03 – Pyro

- Python Remote Objects
  - Pyro3 - https://pythonhosted.org/Pyro/
  - Pyro4 - https://pythonhosted.org/Pyro4/
- Distributed Object Technology
  - RMI
  - Mobile code
- 100% pure Python
- Naming Service
- Mobile objects
- Exceptions transports

Overview

- Server
  - Write a module 'server'
    - containing a class 'serverclass'
  - Create one or more instances of the serverclass',
    - registers them with the Pyro Name Server.
- Client
  - Queries the Name Server for the location of those objects.
    - returns Pyro URI (Universal Resource Identifier) for them.
  - Create proxies for the remote objects.
    - Proxy mimics the real serverClass',
  - Invoke methods on the remote objects.
    - The proxy will forward the method invocations and return the results, just as if it was the local object itself.

Server

- Implement a class
  - To be accessed remotely
  - methods+atributes
- Make it “remotable”
  - Pyro3
    - Make it a subclass of Pyro.core.ObjBase
    - Derive a new class
      - Subclass of Pyro.core.ObjBase
  - Pyro4
    - Expose methods: @Pyro4.expose
    - Create a new “exposed” class:
      - ExposedClass = Pyro4.expose(SomeClassFromLibrary)

Derive a class (PYRO3)

```
class remoteClass(Pyro.core.ObjBase, origClass):
      def __init__(self):
         Pyro.core.ObjBase.__init__(self)
         origClass.__init__(self)
      …
```

- import Pyro.core
- Make the new class sub class of
  - Pyro.core.ObjBase and origClass
- Call the constructors of the super-classes

Expose a class (PYRO4)

```
import Pyro4
class PyroService(object):
  value = 42              # not exposed
  def __dunder__(self):# not exposed
    pass
  @Pyro4.expose
  def get_value(self):   # exposed
    return self.value
  @Pyro4.expose
  @property
  def attr(self):         #exposed as
    return self.value   #remote attr
  @Pyro4.expose
  @attr.setter
  def attr(self, value): # exposed as
    self.value = value   #writable attr
```

```
import Pyro4
@Pyro4.expose
class PyroService(object):
    def normal_method(self, args):
        result = do_calculation(args)
        return result
    @Pyro4.oneway
    def oneway_method(self, args):
        result = do_calculation(args)
```

```
from thirdparty_library import SomeClass
import Pyro4

# expose the class from the library
using @expose as wrapper function:
ExposedClass = Pyro4.expose(SomeClass)
```

Start the server

- PYRO3
- Initialize Pyro3
  - Pyro.core.initServer()
- Start daemon
  - daemon = Pyro.core.Daemon()
- Create the object
  - obj = remoteClass()
- Make object available
  - uri=daemon.connect(obj,"objName")
- Print URI
- Start request loop
  - daemon.requestLoop()

- Pyro4
-
  -
- Start daemon PYRO4
  - daemon = Pyro4.Daemon()
- Create the object
  - obj = exposedClass()
- Make object available
  - uri=daemon.register(obj,"objName")
- Print URI
- Start request loop
  - daemon.requestLoop()

Client

- PYRO3
  - Initialize Pyro
    - Pyro.core.initClient()
  - Get URI
  - Get a proxy for the remote object
    - obj = Pyro.core.getProxyForURI(URI)
    - obj = Pyro.core.getAttrProxyForURI(URI)
  - Call methods
  - Access attributes

- PYRO4
  - Get URI
  - Get Proxy for the remote object
    - obj = Pyro4.Proxy(uri)
  - Call Methods
  - Access attributes

Naming services

- URI is not user friendly
  - PYRO://146.193.41.15:7766/92c1290f512e20e1b13888fdd504a238d5
  - PYRO:addServer@localhost:51989
- Objects can be scattered on the network
- Ns should handle translations
  - Text name → URI
- Clients:
  - Pyro-xnsc / pyro-nsc
  - pyro4-nsc
  - Programms acessinf remote objects

NS location (Pyro3)

- Server
  - pyro-ns pyro4-ns
- Lan broadcast
  - locator = Pyro.naming.NameServerLocator()
  - ns = locator.getNS()
- Explicit location
  - locator = Pyro.naming.NameServerLocator()
  - ns = locator.getNS(host='hostname', port=7777)

NS location (Pyro4)

- Server
  - pyro4-ns
- Lan broadcast
  - ns=Pyro4.locateNS()
- Explicit location
  - ns=Pyro4.locateNS(host='hostname', port=7777)

Object location

- PYRO3
- Server
  - Register objects
    - daemon.useNameServer(ns)
    - uri=daemon.connect(obj,"objName")
- Client
  - Find objects
    - URI=ns.resolve('objName')
    - remExec = Pyro.core.getAttrProxyForURI(URI)

- PYRO4
- Server
  - uri=daemon.register(obj,"addServer")
  - ns.register(name, uri)
- Client
  - uri=nameserver.lookup(objName)
  - obj = Pyro4.Proxy(uri)

Mobile code (only pyro3)

- Allows the transfer of new classes/objects
  - To/from the server
- Requirement
  - Declare classes/functions on separate module
  - Import using **import module**
  - Access using full qualifier name (module.class)

One way calls

- PYRO3
  - Define asynchronous methods
    - obj._setOneway(method)

- Pyro4

```
@Pyro4.oneway
 def oneway_method(self, args):
        result =
          do_long_calculation(args)
```

-

More information

- Pyro3
  - http://pythonhosted.org/Pyro/
- Pyro4
  - https://pythonhosted.org/Pyro4
  - https://pythonhosted.org/Pyro4/servercode.html
  - https://pythonhosted.org/Pyro4/clientcode.html
  - https://pythonhosted.org/Pyro4/nameserver.html