ReadSource Documentation

Release

Kiven

November 21, 2016

Contents

[U+76EE][U+5F55]:

CHAPTER 1

Gunicorn

1.1 Where to start

- *WSGIApplication*
- *Application*
- *BaseApplication*

[U+9996][U+5148][U+4ECE]gunicorn[U+7684] setup.py [U+5F00][U+59CB][U+5427].
[U+91CC][U+9762][U+7684] entry_points [U+91CC][U+8FD9][U+6837]:

```
entry_points="""
[console_scripts]
gunicorn=gunicorn.app.wsgiapp:run
gunicorn_django=gunicorn.app.djangoapp:run
gunicorn_paster=gunicorn.app.pasterapp:run

[paste.server_runner]
main=gunicorn.app.pasterapp:paste_server
"""
```

[U+6240][U+4EE5][U+6211][U+4EEC][U+4ECE][U+6700][U+57FA][U+7840][U+7684]
wsgiapp [U+770B][U+8D77].

1.1.1 WSGIApplication

[U+5165][U+53E3][U+5C31][U+5728][U+8FD9][U+91CC]:

```
def run():
    """Here is docstring"""
    from gunicorn.app.wsgiapp import WSGIApplication
    WSGIApplication("%(prog)s [OPTIONS] [APP_MODULE]").run()
```

[U+7136][U+540E][U+6211][U+4EEC][U+53EF][U+4EE5][U+770B][U+5230]
WSGIApplication [U+8FD9][U+4E2A]class, [U+5176][U+5B9E][U+4EE3][U+7801][U+4E5F][U+4E0D][U+957F

```
class WSGIApplication(Application):
    def init(self, parser, opts, args):
        if opts.paste and opts.paste is not None:
            app_name = 'main'
            path = opts.paste
```

```
            if '#' in path:
                path, app_name = path.split('#')
            path = os.path.abspath(os.path.normpath(
                os.path.join(util.getcwd(), path)))

            if not os.path.exists(path):
                raise ConfigError("%r not found" % path)

            # paste application, load the config
            self.cfgurl = 'config:%s#%s' % (path, app_name)
            self.relpath = os.path.dirname(path)

            from .pasterapp import paste_config
            return paste_config(self.cfg, self.cfgurl, self.relpath)

        if len(args) < 1:
            parser.error("No application module specified.")

        self.cfg.set("default_proc_name", args[0])
        self.app_uri = args[0]

    def chdir(self):
        # chdir to the configured path before loading,
        # default is the current dir
        os.chdir(self.cfg.chdir)

        # add the path to sys.path
        sys.path.insert(0, self.cfg.chdir)

    def load_wsgiapp(self):
        self.chdir()

        # load the app
        return util.import_app(self.app_uri)

    def load_pasteapp(self):
        self.chdir()

        # load the paste app
        from .pasterapp import load_pasteapp
        return load_pasteapp(self.cfgurl, self.relpath, global_conf=self.cfg.paste_global_conf)

    def load(self):
        if self.cfg.paste is not None:
            return self.load_pasteapp()
        else:
            return self.load_wsgiapp()
```

[U+53EF][U+4EE5][U+770B][U+5230][U+8FD9][U+4E2A] WSGIApplication [U+7C7B][U+7EE7][U+627F][U+81EA] Application.

[U+6240][U+4EE5],[U+6211][U+4EEC][U+9700][U+8981][U+8FDB][U+5165][U+5230][U+8FD9][U+4E2A][U

1.1.2 Application

```
class Application(BaseApplication):
```

```
    def get_config_from_filename(self, filename):

        if not os.path.exists(filename):
            raise RuntimeError("%r doesn't exist" % filename)

        cfg = {
            "__builtins__": __builtins__,
            "__name__": "__config__",
            "__file__": filename,
            "__doc__": None,
            "__package__": None
        }
        try:
            execfile_(filename, cfg, cfg)
        except Exception:
            # print("Failed to read config file: %s" % filename, file=sys.stderr)
            traceback.print_exc()
            sys.stderr.flush()
            sys.exit(1)

        return cfg

    def get_config_from_module_name(self, module_name):
        return util.import_module(module_name).__dict__

    def load_config_from_module_name_or_filename(self, location):
        """
        Loads the configuration file: the file is a python file, otherwise raise an RuntimeError
        Exception or stop the process if the configuration file contains a syntax error.
        """

        if location.startswith("python:"):
            module_name = location[len("python:"):]
            cfg = self.get_config_from_module_name(module_name)
        else:
            if location.startswith("file:"):
                filename = location[len("file:"):]
            else:
                filename = location
            cfg = self.get_config_from_filename(filename)

        for k, v in cfg.items():
            # Ignore unknown names
            if k not in self.cfg.settings:
                continue
            try:
                self.cfg.set(k.lower(), v)
            except:
                # print("Invalid value for %s: %s\n" % (k, v), file=sys.stderr)
                sys.stderr.flush()
                raise

        return cfg

    def load_config_from_file(self, filename):
        return self.load_config_from_module_name_or_filename(location=filename)

    def load_config(self):
```

```
        # parse console args
        parser = self.cfg.parser()
        args = parser.parse_args()

        # optional settings from apps
        cfg = self.init(parser, args, args.args)

        # Load up the any app specific configuration
        if cfg and cfg is not None:
            for k, v in cfg.items():
                self.cfg.set(k.lower(), v)

        if args.config:
            self.load_config_from_file(args.config)
        else:
            default_config = get_default_config_file()
            if default_config is not None:
                self.load_config_from_file(default_config)

        # Lastly, update the configuration with any command line
        # settings.
        for k, v in args.__dict__.items():
            if v is None:
                continue
            if k == "args":
                continue
            self.cfg.set(k.lower(), v)

    def run(self):
        if self.cfg.check_config:
            try:
                self.load()
            except:
                msg = "\nError while loading the application:\n"
                # print(msg, file=sys.stderr)
                traceback.print_exc()
                sys.stderr.flush()
                sys.exit(1)
            sys.exit(0)

        if self.cfg.spew:
            debug.spew()

        if self.cfg.daemon:
            util.daemonize(self.cfg.enable_stdio_inheritance)

        # set python paths
        if self.cfg.pythonpath and self.cfg.pythonpath is not None:
            paths = self.cfg.pythonpath.split(",")
            for path in paths:
                pythonpath = os.path.abspath(path)
                if pythonpath not in sys.path:
                    sys.path.insert(0, pythonpath)

        super(Application, self).run()
```

[U+8FD9][U+4E2A]Application[U+7C7B][U+5219][U+7EE7][U+627F][U+4E00][U+4E2A]Base[U+7C7B]BaseApplication:

1.1.3 BaseApplication

```
class BaseApplication(object):
    """
    An application interface for configuring and loading
    the various necessities for any given web framework.
    """
    def __init__(self, usage=None, prog=None):
        self.usage = usage
        self.cfg = None
        self.callable = None
        self.prog = prog
        self.logger = None
        self.do_load_config()

    def do_load_config(self):
        """
        Loads the configuration
        """
        try:
            self.load_default_config()
            self.load_config()
        except Exception as e:
            # print("\nError: %s" % str(e), file=sys.stderr)
            sys.stderr.flush()
            sys.exit(1)

    def load_default_config(self):
        # init configuration
        self.cfg = Config(self.usage, prog=self.prog)

    def init(self, parser, opts, args):
        raise NotImplementedError

    def load(self):
        raise NotImplementedError

    def load_config(self):
        """
        This method is used to load the configuration from one or several input(s).
        Custom Command line, configuration file.
        You have to override this method in your class.
        """
        raise NotImplementedError

    def reload(self):
        self.do_load_config()
        if self.cfg.spew:
            debug.spew()

    def wsgi(self):
        if self.callable is None:
            self.callable = self.load()
        return self.callable

    def run(self):
        try:
            Arbiter(self).run()
```

```
        except RuntimeError as e:
            # print("\nError: %s\n" % e, file=sys.stderr)
            sys.stderr.flush()
            sys.exit(1)
```

[U+6240][U+4EE5][U+5F53][U+5B9E][U+4F8B][U+5316] WSGIApplication [U+7684][U+65F6][U+5019],[U+7B2C][U+4E00][U+6B65][U+505A][U+7684][U+5176][U+5B9E][U+662F] do_load_config(), [U+8FD9][U+4E2A][U+65B9][U+6CD5][U+4E2D], [U+9996][U+5148][U+52A0][U+8F7D][U+4E86][U+9ED8][U+8BA4][U+914D][U+7F6E] load_default_config(),[U+5176][U+5B9E][U+5C31][U+662F][U+5B9E][U+4F8B][U+5316][U+4E86] gunicorn.config.Config() [U+7C7B].[U+7136][U+540E][U+5219][U+662F][U+52A0][U+8F7D][U+6211][U load_config(),load_config() [U+7684][U+65F6][U+5019][U+505A][U+4E86][U+4E00][U+6B65] init() [U+7684][U+64CD][U+4F5C]. [U+8FD9][U+4E2A] init() [U+5C31][U+662F] WSGIApplication() [U+7C7B][U+91CC][U+7684] init() [U+65B9][U+6CD5]. [U+5F53][U+7136],[U+6211][U+4EEC][U+53EF][U+4EE5][U+8986][U+5199][U+8FD9][U+4E2A] init() [U+65B9][U+6CD5]. [U+7136][U+540E][U+5C31][U+662F][U+8C03][U+7528][U+4E86] run() [U+8FD9][U+4E2A][U+65B9][U+6CD5]. run() [U+65B9][U+6CD5][U+91CC], [U+5219][U+5C1D][U+8BD5][U+52A0][U+8F7D][U+6211][U+4EEC]app. util.import_app(self.app_uri) . [U+6700][U+540E][U+8C03][U+7528] BaseApplication [U+7C7B][U+7684] run() [U+65B9][U+6CD5]. [U+8FD9][U+4E2A] run() [U+65B9][U+6CD5][U+4E2D][U+5219][U+542F][U+52A8][U+4E86][U+6211][U+4EEC][U+6240][U+8C13][U+ master: Arbiter(self).run().

[U+63A5][U+4E0B][U+6765],[U+6211][U+4EEC][U+5C31][U+9700][U+8981][U+8FDB][U+5165][U+5230] gunicorn.arbiter.Arbiter [U+4E2D][U+4E00][U+63A2][U+7A76][U+7ADF][U+4E86].

1.2 Main master loop

- *run*
  - *self.start*
  - *self.manage_workers()*

arbiter.Arbiter [U+8FD9][U+4E2A][U+7C7B][U+5C31][U+662F][U+4E00][U+4E2A] *main master loop*.

1.2.1 run

[U+9996][U+5148][U+770B] run [U+8FD9][U+4E2A][U+65B9][U+6CD5]:

```
1  def run(self):
2      "Main master loop."
3      self.start()
4      util._setproctitle("master [%s]" % self.proc_name)
5
6      try:
7          self.manage_workers()
8
9          while True:
10             self.maybe_promote_master()
11
12             sig = self.SIG_QUEUE.pop(0) if len(self.SIG_QUEUE) else None
13             if sig is None:
```

```
                    self.sleep()
                    self.murder_workers()
                    self.manage_workers()
                    continue

                if sig not in self.SIG_NAMES:
                    self.log.info("Ignoring unknown signal: %s", sig)
                    continue

                signame = self.SIG_NAMES.get(sig)
                handler = getattr(self, "handle_%s" % signame, None)
                if not handler:
                    self.log.error("Unhandled signal: %s", signame)
                    continue
                self.log.info("Handling signal: %s", signame)
                handler()
                self.wakeup()
        except StopIteration:
            self.halt()
        except KeyboardInterrupt:
            self.halt()
        except HaltServer as inst:
            self.halt(reason=inst.reason, exit_status=inst.exit_status)
        except SystemExit:
            raise
        except Exception:
            self.log.info("Unhandled exception in main loop",
                          exc_info=True)
            self.stop(False)
            if self.pidfile is not None:
                self.pidfile.unlink()
            sys.exit(-1)
```

[U+8FD9][U+4E2A][U+65B9][U+6CD5][U+5C31][U+662F][U+4E00][U+4E2A][U+6B7B][U+5FAA][U+73AF],
[U+4E0D][U+65AD][U+7684][U+68C0][U+6D4B][U+548C][U+7BA1][U+7406] worker.
[U+6240][U+4EE5] master [U+5176][U+5B9E][U+6CA1][U+5E72][U+4EC0][U+4E48][U+4E8B].
[U+53EA][U+662F][U+7528][U+6765][U+7BA1][U+7406] worker [U+7684].
[U+800C] worker [U+624D][U+662F][U+771F][U+6B63][U+5E72][U+6D3B][U+7684].
[U+8FD9][U+4E2A][U+540E][U+9762][U+4F1A][U+8BE6][U+7EC6][U+8BF4][U+5230].

self.start

[U+73B0][U+5728][U+6211][U+4EEC][U+6765][U+770B] run [U+51FD][U+6570][U+7684][U+7B2C][U+4E09
self.start(), [U+8FD9][U+4E2A] start [U+5C31][U+662F][U+7528][U+6765] Initialize
the arbiter. Start listening and set pidfile if needed.

```
def start(self):
    """\
    Initialize the arbiter. Start listening and set pidfile if needed.
    """
    self.log.info("Starting gunicorn %s", __version__)

    if 'GUNICORN_PID' in os.environ:
        self.master_pid = int(os.environ.get('GUNICORN_PID'))
        self.proc_name = self.proc_name + ".2"
        self.master_name = "Master.2"

```

```
12      self.pid = os.getpid()
13      if self.cfg.pidfile is not None:
14          pidname = self.cfg.pidfile
15          if self.master_pid != 0:
16              pidname += ".2"
17          self.pidfile = Pidfile(pidname)
18          self.pidfile.create(self.pid)
19      self.cfg.on_starting(self)
20
21      self.init_signals()
22      if not self.LISTENERS:
23          self.LISTENERS = create_sockets(self.cfg, self.log)
24
25      listeners_str = ",".join([str(l) for l in self.LISTENERS])
26      self.log.debug("Arbiter booted")
27      self.log.info("Listening at: %s (%s)", listeners_str, self.pid)
28      self.log.info("Using worker: %s", self.cfg.worker_class_str)
29
30      # check worker class requirements
31      if hasattr(self.worker_class, "check_config"):
32          self.worker_class.check_config(self.cfg, self.log)
33
34      self.cfg.when_ready(self)
```

[U+8FD9][U+4E2A][U+51FD][U+6570][U+4E3B][U+8981][U+662F][U+770B][U+6700][U+540E][U+4E00][U+
[U+8FD9][U+4E2A][U+65F6][U+5019][U+662F]master[U+521D][U+59CB][U+5316][U+540E],
[U+6267][U+884C] when_ready [U+8FD9][U+4E2A][U+94A9][U+5B50].
[U+8FD9][U+4E2A][U+94A9][U+5B50],[U+6211][U+4EEC][U+53EF][U+4EE5][U+505A][U+4E00][U+4E9B][U
[U+7ED9][U+4E2A][U+4F8B][U+5B50]:

```
# this is your application file

from gunicron.app.base import Application

def when_ready(server):
    """when gunicorn master start, do something yourself"""


class MyApp(Application):
    def install_hooks(self):
        self.cfg.set('when_ready', when_ready)
```

self.manage_workers()

```
def manage_workers(self):
    """\
    Maintain the number of workers by spawning or killing
    as required.
    """
    if len(self.WORKERS.keys()) < self.num_workers:
        self.spawn_workers()

    workers = self.WORKERS.items()
    workers = sorted(workers, key=lambda w: w[1].age)
    while len(workers) > self.num_workers:
        (pid, _) = workers.pop(0)
        self.kill_worker(pid, signal.SIGTERM)
```

```
    active_worker_count = len(workers)
    if self._last_logged_active_worker_count != active_worker_count:
        self._last_logged_active_worker_count = active_worker_count
        self.log.debug("{0} workers".format(active_worker_count),
                       extra={"metric": "gunicorn.workers",
                              "value": active_worker_count,
                              "mtype": "gauge"})
```

[U+8FD9][U+4E2A][U+51FD][U+6570][U+5C31][U+662F][U+7528][U+6765][U+4FDD][U+8BC1] worker [U+6570][U+91CF], [U+4E3B][U+8981][U+662F][U+901A][U+8FC7] spawn_workers() [U+548C] kill_worker [U+8FD9][U+4E24][U+4E2A][U+51FD][U+6570].

1.3 Workers

- *spawn_worker*
- *worker*

1.3.1 spawn_worker

```
def spawn_worker(self):
    self.worker_age += 1
    worker = self.worker_class(self.worker_age, self.pid, self.LISTENERS,
                               self.app, self.timeout / 2.0,
                               self.cfg, self.log)
    self.cfg.pre_fork(self, worker)
    pid = os.fork()
    if pid != 0:
        self.WORKERS[pid] = worker
        return pid

    # Process Child
    worker_pid = os.getpid()
    try:
        util._setproctitle("worker [%s]" % self.proc_name)
        self.log.info("Booting worker with pid: %s", worker_pid)
        if self.cfg.reuseport:
            worker.sockets = create_sockets(self.cfg, self.log)
            listeners_str = ",".join([str(l) for l in worker.sockets])
            self.log.info("Listening at: %s (%s) using reuseport",
                          listeners_str,
                          worker_pid)
        self.cfg.post_fork(self, worker)
        worker.init_process()
        sys.exit(0)
    except SystemExit:
        raise
    except AppImportError as e:
        self.log.debug("Exception while loading the application: \n%s",
                traceback.format_exc())
        print("%s" % e, file=sys.stderr)
        sys.stderr.flush()
        sys.exit(self.APP_LOAD_ERROR)
```

```
    except:
        self.log.exception("Exception in worker process:\n%s",
                traceback.format_exc())
        if not worker.booted:
            sys.exit(self.WORKER_BOOT_ERROR)
        sys.exit(-1)
    finally:
        self.log.info("Worker exiting (pid: %s)", worker_pid)
        try:
            worker.tmp.close()
            self.cfg.worker_exit(self, worker)
        except:
            pass
```

[U+9996][U+5148],[U+5B9E][U+4F8B][U+5316] worker_class,[U+6267][U+884C] pre_fork
[U+8FD9][U+4E2A][U+94A9][U+5B50],[U+7136][U+540E][U+5C31][U+662F]fork[U+5B50][U+8FDB][U+7A0B
[U+5728][U+7236][U+8FDB][U+7A0B][U+4E2D][U+8BB0][U+5F55] worker[U+8FDB][U+7A0B][U+7684]pid[U+
self.WORKERS[pid] = worker. [U+5728][U+5B50][U+8FDB][U+7A0B][U+4E2D],
[U+8BBE][U+7F6E][U+8FDB][U+7A0B][U+540D][U+8BB0][U+5F55]logger[U+7B49],
[U+7136][U+540E][U+6267][U+884C] post_fork [U+94A9][U+5B50],
[U+6700][U+540E][U+8FD9][U+4E2A][U+5B50][U+8FDB][U+7A0B][U+8FDB][U+5165][U+81EA][U+5DF1][U-
worker.init_process.

1.3.2 worker

```
class Worker(object):

    SIGNALS = [getattr(signal, "SIG%s" % x)
            for x in "ABRT HUP QUIT INT TERM USR1 USR2 WINCH CHLD".split()]

    PIPE = []

    def __init__(self, age, ppid, sockets, app, timeout, cfg, log):
        """\
        This is called pre-fork so it shouldn't do anything to the
        current process. If there's a need to make process wide
        changes you'll want to do that in ``self.init_process()``.
        """
        self.age = age
        self.ppid = ppid
        self.sockets = sockets
        self.app = app
        self.timeout = timeout
        self.cfg = cfg
        self.booted = False
        self.aborted = False
        self.reloader = None

        self.nr = 0
        jitter = randint(0, cfg.max_requests_jitter)
        self.max_requests = cfg.max_requests + jitter or MAXSIZE
        self.alive = True
        self.log = log
        self.tmp = WorkerTmp(cfg)

    def __str__(self):
```

```
32          return "<Worker %s>" % self.pid
33
34      @property
35      def pid(self):
36          return os.getpid()
37
38      def notify(self):
39          """\
40          Your worker subclass must arrange to have this method called
41          once every ``self.timeout`` seconds. If you fail in accomplishing
42          this task, the master process will murder your workers.
43          """
44          self.tmp.notify()
45
46      def run(self):
47          """\
48          This is the mainloop of a worker process. You should override
49          this method in a subclass to provide the intended behaviour
50          for your particular evil schemes.
51          """
52          raise NotImplementedError()
53
54      def init_process(self):
55          """\
56          If you override this method in a subclass, the last statement
57          in the function should be to call this method with
58          super(MyWorkerClass, self).init_process() so that the ``run()``
59          loop is initiated.
60          """
61
62          # start the reloader
63          if self.cfg.reload:
64              def changed(fname):
65                  self.log.info("Worker reloading: %s modified", fname)
66                  os.kill(self.pid, signal.SIGQUIT)
67              self.reloader = Reloader(callback=changed).start()
68
69          # set environment' variables
70          if self.cfg.env:
71              for k, v in self.cfg.env.items():
72                  os.environ[k] = v
73
74          util.set_owner_process(self.cfg.uid, self.cfg.gid)
75
76          # Reseed the random number generator
77          util.seed()
78
79          # For waking ourselves up
80          self.PIPE = os.pipe()
81          for p in self.PIPE:
82              util.set_non_blocking(p)
83              util.close_on_exec(p)
84
85          # Prevent fd inheritance
86          [util.close_on_exec(s) for s in self.sockets]
87          util.close_on_exec(self.tmp.fileno())
88
89          self.log.close_on_exec()
```

```
90
91          self.init_signals()
92
93          self.wsgi = self.app.wsgi()
94
95          self.cfg.post_worker_init(self)
96
97          # Enter main run loop
98          self.booted = True
99          self.run()
100
101     def init_signals(self):
102         # reset signaling
103         [signal.signal(s, signal.SIG_DFL) for s in self.SIGNALS]
104         # init new signaling
105         signal.signal(signal.SIGQUIT, self.handle_quit)
106         signal.signal(signal.SIGTERM, self.handle_exit)
107         signal.signal(signal.SIGINT, self.handle_quit)
108         signal.signal(signal.SIGWINCH, self.handle_winch)
109         signal.signal(signal.SIGUSR1, self.handle_usr1)
110         signal.signal(signal.SIGABRT, self.handle_abort)
111
112         # Don't let SIGTERM and SIGUSR1 disturb active requests
113         # by interrupting system calls
114         if hasattr(signal, 'siginterrupt'):  # python >= 2.6
115             signal.siginterrupt(signal.SIGTERM, False)
116             signal.siginterrupt(signal.SIGUSR1, False)
117
118     def handle_usr1(self, sig, frame):
119         self.log.reopen_files()
120
121     def handle_exit(self, sig, frame):
122         self.alive = False
123
124     def handle_quit(self, sig, frame):
125         self.alive = False
126         # worker_int callback
127         self.cfg.worker_int(self)
128         time.sleep(0.1)
129         sys.exit(0)
130
131     def handle_abort(self, sig, frame):
132         self.alive = False
133         self.cfg.worker_abort(self)
134         sys.exit(1)
135
136     def handle_error(self, req, client, addr, exc):
137         request_start = datetime.now()
138         addr = addr or ('', -1)  # unix socket case
139         if isinstance(exc, (InvalidRequestLine, InvalidRequestMethod,
140                 InvalidHTTPVersion, InvalidHeader, InvalidHeaderName,
141                 LimitRequestLine, LimitRequestHeaders,
142                 InvalidProxyLine, ForbiddenProxyRequest)):
143
144             status_int = 400
145             reason = "Bad Request"
146
147             if isinstance(exc, InvalidRequestLine):
```

```
                    mesg = "Invalid Request Line '%s'" % str(exc)
                elif isinstance(exc, InvalidRequestMethod):
                    mesg = "Invalid Method '%s'" % str(exc)
                elif isinstance(exc, InvalidHTTPVersion):
                    mesg = "Invalid HTTP Version '%s'" % str(exc)
                elif isinstance(exc, (InvalidHeaderName, InvalidHeader,)):
                    mesg = "%s" % str(exc)
                    if not req and hasattr(exc, "req"):
                        req = exc.req  # for access log
                elif isinstance(exc, LimitRequestLine):
                    mesg = "%s" % str(exc)
                elif isinstance(exc, LimitRequestHeaders):
                    mesg = "Error parsing headers: '%s'" % str(exc)
                elif isinstance(exc, InvalidProxyLine):
                    mesg = "'%s'" % str(exc)
                elif isinstance(exc, ForbiddenProxyRequest):
                    reason = "Forbidden"
                    mesg = "Request forbidden"
                    status_int = 403

                msg = "Invalid request from ip={ip}: {error}"
                self.log.debug(msg.format(ip=addr[0], error=str(exc)))
            else:
                self.log.exception("Error handling request")

                status_int = 500
                reason = "Internal Server Error"
                mesg = ""

            if req is not None:
                request_time = datetime.now() - request_start
                environ = default_environ(req, client, self.cfg)
                environ['REMOTE_ADDR'] = addr[0]
                environ['REMOTE_PORT'] = str(addr[1])
                resp = Response(req, client, self.cfg)
                resp.status = "%s %s" % (status_int, reason)
                resp.response_length = len(mesg)
                self.log.access(resp, req, environ, request_time)

            try:
                util.write_error(client, status_int, reason, mesg)
            except:
                self.log.debug("Failed to send error message.")

        def handle_winch(self, sig, fname):
            # Ignore SIGWINCH in worker. Fixes a crash on OpenBSD.
            return
```

[U+8FD9][U+4E2A][U+7C7B][U+4E2D],[U+4E3B][U+8981][U+662F][U+521D][U+59CB][U+5316][U+4E86][U

```
def init_signals(self):
    # reset signaling
    [signal.signal(s, signal.SIG_DFL) for s in self.SIGNALS]
    # init new signaling
    signal.signal(signal.SIGQUIT, self.handle_quit)
    signal.signal(signal.SIGTERM, self.handle_exit)
    signal.signal(signal.SIGINT, self.handle_quit)
    signal.signal(signal.SIGWINCH, self.handle_winch)
    signal.signal(signal.SIGUSR1, self.handle_usr1)
```

```
    signal.signal(signal.SIGABRT, self.handle_abort)

    # Don't let SIGTERM and SIGUSR1 disturb active requests
    # by interrupting system calls
    if hasattr(signal, 'siginterrupt'):  # python >= 2.6
        signal.siginterrupt(signal.SIGTERM, False)
        signal.siginterrupt(signal.SIGUSR1, False)
```

[U+4E3B][U+8981][U+6CE8][U+610F][U+7684][U+662F], signal.SIGTERN [U+8FD9][U+4E2A][U+4FE1][U+53F7],[U+5B83][U+7684][U+5904][U+7406][U+51FD][U+6570][U+662F] handle_exit.

```
def handle_exit(self, sig, frame):
    self.alive = False
```

[U+53EA][U+662F][U+5C06] worker [U+7684][U+72B6][U+6001][U+6807][U+5FD7][U+4E3A] self.alive = False. [U+8FD9][U+5C31][U+4F1A][U+4F7F][U+5F97]worker[U+80FD][U+591F][U+5728][U+ [U+5728] init_process [U+7684][U+6700][U+540E], [U+8C03][U+7528] self.run() [U+51FD][U+6570][U+8FD0][U+884C]worker[U+8FDB][U+7A0B]. self.run() [U+5177][U+4F53][U+7684][U+5B9E][U+73B0][U+9700][U+8981][U+770B] worker[U+7684][U+7C7B][U+578B] self.worker_class. [U+652F][U+6301][U+7684][U+7C7B][U+578B][U+5 gunicorn.workers.__init__.SUPPORTED_WORKERS [U+91CC].

CHAPTER 2

Redis-py

CHAPTER 3

Gevent

CHAPTER 4

Flask

1. config[U+539F][U+7406]
2. [U+8DEF][U+7531][U+539F][U+7406]
3. Wsgi[U+63A5][U+53E3][U+8C03][U+7528]
4. [U+7406][U+89E3]session
5. [U+7406][U+89E3]threading.local
6. [U+7406][U+89E3]flask[U+81EA][U+5DF1][U+5C01][U+88C5][U+7684]thread local
7. [U+7406][U+89E3]g[U+548C]request
8. [U+7406][U+89E3]app context[U+548C]request context

CHAPTER 5

werkzeug

CHAPTER 6

Celery

CHAPTER 7

Kazoo