PEP 3148: Futures - Execute computations asynchronously

Neil Muller

October 16, 2010

The Goal

- Quoting from PEP Motivation:

  "Python currently has powerful primitives to construct multi-threaded and multi-process applications but parallelizing simple operations requires a lot of work i.e. explicitly launching processes/threads, constructing a work/results queue, and waiting for completion or some other termination condition (e.g. failure, timeout)."

- Design inspired by the java.util.concurrent package
- Allows evaluating functions asynchronously with minimal effort
- Provides support for both multi-processing and threading

Basic Structure

- Basic building block is the Future
  - State of a function that is being executed asynchronously and may not yet have finished
- Futures are managed by Executors
  - `submit(fn, *args, **args)` - creates a Future that will execute the given callable with the given arguments
  - `map(fn, *iterables, timeout=None)` - `map(fn, *iterables)` executed asynchronously.
  - `shutdown(wait=True)` - cleanup resources when pending futures are complete
- Executor is an abstract base class, needs to be overridden to provide needed functionality
  - Two implementations provides - `ThreadPoolExecutor` and `ProcessPoolExecutor`

- Futures methods:
  - cancel() a Future that hasn't started executing
  - `result(timeout=None)` - attempt to get the result of the future
    - will raise any exceptions raised by the callable
  - `add_done_callback(fn)` - call fn when the future is either completed or cancelled
    - function called with the future as the only argument
  - `exception(timeout=None)` - get any exceptions raised by the callable
    - Mainly useful so the callback function can get the exception details
  - `cancelled()`, `done()`, `running()` - state querying functions

Additional Interesting notes

- Creates the concurrent namespace
  - Plans to move various useful generic bits out of multi-processing into concurrent namespace ongoing, look likely to happen in the python 3.3 development cycle
  - Likely to be largely agnostic between threads and processes.
- BDFL for 1 PEP approach taken to accept the PEP
  - final decision delegated to Jesse Noller
  - seems to be an approach that will become more common in the future