tf.contrib.staging.StagingArea

Contents

Class `StagingArea`

Defined in `tensorflow/python/ops/data_flow_ops.py`.

See the guide: Staging (contrib)

Class for staging inputs. No ordering guarantees.

A `StagingArea` is a TensorFlow data structure that stores tensors across multiple steps, and exposes operations that can put and get tensors.

Each `StagingArea` element is a tuple of one or more tensors, where each tuple component has a static dtype, and may have a static shape.

The capacity of a `StagingArea` may be bounded or unbounded. It supports multiple concurrent producers and consumers; and provides exactly-once delivery.

Each element of a `StagingArea` is a fixed-length tuple of tensors whose dtypes are described by `dtypes`, and whose shapes are optionally described by the `shapes` argument.

If the `shapes` argument is specified, each component of a staging area element must have the respective fixed shape. If it is unspecified, different elements may have different shapes,

It can be configured with a capacity in which case put(values) will block until space becomes available.

Similarly, it can be configured with a memory limit which will block put(values) until space is available. This is mostly useful for limiting the number of tensors on devices such as GPUs.

All get() and peek() commands block if the requested data is not present in the Staging Area.

Properties

`capacity`

The maximum number of elements of this staging area.

`dtypes`

The list of dtypes for each component of a staging area element.

`memory_limit`

The maximum number of bytes of this staging area.

`name`

The name of the staging area.

`names`

The list of names for each component of a staging area element.

`shapes`

The list of shapes for each component of a staging area element.

Methods

`__init__`

```
__init__(
    dtypes,
    shapes=None,
    names=None,
    shared_name=None,
    capacity=0,
    memory_limit=0
)
```

Constructs a staging area object.

The two optional lists, `shapes` and `names`, must be of the same length as `dtypes` if provided. The values at a given index `i` indicate the shape and name to use for the corresponding queue component in `dtypes`.

The device scope at the time of object creation determines where the storage for the `StagingArea` will reside. Calls to `put` will incur a copy to this memory space, if necessary. Tensors returned by `get` will be placed according to the device scope when `get` is called.

Args:

- `dtypes`: A list of types. The length of dtypes must equal the number of tensors in each element.
- `capacity`: (Optional.) Maximum number of elements. An integer. If zero, the Staging Area is unbounded
- `memory_limit`: (Optional.) Maximum number of bytes of all tensors in the Staging Area. An integer. If zero, the Staging Area is unbounded
- `shapes`: (Optional.) Constraints on the shapes of tensors in an element. A list of shape tuples or None. This list is the same length as dtypes. If the shape of any tensors in the element are constrained, all must be; shapes can be None if the shapes should not be constrained.
- `names`: (Optional.) If provided, the `get()` and `put()` methods will use dictionaries with these names as keys. Must be None or a list or tuple of the same length as `dtypes`.
- `shared_name`: (Optional.) A name to be used for the shared object. By passing the same name to two different python objects they will share the underlying staging area. Must be a string.

Raises:

- `ValueError`: If one of the arguments is invalid.

clear

```
clear(name=None)
```

Clears the staging area.

Args:

- `name`: A name for the operation (optional)

Returns:

The created op

get

```
get(name=None)
```

Gets one element from this staging area.

If the staging area is empty when this operation executes, it will block until there is an element to dequeue.

Note that unlike others ops that can block, like the queue Dequeue operations, this can stop other work from happening. To avoid this, the intended use is for this to be called only when there will be an element already available. One method for doing this in a training loop would be to run a `put()` call during a warmup session.run call, and then call both `get()` and `put()` in each subsequent step.

The placement of the returned tensor will be determined by the current device scope when this function is called.

Args:

- `name`: A name for the operation (optional).

Returns:

The tuple of tensors that was gotten.

peek

```
peek(
    index,
    name=None
)
```

Peeks at an element in the staging area.

If the staging area is too small to contain the element at the specified index, it will block until enough elements are inserted to complete the operation.

The placement of the returned tensor will be determined by the current device scope when this function is called.

Args:

- `index`: The index of the tensor within the staging area to look up.
- `name`: A name for the operation (optional).

Returns:

The tuple of tensors that was gotten.

put

```
put(
    values,
    name=None
)
```

Create an op that places a value into the staging area.

This operation will block if the `StagingArea` has reached its capacity.

Args:

- `values`: Tensor (or a tuple of Tensors) to place into the staging area.
- `name`: A name for the operation (optional).

Returns:

The created op.

Raises:

- `ValueError`: If the number or type of inputs don't match the staging area.

size

```
size(name=None)
```

Returns the number of elements in the staging area.

Args:

- `name`: A name for the operation (optional)

Returns:

The created op

Last updated November 2, 2017.