MaybeOwned<Tensor>

`MaybeOwned<Tensor>` is a C++ smart pointer class that dynamically encodes whether a Tensor is *owned* or *borrowed*. It is used in certain performance-sensitive situations to avoid unnecessarily incrementing a Tensor's reference count (at a small cost in overhead from the extra indirection).

> **Warning**
>
> MaybeOwned must be used with **extreme** care. Claims of (non-)ownership are not statically checked, and mistakes can cause reference undercounting and use-after-free crashes.
>
> Due to this lack of safety net, we discourage the use of MaybeOwned outside code paths that are known to be highly performance sensitive. However, if you encounter pre-existing uses of MaybeOwned in code that you want to modify, it's critical to understand how to use it correctly.

The primary use case for `MaybeOwned<Tensor>` is a function or method that dynamically chooses between returning one of its arguments (typically from a passthrough or "no-op" code path) and returning a freshly constructed Tensor. Such a function would return a `MaybeOwned<Tensor>` in both cases, the former in a "borrowed" state via a call to `MaybeOwned<Tensor>::borrowed()`, and the latter in an "owned" state via a call to `MaybeOwned<Tensor>::owned()`.

The canonical example is `Tensor`'s `expect_contiguous` method, which shortcuts and returns a borrowed self-reference when already contiguous:

```
inline c10::MaybeOwned<Tensor> Tensor::expect_contiguous(MemoryFormat memory_format) const & {
  if (is_contiguous(memory_format)) {
    return c10::MaybeOwned<Tensor>::borrowed(*this);
  } else {
    return c10::MaybeOwned<Tensor>::owned(__dispatch_contiguous(memory_format));
  }
}
```

Using the vocabulary of lifetimes, the essential safety requirement for borrowing is that a borrowed Tensor must outlive any borrowing references to it. Here, for example, we can safely borrow `*this`, but the Tensor returned by `__dispatch_contiguous()` is freshly created, and borrowing a reference would effectively leave it ownerless.

So, general rules of thumb:

- When in doubt, don't use `MaybeOwned<Tensor>` at all - in particular, prefer avoiding using it in code that doesn't use it already. New usage should only be introduced when critical (and demonstrable) performance gains result.
- When modifying or calling code that already uses `MaybeOwned<Tensor>`, remember that it's always safe to produce a `MaybeOwned<Tensor>` from a Tensor in hand via a call to `MaybeOwned<Tensor>::owned()`. This may result in an unnecessary reference count, but never in misbehavior - so it's always the safer bet, unless the lifetime of the Tensor you're looking to wrap is crystal clear.

More details and implementation code can be found at <https://github.com/pytorch/pytorch/blob/master/c10/util/MaybeOwned.h> and <https://github.com/pytorch/pytorch/blob/master/aten/src/ATen/templates/TensorBody.h>.