broker): insert useful content to queue
    """

    def __init__(self):
        self.input_filter_fn = None
        self.broker = []

    def register_input_filter_hook(self, input_filter_fn):
        """
        register input filter function, parameter is content dict
        Args:
            input_filter_fn: input filter function

        Returns:

        """
        self.input_filter_fn = input_filter_fn

    def insert_queue(self, content):
        """
        insert content to queue
        Args:
            content: dict

        Returns:

        """
        self.broker.append(content)

    def input_pipeline(self, content, use=False):
        """
        pipeline of input for content stash
        Args:
            use: is use, defaul False
            content: dict

        Returns:

        """
        if not use:
            return

        # input filter
        if self.input_filter_fn:
            _filter = self.input_filter_fn(content)

        # insert to queue
        if not _filter:
            self.insert_queue(content)



# test
## 实现一个你所需要的钩子实现：比如如果content 包含time就过滤掉，否则插入队列
def input_filter_hook(content):
    """
    test input filter hook
    Args:
        content: dict

    Returns: None or content

    """
    if content.get('time') is None:
        return
    else:
        return content


# 原有程序
content = {'filename': 'test.jpg', 'b64_file': "#test", 'data': {"result": "cat",
"probility": 0.9}}
content_stash = ContentStash('audit', work_dir='')

# 挂上钩子函数， 可以有各种不同钩子函数的实现，但是要主要函数输入输出必须保持原有程序中一
致，比如这里是content
content_stash.register_input_filter_hook(input_filter_hook)

# 执行流程
content_stash.input_pipeline(content)
```

3. hook在开源框架中的应用

3.1 keras

在深度学习训练流程中，hook函数体现的淋漓尽致。

一个训练过程（不包括数据准备），会轮询多次训练集，每次称为一个epoch，每个epoch又分为多个batch来训练。 流程先后拆解成：

- 开始训练
- 训练一个epoch前
- 训练一个batch前
- 训练一个batch后
- 训练一个epoch后
- 评估验证集
- 结束训练

这些步骤是穿插在训练一个batch数据的过程中，这些可以理解成是钩子函数，我们可能需要在这些钩子函数中实现一些定制化的东西，比如在训练一个epoch后我们要保存下训练的模型，在结束训练时用最好的模型执行下测试集的效果等等。

keras中是通过各种回调函数来实现钩子hook功能的。这里放一个callback的父类，定制时只要继承这个父类，实现你过关注的钩子就可以了。

```
@keras_export('keras.callbacks.Callback')
class Callback(object):
  """Abstract base class used to build new callbacks.

  Attributes:
      params: Dict. Training parameters
          (eg. verbosity, batch size, number of epochs...).
      model: Instance of `keras.models.Model`.
          Reference of the model being trained.

  The `logs` dictionary that callback methods
  take as argument will contain keys for quantities relevant to
  the current batch or epoch (see method-specific docstrings).
  """

  def __init__(self):
    self.validation_data = None  # pylint: disable=g-missing-from-attributes
    self.model = None
    # Whether this Callback should only run on the chief worker in a
    # Multi-Worker setting.
    # TODO(omalleyt): Make this attr public once solution is stable.
    self._chief_worker_only = None
    self._supports_tf_logs = False

  def set_params(self, params):
    self.params = params

  def set_model(self, model):
    self.model = model

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_batch_begin(self, batch, logs=None):
    """A backwards compatibility alias for `on_train_batch_begin`."""

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_batch_end(self, batch, logs=None):
    """A backwards compatibility alias for `on_train_batch_end`."""

  @doc_controls.for_subclass_implementers
  def on_epoch_begin(self, epoch, logs=None):
    """Called at the start of an epoch.

    Subclasses should override for any actions to run. This function should only
    be called during TRAIN mode.

    Arguments:
        epoch: Integer, index of epoch.
        logs: Dict. Currently no data is passed to this argument for this method
          but that may change in the future.
    """

  @doc_controls.for_subclass_implementers
  def on_epoch_end(self, epoch, logs=None):
    """Called at the end of an epoch.
    Subclasses should override for any actions to run. This function should only
    be called during TRAIN mode.

    Arguments:
        epoch: Integer, index of epoch.
        logs: Dict, metric results for this training epoch, and for the
          validation epoch if validation is performed. Validation result keys
          are prefixed with `val_`.
    """

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_train_batch_begin(self, batch, logs=None):
    """Called at the beginning of a training batch in `fit` methods.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict, contains the return value of `model.train_step`. Typically,
          the values of the `Model`'s metrics are returned.  Example:
          `{'loss': 0.2, 'accuracy': 0.7}`.
    """
    # For backwards compatibility.
    self.on_batch_begin(batch, logs=logs)

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_train_batch_end(self, batch, logs=None):
    """Called at the end of a training batch in `fit` methods.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict. Aggregated metric results up until this batch.
    """
    # For backwards compatibility.
    self.on_batch_end(batch, logs=logs)

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_test_batch_begin(self, batch, logs=None):
    """Called at the beginning of a batch in `evaluate` methods.

    Also called at the beginning of a validation batch in the `fit`
    methods, if validation data is provided.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict, contains the return value of `model.test_step`. Typically,
          the values of the `Model`'s metrics are returned.  Example:
          `{'loss': 0.2, 'accuracy': 0.7}`.
    """

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_test_batch_end(self, batch, logs=None):
    """Called at the end of a batch in `evaluate` methods.

    Also called at the end of a validation batch in the `fit`
    methods, if validation data is provided.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict. Aggregated metric results up until this batch.
    """

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_predict_batch_begin(self, batch, logs=None):
    """Called at the beginning of a batch in `predict` methods.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict, contains the return value of `model.predict_step`,
          it typically returns a dict with a key 'outputs' containing
          the model's outputs.
    """

  @doc_controls.for_subclass_implementers
  @generic_utils.default
  def on_predict_batch_end(self, batch, logs=None):
    """Called at the end of a batch in `predict` methods.

    Subclasses should override for any actions to run.

    Arguments:
        batch: Integer, index of batch within the current epoch.
        logs: Dict. Aggregated metric results up until this batch.
    """

  @doc_controls.for_subclass_implementers
  def on_train_begin(self, logs=None):
    """Called at the beginning of training.

    Subclasses should override for any actions to run.

    Arguments:
        logs: Dict. Currently no data is passed to this argument for this method
          but that may change in the future.
    """

  @doc_controls.for_subclass_implementers
  def on_train_end(self, logs=None):
    """Called at the end of training.

    Subclasses should override for any actions to run.

    Arguments:
        logs: Dict. Currently the output of the last call to `on_epoch_end()`
          is passed to this argument for this method but that may change in
          the future.
    """

  @doc_controls.for_subclass_implementers
  def on_test_begin(self, logs=None):
    """Called at the beginning of evaluation or validation.

    Subclasses should override for any actions to run.

    Arguments:
        logs: Dict. Currently no data is passed to this argument for this method
          but that may change in the future.
    """

  @doc_controls.for_subclass_implementers
  def on_test_end(self, logs=None):
    """Called at the end of evaluation or validation.

    Subclasses should override for any actions to run.

    Arguments:
        logs: Dict. Currently the output of the last call to
          `on_test_batch_end()` is passed to this argument for this method
          but that may change in the future.
    """

  @doc_controls.for_subclass_implementers
  def on_predict_begin(self, logs=None):
    """Called at the beginning of prediction.

    Subclasses should override for any actions to run.

    Arguments:
        logs: Dict. Currently no data is passed to this argument for this method
          but that may change in the future.
    """

  @doc_controls.for_subclass_implementers
  def on_predict_end(self, logs=None):
    """Called at the end of prediction.

    Subclasses should override for any actions to run.
```

```
      Arguments:
          logs: Dict. Currently no data is passed to this argument for this method
            but that may change in the future.
      """

    def _implements_train_batch_hooks(self):
      """Determines if this Callback should be called for each train batch."""
      return (not generic_utils.is_default(self.on_batch_begin) or
              not generic_utils.is_default(self.on_batch_end) or
              not generic_utils.is_default(self.on_train_batch_begin) or
              not generic_utils.is_default(self.on_train_batch_end))
```

这些钩子的原始程序是在模型训练流程中的

> keras源码位置： tensorflow\python\keras\engine\training.py

部分摘录如下(## I am hook)：

```
# Container that configures and calls `tf.keras.Callback`s.
      if not isinstance(callbacks, callbacks_module.CallbackList):
        callbacks = callbacks_module.CallbackList(
            callbacks,
            add_history=True,
            add_progbar=verbose != 0,
            model=self,
            verbose=verbose,
            epochs=epochs,
            steps=data_handler.inferred_steps)

      ## I am hook
      callbacks.on_train_begin()
      training_logs = None
      # Handle fault-tolerance for multi-worker.
      # TODO(omalleyt): Fix the ordering issues that mean this has to
      # happen after `callbacks.on_train_begin`.
      data_handler._initial_epoch = (  # pylint: disable=protected-access
          self._maybe_load_initial_epoch_from_ckpt(initial_epoch))
      for epoch, iterator in data_handler.enumerate_epochs():
        self.reset_metrics()
        callbacks.on_epoch_begin(epoch)
        with data_handler.catch_stop_iteration():
          for step in data_handler.steps():
            with trace.Trace(
                'TraceContext',
                graph_type='train',
                epoch_num=epoch,
                step_num=step,
                batch_size=batch_size):
              ## I am hook
              callbacks.on_train_batch_begin(step)
              tmp_logs = train_function(iterator)
              if data_handler.should_sync:
                context.async_wait()
              logs = tmp_logs  # No error, now safe to assign to logs.
              end_step = step + data_handler.step_increment
              callbacks.on_train_batch_end(end_step, logs)
        epoch_logs = copy.copy(logs)

        # Run validation.

        ## I am hook
        callbacks.on_epoch_end(epoch, epoch_logs)
```

3.2 mmdetection

mmdetection是一个目标检测的开源框架，集成了许多不同的目标检测深度学习算法（pytorch版），如faster-rcnn, fpn, retianet等。里面也大量使用了hook，暴露给应用实现流程中具体部分。

详见`https://github.com/open-mmlab/mmdetection`

这里看一个训练的调用例子(摘录)（`https://github.com/open-mmlab/mmdetection/blob/5d592154cca589c5113e8aadc8798bbc73630d98/mmdet/apis/train.py`）

```
def train_detector(model,
                   dataset,
                   cfg,
                   distributed=False,
                   validate=False,
                   timestamp=None,
                   meta=None):
    logger = get_root_logger(cfg.log_level)

    # prepare data loaders

    # put model on gpus

    # build runner
    optimizer = build_optimizer(model, cfg.optimizer)
    runner = EpochBasedRunner(
        model,
        optimizer=optimizer,
        work_dir=cfg.work_dir,
        logger=logger,
        meta=meta)
    # an ugly workaround to make .log and .log.json filenames the same
    runner.timestamp = timestamp

    # fp16 setting
    # register hooks
    runner.register_training_hooks(cfg.lr_config, optimizer_config,
                                   cfg.checkpoint_config, cfg.log_config,
                                   cfg.get('momentum_config', None))
    if distributed:
        runner.register_hook(DistSamplerSeedHook())

    # register eval hooks
    if validate:
        # Support batch_size > 1 in validation
        eval_cfg = cfg.get('evaluation', {})
        eval_hook = DistEvalHook if distributed else EvalHook
        runner.register_hook(eval_hook(val_dataloader, **eval_cfg))

    # user-defined hooks
    if cfg.get('custom_hooks', None):
        custom_hooks = cfg.custom_hooks
        assert isinstance(custom_hooks, list), \
            f'custom_hooks expect list type, but got {type(custom_hooks)}'
        for hook_cfg in cfg.custom_hooks:
            assert isinstance(hook_cfg, dict), \
                'Each item in custom_hooks expects dict type, but got ' \
                f'{type(hook_cfg)}'
            hook_cfg = hook_cfg.copy()
            priority = hook_cfg.pop('priority', 'NORMAL')
            hook = build_from_cfg(hook_cfg, HOOKS)
            runner.register_hook(hook, priority=priority)
```

4. 总结

本文介绍了hook的概念和应用，并给出了python的实现细则。希望对比有帮助。总结如下：

- hook函数是流程中预定义好的一个步骤，没有实现
- 挂载或者注册时， 流程执行就会执行这个钩子函数
- 回调函数和hook函数功能上是一致的
- hook设计方式带来灵活性，如果流程中有一个步骤，你想让调用方来实现，你可以用hook函数