美吉生物 Majorbio

# biocluster开发流程规范

——何胜 2016.7.18

上海美吉生物医药科技有限公司
Shanghai Majorbio Bio-Pharm Technology Co., Ltd.

# 目录

# tool&agent开发和测试—软件安装

- ~/app/ # app目录下用于安装所有软件
- bioinfo # 生信软件安装目录
- database # 数据库存放目录
- gcc # gcc编译器目录
- install_packages # 软件安装包存放目录
- library # 库文件目录
- program # 其他软件程序安装目录(包括python、perl、R、java等)
- 软件安装流程：
  - 安装包下载到install_packages,当前路径解压
  - 咨询邱萍(余果)安装到特定目录下编译安装
  - 不要添加任何环境变量
- python包的安装：
  - 尽量使用源码编译，安装包也放在install_packages

# tool&agent开发和测试—软件安装

常见软件/包的安装方式：

- 普通软件
  - cd 到软件目录，请务必仔细阅读README文件，很多情况不遵循下面的安装方式
  - ./configure -h  # 运行配置程序，查看所有可配置选项
  - ./configure --prefix=/to/your/target/dir  # 配置安装时的目的路径等等，检查安装可行性(有时需要在目的文件夹运行软件安装包中的configure程序)，生成make文件
  - make  # 编译程序，生成可执行文件
  - make install  # 安装软件程序到目标目录
- python包的安装：
  1. 在源码安装包解压后，在目录下使用python set_up.py install进行安装，安装到当前python的第三方包处
  2. 直接使用pip install 包名，自动安装完

# tool&agent开发和测试—软件安装

常见软件/包的安装方式：

- 普通软件
  - cd 到软件目录，请务必仔细阅读README文件，很多情况不遵循下面的安装方式
  - ./configure -h  # 运行配置程序，查看所有可配置选项
  - ./configure --prefix=/to/your/target/dir  # 配置安装时的目的路径等等，检查安装可行性(有时需要在目的文件夹运行软件安装包中的configure程序)，生成make文件
  - make  # 编译程序，生成可执行文件
  - make install  # 安装软件程序到目标目录
- python包的安装：
  1. 在源码安装包解压后，在目录下使用python set_up.py install进行安装，安装到当前python的第三方包处
  2. 直接使用pip install 包名，自动安装完

# tool&agent开发和测试—软件安装

- perl包安装：
  - cd到包解压目录，看说明，一般情况如下
  - perl Makefile.PL
  - make test　# 可选
  - make install
  - 或者直接使用cpan –i 包名，自动安装
- R包的安装：
  - R CMD INSTALL --byte-compile(可选) R模块的压缩包
  - 或者，进入R的控制台：
    - install.packages(“包名称", dependencies=TRUE)
    - install.packages(“安装压缩包", repos=NULL, type="source")

# tool&agent开发和测试—files

只有当文件会作为tool的输入文件时需要写file类型
命名：文件名:blast_xml.py>>类名:BlastXmlFile

```python
# -*- coding: utf-8 -*-
# __author__ = 'shenghe'
from biocluster.iofile import File
from biocluster.core.exceptions import FileError
from Bio.Blast import NCBIXML


class BlastXmlFile(File):
    """
    定义blast+比对输出类型为5结果文件的xml格式， 测试blast+为2.3.0版本
    """

    def __init__(self):
        super(BlastXmlFile, self).__init__()

    def get_info(self):
        """
        获取文件属性
        """
        super(BlastXmlFile, self).get_info()
        blast_info = self.get_xml_info()
        self.set_property('query_num', blast_info[0])
        self.set_property('hit_num', blast_info[1])
        self.set_property('hsp_num', blast_info[2])

    def get_xml_info(self):
        """
        获取blast结果xml的信息

        :return
        """
```

继承自File类

重写get_info

设定prop

```python
    def check(self):
        """
        检测文件是否满足要求，发生错误时应该触发FileError异常
        """
        if super(BlastXmlFile, self).check():
            # 父类check方法检查文件路径是否设置，文件是否存在，文件是否为空
            blastxml = NCBIXML.parse(open(self.path))
            try:
                blastxml.next()
            except ValueError:
                raise FileError('传入文件类型不正确，无法解析，请检查文件是否正确，或者生成文
            except Exception as e:
                raise FileError('未知原因导致blastxml文件解析失败:{}'.format(e))
            return True
```

重写 check方法

```python
    def convert2table(self, outfile):
        """调用packages中的xml2table方法来转换到table格式"""
        from mbio.packages.align.blast.xml2table import xml2table
        xml2table(self.path, outfile)
```

# tool&agent开发和测试—files—测试

测试：本地可直接测试基本功能

```
if __name__ == '__main__':  # for test
    a = BlastXmlFile()
    # a.set_path('C:\\Users\\sheng.he.MAJORBIO\\Desktop\\annotation\\annotation\\NR\\transcript.fa_vs_nr.blasttable.xls')
    a.set_path('C:\\Users\\sheng.he.MAJORBIO\\Desktop\\annotation\\annotation\\NR\\transcript.fa_vs_nr.xml')
    a.check()
    a.get_info()
    a.convert2table('C:\\Users\\sheng.he.MAJORBIO\\Desktop\\test.xls')
```

# tool&agent开发和测试—packages

使用相对自由的功能函数

1. tool中的实际计算函数，必须占用较少的资源
2. tool/module中常被公用的处理功能函数

注意：

- packages如果消耗资源很多，尽量写成脚本文件放在软件安装处
- 不要使用绝对路径，config对象提供app目录的路径
- 需要使用r模板之类的文件(当python需要执行大段R代码或者其他语言的代码，而该代码中需要python处理数据信息时，务必使用模板库，例如mako，保证代码整洁可读)，使用os.path.realpath(__file__)获取当前文件夹路径，同样不要使用绝对路径
- packages作为相对自由的函数，可以在python控制台中直接测试，import函数对象，测试函数功能

# tool&agent开发和测试—tool&agent开发—agent

美吉生物 Majorbio

```python
class BlastAgent(Agent):
    """
    ncbi blast+ 2.3.0  注意：在outfmt为6时不按照ncbi格式生成table，而是按照特殊的表头类型，参见packages.align.blast.xml2table
    version 1.0
    author: shenghe
    last_modify: 2016.6.15
    """
    def __init__(self, parent):
        super(BlastAgent, self).__init__(parent)
        self._fasta_type = {'Protein': 'prot', 'DNA': 'nucl'}
        self._blast_type = {'nucl': {'nucl': ['blastn', 'tblastn'],
                                     'prot': ['blastx']},
                            'prot': {'nucl': [],
                                     'prot': ['blastp']}}
        self._database_type = {'nt': 'nucl', 'nr': 'prot', 'kegg': 'prot', 'swissprot': 'prot', 'string': 'prot'}
        options = [
            {"name": "query", "type": "infile", "format": "sequence.fasta"},  # 输入文件
            {"name": "query_type", "type": "string"},  # 输入的查询序列的格式，为nucl或者prot
            {"name": "database", "type": "string", "default": "nr"},
            # 比对数据库 nt nr string swissprot kegg customer_mode
            {"name": "outfmt", "type": "int", "default": 5},  # 输出格式，数字遵从blast+
            {"name": "blast", "type": "string"},  # 设定blast程序有blastn，blastx，blastp，tblastn，此处需要严格警告使用者必须选择正确的比对程序
            {"name": "reference", "type": "infile", "format": "sequence.fasta"},  # 参考序列  选择customer时启用
            {"name": "reference_type", "type": "string"},  # 参考序列(库)的类型  为nucl或者prot
            {"name": "evalue", "type": "float", "default": 1e-5},  # evalue值
            {"name": "num_threads", "type": "int", "default": 10},  # cpu数
            {"name": "num_alignment", "type": "int", "default": 500},  # 序列比对最大输出条数，默认500
            {"name": "outxml", "type": "outfile", "format": "align.blast.blast_xml"},  # 输出格式为6时输出
            {"name": "outtable", "type": "outfile", "format": "align.blast.blast_table"},  # 输出格式为5时输出
            # 当输出格式为非5，6时，只产生文件不作为outfile
            ]
        self.add_option(options)
        self.step.add_steps('blast')
        self.on('start', self.step_start)
        self.on('end', self.step_end)
```

```python
    def step_start(self):
        self.step.blast.start()
        self.step.update()

    def step_end(self):
        self.step.blast.finish()
        self.step.update()
```

# tool&agent开发和测试—tool&agent开发—agent

- 目录与命名：tool需要建立或放在相对应的目录，文件名称小写，下划线分割单词，Agent/Tool的命名使用驼峰命名法，单词首字母大写加上父类名称，例如：distance_calc.py >> DistanceCalcAgent/Tool
- add_option：option为一个列表，包含元素为字典，字典的键为name、type、default(infile/outfile之外可有)、format(infile/outfile类型独有)
  - type：为字符串类型。有int，float，string，infile，outfile
  - format：files目录下的路径，可以为多个不同的文件类型，用逗号分隔，例如：align.blast.blast_table, align.blast.blast_xml
- step：tool运行的步骤状态，通过add_steps的方式添加，可同时添加多个，具体步骤(step)对象有start和finish的方法，用于添加步骤信息，主步骤的update方法用于发送前端步骤信息。
- on：使用绑定，第一个参数为事件名称，第二个为绑定函数对象，第三个为函数参数，函数参数的使用不同于一般参数，后续细讲

# tool&agent开发和测试—tool&agent开发—agent

美吉生物 Majorbio

```
def check_options(self):
    if not self.option("query").is_set:
        raise OptionError("必须设置参数query")
    if self.option('query_type') not in ['nucl', 'prot']:
        raise OptionError('query_type查询序列的类型为nucl(核酸)或者prot(蛋白):{}'.format(self.option('query_type')))
    else:
        if self._fasta_type[self.option('query').prop['seq_type']] != self.option('query_type'):
            raise OptionError(
                '文件检查发现查询序列为:{}, 而说明的文件类型为:{}'.format(
                    self._fasta_type[self.option('query').prop['seq_type'], self.option('query_type')]))
    if self.option("database") == 'customer_mode':
        if not self.option("reference").is_set:
            raise OptionError("使用自定义数据库模式时必须设置reference")
        if self.option('reference_type') not in ['nucl', 'prot']:
            raise OptionError('reference_type参考序列的类型为nucl(核酸)或者prot(蛋白):{}'.format(self.option('query_type')))
        else:
            if self._fasta_type[self.option('reference').prop['seq_type']] != self.option('reference_type'):
                raise OptionError(
                    '文件检查发现参考序列为:{}, 而说明的文件类型为:{}'.format(
                        self._fasta_type[self.option('reference').prop['seq_type'], self.option('reference_type')]))
    elif self.option("database").lower() not in ["nt", "nr", "string", 'kegg', 'swissprot']:
        raise OptionError("数据库%s不被支持" % self.option("database"))
    else:
        self.option('reference_type', self._database_type[self.option("database").lower()])
    if not 15 > self.option('outfmt') > -1:
        raise OptionError('outfmt遵循blast+输出规则，必须为0-14之间：{}'.format(self.option('outfmt')))
    if not 1 > self.option('evalue') >= 0:
        raise OptionError('E-value值设定必须为[0-1)之间：{}'.format(self.option('evalue')))
    if not 0 < self.option('num_alignment') < 1001:
        raise OptionError('序列比对保留数必须设置在1-1000之间:{}'.format(self.option('num_alignment')))
    if self.option('blast') not in self._blast_type[self.option('query_type')][self.option('reference_type')]:
        raise OptionError(
            '程序不试用于提供的查询序列和库的类型，请仔细检查，核酸比对核酸库只能使用blastn或者tblastn，\
             核酸比对蛋白库只能使用blastp， 蛋白比对蛋白库只能使用blastp，或者没有提供blast参数')
    return True
```

- 重写*check_options*方法：检查*tool*运行时传入的参数是否合理和逻辑关系，以确保*tool*能够正确运行
- 不可消耗过多资源和大量计算
- 检查出错*raise OptionError*(“错误信息”)

注意：

- self.option(“参数名”)如果是infile，可以有is_set的属性，检查是否提供该文件。
- self.option(“参数名”)如果是infile/outfile(结束后设定)，返回的是文件对象，具有prop的属性是一个字典。由file中的set_property添加，已有size和md5。path属性为文件路径。
- self.option(“参数名”)不是文件时直接返回该值，值为设定的类型
- infile在参数传入时自动调用了check方法，但是并没有自动调用get_info方法(如果check中调用了get_info则另当别论)，以节省不必要的计算资源浪费。所以在使用prop属性前，务必确认，get_info已经被调用

# tool&agent开发和测试—tool&agent开发—agent

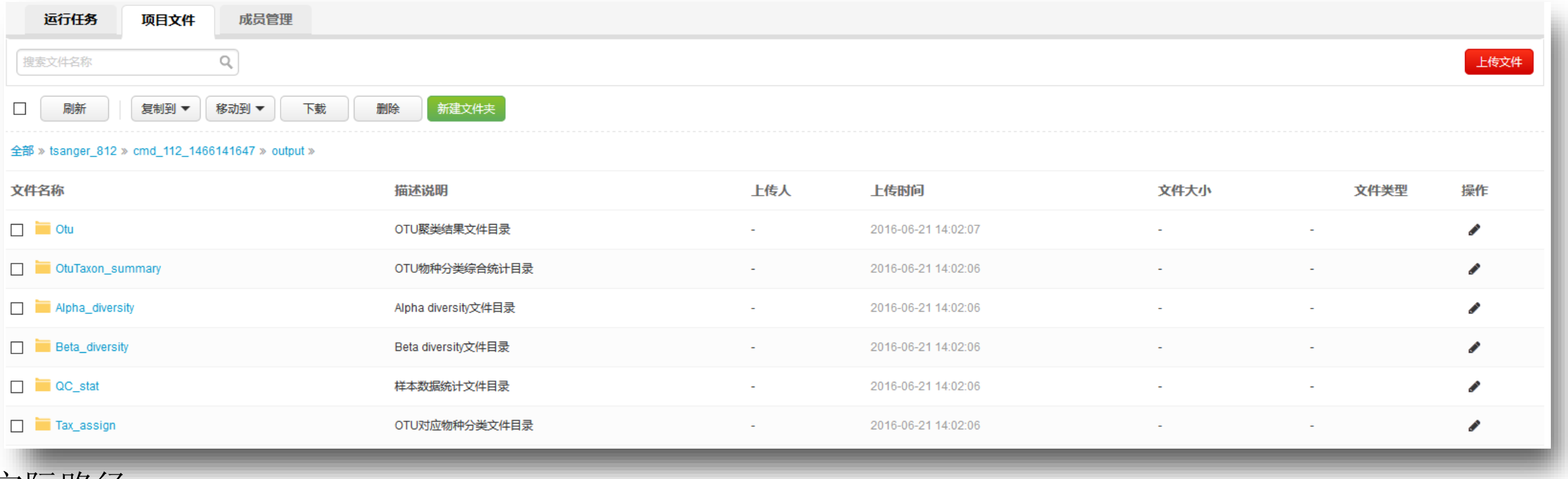
资源设定：必须重写set_resource，cpu至少为1，memory设置为自由容量单位，如”100M”或者”10G”。软件需要测试，了解资源消耗。如果和输入文件大小等有关，请进行测试，估算资源消耗随文件大小变化的规律。
java等程序可以指定程序需求资源，此处资源请求不得少于程序运行设置。

```
    def set_resource(self):
        self._cpu = 10
        self._memory = ''

    def end(self):
        result_dir = self.add_upload_dir(self.output_dir)
        result_dir.add_relpath_rules([
            [".", "", "结果输出目录"],
            ])
        result_dir.add_regexp_rules([
            [r".+_vs_.+\.xml", "xml", "blast比对输出结果，xml格式"],
            [r".+_vs_.+\.xls", "xls", "blast比对输出结果，表格(制表符分隔)格式"],
            [r".+_vs_.+\.txt", "txt", "blast比对输出结果，非xml和表格(制表符分隔)格式"],
            [r".+_vs_.+\.txt_\d+\.xml", "xml", "Blast比对输出多xml结果，输出格式为14的单个比对结果文件,主结果文件在txt文件中"],
            [r".+_vs_.+\.txt_\d+\.json", "json", "Blast比输出对多json结果，输出格式为13的单个比对结果文件,主结果文件在txt文件中"],
            ])
        # print self.get_upload_files()
        super(BlastAgent, self).end()
```

# tool&agent开发和测试—tool&agent开发—agent

上传结果与上传结果文件信息：文件被上传到特定网盘位置。用于网页显示结果文件，提供下载。




实际路径：

*/mnt/ilustre/192.168.12.102/rerewrweset/files/m_160/10000411/tsanger_822/cmd_112_1466478373/output*

网盘文件目录、用户*id*、*project_sn*、*task_id*、 *stage_id*(一个任务下的子*id*没有实际使用)

- add_upload_dir(目录) # 添加需要上传的目录，返回上传对象
- add_relpath_rules #上传对象添加目录下完整相对路径信息规则
- add_regexp_rules #上传对象添加目录下正则表达式匹配路径信息规则

注意：

- 列表的第一个元素为匹配名称规则，第二个元素为文件类型(没有实际意义，文件夹时为空)，第三个元素为文件描述
- 先匹配正则表达式规则，后匹配完整路径规则，所有文件对一个规则进行匹配，整体结果是后匹配上生效

# tool&agent开发和测试—tool&agent开发—tool

- 软件目录：使用self.config.SOFTWARE_DIR表示app目录
- 设置环境变量：set_environ，当软件运行需要特定的其他软件或者库文件时，需要设置环境变量，具体参考软件说明。设置方法例如：
  self.set_environ(LD_LIBRARY_PATH=self.config.SOFTWARE_DIR + 'gcc/5.1.0/lib64')
- 工作目录：self.work_dir(工作目录)，self.output_dir(输出结果目录)
- run：重写run方法，run函数中运行tool计算的实际部分

```
    def run(self):
        """
        运行
        :return:
        """
        super(BlastTool, self).run()
        if self.option("database") == 'customer_mode':
            self.run_makedb_and_blast()
        else:
            self.run_blast(self.option("database"))
```

# tool&agent开发和测试—tool&agent开发—tool

```
        blast_command = self.add_command("blast", cmd)
        blast_command.run()
        self.wait()
        if blast_command.return_code == 0:
```

- 命令字符串应该从app目录后起，add_command自动添加app前的目录
- add_command方法返回command对象，对象的run方法运行命令。
- self.wait()运行等待所有command运行结束。
- self.wait()参数为command名称，可以为多个，等待特定commands完成。
- command对象需要检查return_code是否为0，绝多数情况，程序正常结束结果为0

注意：

- 尽量使用add_command而不是使用其他方式（subprocess等）运行命令
- 在命令中有’|’，’<’，’>’符号时不可使用add_command，可以使用subprocess方法或者其他多进程方法，同样需要检查是否正确结束。
- packages函数的使用计算量小的可以直接执行，也可以单开进程和线程运行，大计算量的写成脚本用add_command的方式运行脚本

- 结果文件可能成为其他tool等的输入，使用self.option('name', file_path)赋值，请勿使用self.option('name').set_path(file_path)，前者具有文件检查的功能。
- run的结束必须为set_error("错误信息")或者是end()，一般情况在判断程序是否正常结束后选择。
- 结果文件必须拷贝到输出目录下，大文件放到结果目录务必使用链接的方式(os.link)
- 结果文件尽量按用户可识别文件格式标识后缀，例如，制表符分隔的表格文件以.xls为后缀

# tool&agent开发和测试—api

database:将tool等的结果数据导入mongo数据库，后续细讲
to_file:将mongo数据库中的数据转成tool实际计算需要的文件，后续细讲
web:向web端传递任务信息（step，upload_dir信息等）

# tool&agent开发和测试—测试

- 只能运行workflow：只有workflow可以接收参数
- sheet对象：workflow的参数形式
- 使用singleworkflow：接收sheet参数，根据type运行模块
- 结果目录：~/workspace/日期/Single_json中的id，
- 错误信息查看：主目录和tool目录下的log.txt，tool目录下的.err文件和.out文件，还有屏幕。查错请在log中写入适当的信息

```
{
  "id":"blast001",
  "type" : "tool",
  "name" : "align.ncbi.blast",
  "options":{
      "query":"/mnt/ilustre/users/sanger/sgBioinfo/shenghe/test_file/small_trinity.fasta",
      "query_type":"nucl",
      "database":"string",
      "blast": "blastx"
  }
}
```

```
from mbio.workflows.single import SingleWorkflow
from biocluster.wsheet import Sheet

wsheet = Sheet("path/to/single_blast_1.json")
wf = SingleWorkflow(wsheet)
wf.run()
```

# tool&agent开发和测试—测试

使用logger：

- 日志对象，用于向屏幕和特定文件以特定的格式写入运行信息
- 信息分四个等级：debug、info、warning、error
- 一般情况下只用info
- 记录程序运行状态进度，用于调试打印关键值

# tool&agent开发和测试—常见问题

- 不要改变传入的参数对象
- 在使用文件属性时确认已经get_info
- add_command会自动添加路径到app目录，这个路径尾部没有’/’
- 所有路径，self.workdir, self.config.SOFTWARE_DIR等尾部都没有’/’，使用’+’时需要注意，最好使用os.path.join()

# module&workflow开发和测试—workflow

- add_tool/add_module方法：给workflow添加一个子tool（实际是agent）/module，并返回该对象，对象在set_options(字典：参数名为键，参数值为值，值可以为文件对象，即其他tool的outfile)后可以通过该对象的run方法，运行该tool（agent）/module
- run方法：super父类(即workflow)的run方法必需在run的结尾处，因为该方法是阻塞的。

强调：tool等的run方法是不阻塞的

- 依赖：
  - on：方法参数有三个，第一个为事件名称，第二个是绑定处理函数对象，第三个是函数对象的参数，必须为简单的python对象。处理函数不能直接获取到参数，需要处理函数的第一个参数接收一个字典对象，字典的键为data的值为参数，而bind_object键的值为绑定的对象，依赖必须在tool/module/workflow运行之前
  - on_rely：至少两个tool等的end事件触发一个函数对象，与on一样的方式获取参数

# module&workflow开发和测试—workflow

- database：需要使用api/database下的对象将结果文件写入mongo库
  - 继承于biocluster.api.database.base Base
  - self.db为需要写入的mongo数据库对象
  - self.bind_object有可能是运行的workflow对象，也可能是只有workflow一些基本属性的假对象。
  - 故只可以使用假对象的属性：
    - id # task_id
    - name # 名称
    - workdir # 工作目录
    - fullname # 全称(加上父模块名称)
    - output # 模块输出目录
    - sheet # workflow参数对象完整还原，主要信息可以在sheet中找到，例如：project_sn，options等等

# module&workflow开发和测试—workflow

- database：
  - 写入数据库函数必须提供一个参数，main(控制主表和detail表的写入)
  - 主表：一个结果文件(结果表)的基本信息
  - detail表：结果文件的拆分存放的内容表
  - 函数必须返回主表id(ObjectId)

# module&workflow开发和测试—workflow

End：

- end：在end之前需要将各个调用的子模块的结果放到worflow的output中，重写end方法时，需要在结尾super父类的end
- upload_dir：需要上传文件，同样需要将结果文件上传，上传文件并不在workflow 中运行，而是另外将上传信息写入mysql数据库中，由特定进程上传文件，可以使用clone_upload_dir_from(obj)克隆子模块的上传文件夹/文件信息
- 关于代码截停：IMPORT_REPORT_DATA, IMPORT_REPORT_AFTER_END的参数同时为真，会使api-database对象的函数调用被不执行，在程序结束后再通过记录执行入库操作，一般情况下不需要使用，只有在入库数据量特别大时使用。
- 由于代码截停，入库操作函数返回值不可再workflow中使用，否则造成错误，但是可以在其他后续入库操作中使用。

# module&workflow开发和测试—module

module和workflow的编写基本一致：

- 不同的地方在于super父类的run方法和tool一样，在run的开始
- 不需要写入数据库操作

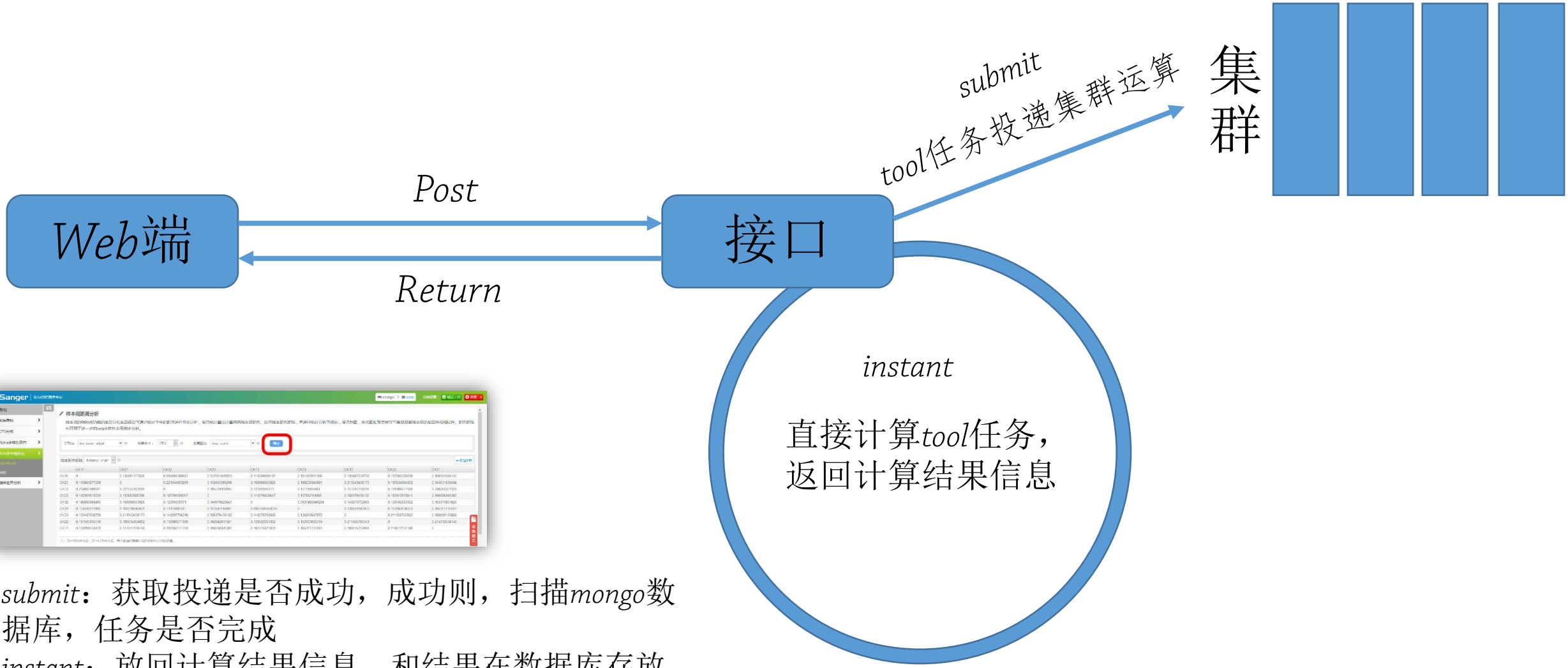
# module&workflow开发和测试—测试

与tool同样的测试方式

# module&workflow开发和测试—常见问题

- a

# controller网页接口开发和测试—网页投递流程与mongo数据库



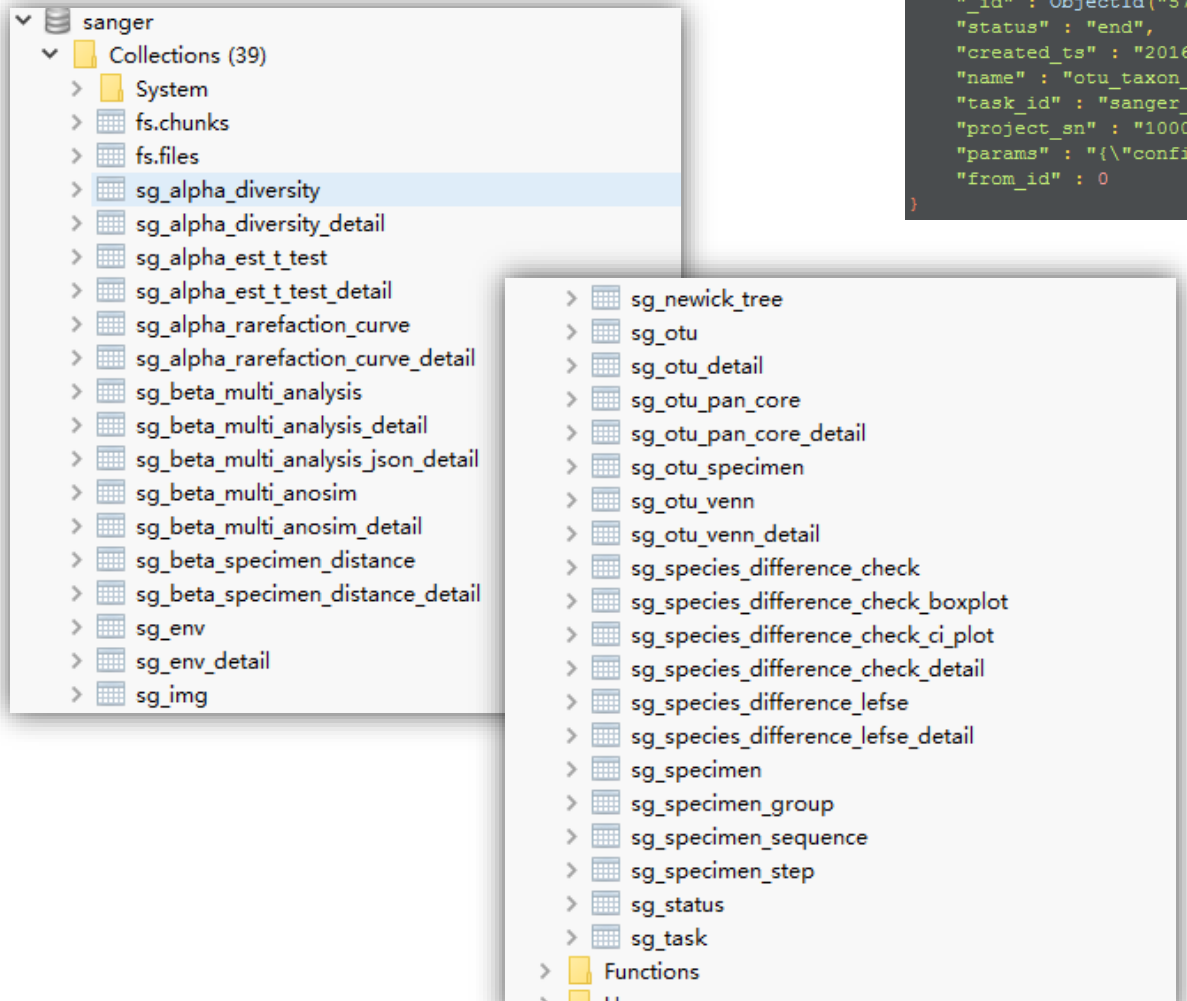



*submit*：获取投递是否成功，成功则，扫描*mongo*数据库，任务是否完成

*instant*：放回计算结果信息，和结果在数据库存放信息

# controller网页接口开发和测试—网页投递流程与mongo数据库

## mongo数据库：




### sg_otu


```
{
    "_id" : ObjectId("578d851a17b2bf28d858d99f"),
    "status" : "end",
    "created_ts" : "2016-07-19 09:40:42",
    "name" : "otu_taxon_origin",
    "task_id" : "sanger_882",
    "project_sn" : "10000440",
    "params" : "{\"confidence\":0.8,\"database\":\"unite7.0/its_fungi\",\"identity\":0.97,\"otu_id\":\"None\",\"revcomp\":false}",
    "from_id" : 0
}
```


### sg_otu_detail


```
{
    "_id" : ObjectId("56a851f20e6da959c9029b0f"),
    "task_id" : "sanger_268",
    "otu_id" : ObjectId("56a851f20e6da959c9029afa"),
    "d__" : "d__Eukaryota",
    "k__" : "k__Fungi",
    "p__" : "p__Ascomycota",
    "c__" : "c__Sordariomycetes",
    "o__" : "o__Xylariales",
    "otu" : "OTU14",
    "sample1" : 17,
    "sample5" : 0,
    "sample6" : 0,
    "sample2" : 0,
    "sample4" : 0,
    "sample7" : 0,
    "sample3" : 0
}
```

# controller网页接口开发和测试—即时(instant)模式—controller

controller：

- 对象继承自MetaController(只限于meta开发)
- Metacontroller是对post进来的内容进行了meta分析信息的检查，并获取基础信息：taskId、projectSn、memberId
- Metacontroller继承自webroot\mainapp\controllers\core\basic.py，basic提供了即时运行的基本设置
- super父类的post方法。返回值必须为空，否则返回前端success:false
- 检查模块需要的参数和参数逻辑。
- 设置运行workflow需要的sheet中的部分参数，self.task_name(name)，self.options(options)，self.to_file(to_file)。
- 使用run方法
- return returnInfo

注意：将写数据库必要的参数放在options中传入workflow

params：

- params需要将前端传入的参数打包成字符串存入mongo，主要用于参数比对，避免重复运行相同的任务，键参数名需要排序，不可有空格
- 一般情况下使用json.dumps(params_json, sort_keys=True, separators=(',', ':'))
- 特殊情况下，例如：group信息，内部同样需要排序

# controller网页接口开发和测试—即时(instant)模式—workflow

与一般的workflow不同的是主表的id需要使用self.add_return_mongo_id()的方法，参数有三个：表名， id，说明描述

# controller网页接口开发和测试—投递(submit)模式

controller：

- 没有任何继承
- post函数需要身份验证修饰器check_sig
- 检查参数和逻辑
- 需要获取task_id
- get_new_id　#渊源为前端从交互分析发起的任务不认为是一个任务，而是基于原有任务的任务，后端为了区分，必须为新的任务生成新的id
- 获取member_id和project_sn生成特定的文件上传目标目录
- 写主表(包括params，状态status为start)
- 写入mysql表任务信息(sheet信息等)
- return 投递成功信息

workflow：不需要写入主表，其他与即时模块相同

# controller网页接口开发和测试—测试

- webroot\main.py(main_wsgi.py)
- 新的controller模块需要添加到main.py在main.py测试通过后，加入main_wsgi.py，例如：

  from mainapp.controllers.instant.meta.beta.anosim import Anosim

  在url中添加url指向

  "/meta/beta/anosim", "Anosim“
- 开启信息接收端口：1-65536，自选较大的端口号，不可使用已被占用的端口号，例如80，22，24，3366，27017等，相互协调，不可重选

  例如：python main.py 8090
- 向端口发送测试数据：bin\webapitest.py

  例如：python webapitest.py post meta/beta/anosim -c client01 -n "otu_id;level_id;distance_algorithm;permutations;group_id;group_detail“ -d "56ce51860e6da9cf6bd716f3;8;pearson;999;56ce50430e6da9cf6bd716e9;/mnt/ilustre/users/sanger/sheng.he_test/group_test_post.txt" -b http://192.168.12.102:8090

# *controller*网页接口开发和测试—常见问题

- *a*