Native ACT定义文档

1. kOS数据类型

1.1 常量类型

1.1.1 DataType

```python
class DataType(Enum):
    """
    数据类型
    """
    # 文本
    TEXT = 'text'
    # 语音
    VOICE = 'voice'
    # 图片
    IMAGE = 'image'
    # 视频
    VIDEO = 'video'
```

1.1.2 DataPriority

```python
class DataPriority(Enum):
    """
    数据优先级，高优先级的数据在默认搜索下会被优先检索
    """
    # 高优先级
    HIGH = 'HIGH'
    # 中优先级
    MEDIUM = 'MEDIUM'
    # 低优先级
    LOW = 'LOW'
```

1.1.3 MsgType

```python
class MsgType(Enum):
    """
    消息类型
    """
    # 文本消息
    TEXT = 'text'
    # 语音消息
    VOICE = 'voice'
    # 图片消息
    IMAGE = 'image'
    # 视频消息
    VIDEO = 'video'
    # 文件消息
    FILE = 'file'
```

1.1.4 FileMountType

```python
class FileMountType(Enum):
    """
    文件的挂载类型
    """
    # 网盘挂载
    NET_DISK = 'netDisk'
    # 资料库挂载
    MATERIAL = 'material'
```

1.1.5 FileOpenMode

```python
class FileOpenMode(Enum):
    """
    文件打开模式
    """
    # 只读模式，文件必须存在
    READ = 'read'
    # 追加模式，文件不存在则新建，存在则在文件末尾追加内容
```

```
    APPEND = 'append'
    # 覆盖模式，文件不存在则新建，存在则open的时候会先清空已有文件内容，再写入文件内容
    OVERWRITE = 'overwrite'
```

1.2 LUI

1.2.1 KOSMsg

```
class KOSMsg:
    """
    LUI消息
    """
    def __init__(self, msg_id: str, msg_type: str, text: str, file: KOSFile = No
        """
        :param msg_id:
          消息ID，全局唯一
        :param msg_type:
          消息类型
        :param text:
          消息文本内容
        :param file:
          消息文件对象，只在文件类消息有效
        """
        self.msg_id = msg_id
        self.msg_type = msg_type
        self.text = text
        self.file = file
```

1.3 元空间

1.3.1 KOSMetaSpace

```
class KOSMetaSpace:
    """
    元空间，通常由对用户数据进行维度展开之后的元数据集合组成
    """
    def __init__(self, space_id: str):
        """
```

```
        :param space_id:
          元空间对象ID，全局唯一
        """
        self.space_id = space_id
```

1.3.2 KOSMetaData

```
class KOSMetaData:
    """
    元数据，用户数据经过维度展开之后形成的特征数据
    """
    def __init__(self, data_id: str, data_type: str, content: str):
        """
        :param data_id:
          元数据对象ID，全局唯一
        :param data_type:
          数据类型
        :param content:
          数据内容
        """
        self.data_id = data_id
        self.data_type = data_type
        self.content = content
```

1.3.3 KOSFeature

```
class KOSFeature:
    """
    数据的特征数据，通常通过向量来表达
    """
    def __init__(self, query: str, vector: Any):
        """
        :param query:
          原数据内容
        :param vector:
          数据特征属性向量，不同场景下的数据类型可能不同
        """
        self.query = query
        self.vector = vector
```

1.4 文件系统

1.4.1 KOSFile

```
class KOSFile:
    """
    文件系统中的文件对象。kOS文件系统是存储器中的核心部件之一。
    在文件系统中，文件有两种不同的挂载方式（具体值参考FileMountType常量）：
    * NET_DISK，网盘挂载，为默认的挂载方式，此方式下的根目录为控制台星盘的"我的文件/用户文
      务必注意，若文件执行k_semantic_analyse_file语义解析之后，则会被自动重新进行MATER
    * MATERIAL，资料库挂载，此方式下的根目录为控制台星盘的"星伴知识库/用户文件"。
      此挂载下的文件，会被自动进行语义解析，提取相关的语义信息。

    此外，文件还具有数据优先级，此优先级在进行有关数据搜索的时候会影响检索的优先级。具体参考
    """
    def __init__(self, file_id: str, file_path: str, file_name: str, file_size:
                 mount_type: FileMountType, priority: DataPriority, open_mode: F
        """
        :param file_id:
          文件对象ID，全局唯一
        :param file_path:
          文件绝对路径
        :param file_name:
          文件名
        :param file_size:
          文件大小，单位字节
        :param create_time:
          文件创建时间戳
        :param mount_type:
          挂载类型
        :param priority:
          优先级
        :param open_mode:
          文件打开模式，只在文件被open的时候有效
        """
        self.file_id = file_id
        self.file_path = file_path
        self.file_name = file_name
        self.file_size = file_size
        self.create_time = create_time
        self.mount_type = mount_type
        self.priority = priority
```

```
        self.open_mode = open_mode
```

1.5 资料库

1.5.1 KOSMaterialData

```
class KOSMaterialData:
    """
    资料库数据，文件经过语义理解之后会形成资料库数据并存储在资料库中。
    """
    def __init__(self, data_id: str, file_id: str, content: str):
        """
        :param data_id:
          资料库数据ID
        :param file_id:
          资料库数据所属的文件ID
        :param content:
          资料库数据内容
        """
        self.data_id = data_id
        self.file_id = file_id
        self.content = content
```

1.6 网络

1.6.1 KOSWebSearchTopic

```
class KOSWebSearchTopic:
    """
    互联网搜索结果，由标题、摘要、源url组成
    """
    def __init__(self, title: str, abstraction: str, source_url: str):
        """
        :param title:
          标题
        :param abstraction:
          摘要
```

```
        :param source_url:
          源url
        """
        self.title = title
        self.abstraction = abstraction
        self.source_url = source_url
```

2. LUI

2.1 k_message_send：消息发送

```
def k_message_send(content: Union[KOSFile, str]):
```

```
向用户发送消息。注意，若是直接发送文本，最大长度不能超过512，若是超过会被自动截断
:param content:
  消息内容，支持文本或者文件对象，其它类型会抛异常退出
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    ofile = nact.k_file_open("/data/news.txt")
    nact.k_message_send(str(ofile.file_path))
    nact.k_message_send(ofile)

main()
```

2.2 k_ask_for_file：要求用户上传文件

```
def k_ask_for_file() -> KOSFile:
```

```
基于LUI的多轮对话，向用户要求上传1个文件
:return:
  返回用户上传的文件对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    ofile = nact.k_ask_for_file()
    nact.k_message_send(str(ofile.file_path))
    nact.k_message_send(ofile)

main()
```

2.3 k_ask_for_files：要求用户上传多个文件

```
def k_ask_for_files(min_num: int, max_num: int) -> List[KOSFile]:
```

```
基于LUI的多轮对话，向用户要求上传多个文件，比如要求用户上传2-5个文件
:param min_num:
  最少几个文件
:param max_num:
  最多几个文件
:return:
```

```
    返回用户上传的文件对象列表
```

示例：

```python
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    file_list = nact.k_ask_for_files(1,2)
    for f in file_list:
        nact.k_message_send(str(f.file_path))

 main()
```

2.4 k_ask_for_answer：要求用户补充回答

```python
def k_ask_for_answer(question: str) -> str:
```

```
基于LUI的多轮对话，要求用户回答有关问题
:param question:
:return:
  返回用户的回答
```

示例：

```python
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_ask_for_answer("你是谁")
    nact.k_message_send(res)

main()
```

2.5 k_get_act_query：获取ACT启动输入内容

```
def k_get_act_query() -> str:
```

```
获取ACT启动时携带的用户默认输入
:return:
    返回用户默认输入
```

3. 元空间

3.1 k_meta_space_open：打开元空间

```
def k_meta_space_open() -> KOSMetaSpace:
```

```
打开用户的元空间，每个用户只有一个元空间
:return:
    元空间对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    meta_space= nact.k_meta_space_open()
```

```
    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature)

    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature)
    nact.k_message_send("search context for : "+mdata_list[0].content)

    preceding ,succeeding =nact.k_meta_data_get_context(mdata_list[0],4)
    nact.k_message_send("search over")
    if preceding:
        for pre in preceding:
            nact.k_message_send(pre.conten)
    if succeeding:
        for suc in succeeding:
            nact.k_message_send(suc.conten)


main()
```

3.2 k_data_dehydration：数据脱水

```
def k_data_dehydration(file: KOSFile, meta_space: KOSMetaSpace = None) -> KOSMet
```

```
数据脱水，将文件进行降维展开到元空间，展开后元空间会包含该文件对应的一系列元数据对象集合
:param file:
  需要展开的文件对象
:param meta_space:
  需要在哪个元空间展开，不指定的话会自动获取用户的元空间
:return:
  返回该元空间对象
```

示例：

```python
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    nact.k_message_send("start ")
    file = nact.k_file_open("/data/news-short.txt")

    meta_space= nact.k_data_dehydration(file)

    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    nact.k_message_send("compute over")

    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
    nact.k_message_send("search over")

    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(str(idx))
        nact.k_message_send(nact.k_meta_data_get_text(mdata))

main()

```

3.3 k_meta_space_correlation_compute：元空间关联计算

```python
def k_meta_space_correlation_compute(meta_space: KOSMetaSpace, feature:
KOSFeature, file: KOSFile= None):
```

```
元空间数据关联计算，会根据输入的feature来计算关联的元数据，并为元数据添加与feature的关联。
:param meta_space:
  元空间对象
:param feature:
  要关联的feature
:param file:
```

```
    是否指定文件关联，若指定则只会计算该文件有关的元数据
```

示例：

```python
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    nact.k_message_send("start ")
    file = nact.k_file_open("/data/news-short.txt")

    meta_space= nact.k_data_dehydration(file)

    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    nact.k_message_send("compute over")

    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
    nact.k_message_send("search over")

    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(str(idx))
        nact.k_message_send(nact.k_meta_data_get_text(mdata))

main()
```

3.4 k_meta_space_search：元空间搜索

```python
def k_meta_space_search(meta_space: KOSMetaSpace, feature: KOSFeature, file:
KOSFile= None) -> List[KOSMetaData]:
```

```
元空间搜索与feature有关的元数据
:param meta_space:
  元空间
:param feature:
```

```
    有关联的feature
:param file:
    文件对象，若指定则只会搜索该文件有关的元数据
:return:
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    nact.k_message_send("start ")
    file = nact.k_file_open("/data/news-short.txt")

    meta_space= nact.k_data_dehydration(file)

    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    nact.k_message_send("compute over")

    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
    nact.k_message_send("search over")

    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(str(idx))
        nact.k_message_send(nact.k_meta_data_get_text(mdata))

main()

```

3.5 k_meta_space_update_meta_data：元空间更新元数据

```
def k_meta_space_update_meta_data(meta_space: KOSMetaSpace, meta_data:
KOSMetaData, new_content: str):
```

```
更新元数据内容
:param meta_space:
  元空间对象
:param meta_data:
  元数据对象
:param new_content:
  要更新的内容
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    nact.k_message_send("start ")
    file = nact.k_file_open("/data/news-short.txt")
    meta_space= nact.k_data_dehydration(file)

    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
    nact.k_message_send("search over"+str(mdata_list))

    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(str(idx))
        nact.k_message_send(nact.k_meta_data_get_text(mdata))
        nact.k_meta_space_update_meta_data(meta_space,mdata,"upd"+mdata.content+

    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(nact.k_meta_data_get_text(mdata)+": updated")

main()
```

3.6 k_meta_data_rehydration：元数据浸泡

```
def k_meta_data_rehydration(meta_space: KOSMetaSpace, org_file: KOSFile,
new_file: KOSFile) -> KOSFile:
```

```
元数据浸泡，将元数据从元空间还原为文本数据，并保存在结果文件。结果文件需要先通过k_file_ope
:param meta_space:
  元空间对象
:param org_file:
  原文件对象
:param new_file:
  结果文件对象
:return:
  结果文件对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact
def main():
    query="把中信银行改为兔不二科技"
    # 语义特征提取：即：标签
    semantic_feature = nact.k_semantic_compute_feature(query)

    # 搜索 待修改元空间：根据 语义特征 搜索元空间.
    ready_edit_meta_data_list = nact.k_meta_space_search(meta_space, semantic_fe
    nact.k_message_send("search "+str(len(ready_edit_meta_data_list)))

    # 循环 待修改元空间
    for ready_edit_meta_data in ready_edit_meta_data_list:
        # 元数据重新生成
        new_meta_data = nact.k_meta_data_semantic_rephrase(ready_edit_meta_data,
        # 修改元空间:
        nact.k_meta_space_update_meta_data(meta_space, ready_edit_meta_data, new

    # 生成文件
    nfile = nact.k_file_open("/data/news-gen.txt",nact.FileOpenMode.OVERWRITE)
    new_file = nact.k_meta_data_rehydration(meta_space, file,nfile)
    nact.k_message_send("new file  gen over")
```

```
23
24 main()
```

3.7 k_meta_data_get_text：获取元数据文本内容

```
1 def k_meta_data_get_text(meta_data: KOSMetaData) -> str:
```

```
1 获取元数据的文本内容
2 :param meta_data:
3   元数据对象
4 :return:
5   元数据的文本内容，如果不是文本元数据则返回为空
```

示例：

```
1 # -*- coding: utf-8 -*-
2 from kOS import nact
3
4 def main():
5     nact.k_message_send("start ")
6     file = nact.k_file_open("/data/news-short.txt")
7     meta_space= nact.k_data_dehydration(file)
8
9     semantic_feature = nact.k_semantic_compute_feature("中信")
10    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur
11
12    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
13    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
14    nact.k_message_send("search over : "+len(mdata_list))
15
16    for idx,mdata in enumerate(mdata_list):
17        nact.k_message_send(nact.k_meta_data_get_text(mdata)+": updated")
18
```

```
19 main()
```

3.8 k_meta_data_update_tags：更新元数据标签

```
1 def k_meta_data_update_tags(meta_data: KOSMetaData, tags: List[str]):
```

```
1 更新元数据对象的标签
2 :param meta_data:
3   元数据对象
4 :param tags:
5   标签列表
```

示例：

```
1  # -*- coding: utf-8 -*-
2  from kOS import nact
3
4  def main():
5      nact.k_message_send("start")
6      file = nact.k_file_open("/data/news-short.txt")
7      meta_space= nact.k_data_dehydration(file)
8      semantic_feature = nact.k_semantic_compute_feature("中信")
9      nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur
10
11     nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
12     mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)
13
14     nact.k_message_send("search over"+str(mdata_list))
15
16     tag_list=["理财", "笔记本", "耳机", "路由器", "电脑", "打印机", "扬声器", "U盘"]
17     for idx,mdata in enumerate(mdata_list):
18         nact.k_message_send(str(idx))
19
20         nact.k_message_send(nact.k_meta_data_get_text(mdata))
21         nact.k_meta_data_update_tags(mdata,[tag_list[idx%8]])
22         nact.k_message_send("tag update:"+tag_list[idx%8])
23
```

```
        idx_tag = nact.k_semantic_compute_feature(tag_list[idx%8])

        idx_mdata_list=nact.k_meta_space_search(meta_space,idx_tag,file)
        for idx_mdata in idx_mdata_list:
            nact.k_message_send("update:"+str(idx)+":"+nact.k_meta_data_get_text

main()

```

3.9 k_meta_data_semantic_rephrase：元数据语义改写

```
def k_meta_data_semantic_rephrase(data: KOSMetaData, feature: KOSFeature) ->
str:
```

```
根据语义重新表达元数据内容
:param data:
  元数据对象
:param feature:
  语义feature
:return:
  重新表达后的内容
```

3.10 k_meta_data_get_context：获取元数据上下文

```
def k_meta_data_get_context(meta_data: KOSMetaData, offset: int) ->
Tuple[List[KOSMetaData], List[KOSMetaData]]:
```

```
获取元数据对象的上下关联的元数据对象列表，即其上下文
:param meta_data:
  元数据对象
:param offset:
```

```
    要获取多少范围内的上下文元数据对象
:return:
    返回元组，第一个元素为上文元数据对象列表，第二个为下文元数据对象列表
```

示例：

```python
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    file = nact.k_file_open("/data/news-short.txt")
    meta_space= nact.k_data_dehydration(file)

    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)

    nact.k_message_send("search over"+mdata_list[0].content)
    preceding ,succeeding =nact.k_meta_data_get_context(mdata_list[0],4)

    if preceding:
        for pre in preceding:
            nact.k_message_send(pre.content)
    if succeeding:
        for suc in succeeding:
            nact.k_message_send(suc.content)

main()

```

4. 控制

4.1 k_semantic_evaluate_input：评估用户输入

```
def k_semantic_evaluate_input(input: str, scenario: str, requirements: str) ->
bool:
```

```
评估用户给的输入是否满足特定场景下的输入要求
:param input:
  用户输入
:param scenario:
  指定的场景
:param requirements:
  该场景下的输入要求
:return:
  返回是否满足，True表示满足，False表示不满足
```

4.2 k_semantic_confirm_input：评估确认用户输入

```
def k_semantic_confirm_input(input: str, scenario: str, requirements: str) ->
str:
```

```
确认用户输入是否满足特定场景下的输入要求，如果不满足，则会通过LUI要求用户重新输入
:param input:
  用户输入
:param scenario:
  指定的场景
:param requirements:
  该场景下的输入要求
:return:
  返回满足该场景下输入要求的用户输入，如果原用户输入本来就满足则返回原输入，否则会要求用户重
```

5. 存储

5.1 文件系统

5.1.1 k_file_open：打开文件

```
def k_file_open(file_path: str, mode: FileOpenMode = FileOpenMode.READ,
                priority: DataPriority = None, mount_type: FileMountType  =
FileMountType.NET_DISK) -> KOSFile:
```

```
打开一个文件，不同模式下行为不一样，具体参考常量类型FileOpenMode的说明。
注意：与传统操作系统中的文件打开不同，这里打开后无需close。
:param file_path:
  文件的绝对路径，如/a/b/c/file.txt
:param mode:
  文件打开模式，默认是只读模式，类型为FileOpenMode
:param priority:
  文件优先级，当文件被语义化解析之后，这个字段会影响文件语义检索优先级。具体参考DataPrior
:param mount_type:
  文件挂载类型，详细参考KOSFile说明。默认挂载在网盘上
:return:
  返回文件对象，只读模式下若文件不存在则返回None，其他情况都会返回文件对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    try:
        ofile = nact.k_file_open("/data/news.txt")
        oafile = nact.k_file_open("/data/news.txt",nact.FileOpenMode.APPEND)
        nact.k_file_append(oafile,"固收、固收+、混合、权益六大产品赛道,引入先进金融科技
        nact.k_file_append(ofile,"固收、固收+、混合、权益六大产品赛道,引入先进金融科技
     except Exception as e:
        nact.k_message_send(str(e))

main()
```

5.1.2 k_file_delete：删除文件

```
def k_file_delete(file: KOSFile):
```

```
删除文件
:param file:
  要删除的文件的文件对象
```

示例

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    ofile = nact.k_file_open("/data/noexist.txt",)
    nact.k_file_delete(ofile)
main()
```

5.1.3 k_file_append：追加文件内容

```
def k_file_append(file: KOSFile, content: str) -> KOSFile:
```

```
向一个已经存在的文件中追加内容
:param file:
  文件对象
:param content:
  要追加的文件内容
:return:
  返回新的文件对象，里面包括了文件最新的信息。若文件不存在，则返回None；若向一个只读文件追
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    ofile = nact.k_file_open("/data/append.txt",nact.FileOpenMode.OVERWRITE)
    nact.k_file_append(ofile,"固收、固收+、混合、权益六大产品赛道,引入先进金融科技赋")

main()
```

5.1.4 k_file_read：读取文件内容

```
def k_file_read(file: KOSFile, offset: int = None, length: int = None) ->
Tuple[int, str]:
```

```
读取文件内容，支持指定偏移位置及读取的长度
:param file:
  要读取的文件对象
:param offset:
  文件偏移，单位字节。默认表示从0开始
:param length:
  读取长度，单位字节。默认表示读取全部，但需要注意，最大只支持一次读取1MB长度
:return:
  返回实际读取的长度，单位字节，以及内容
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main()
    ofile = nact.k_file_open("/data/news.txt",)
    reslen,resstr=nact.k_file_read(ofile,10,10)
    nact.k_message_send("read "+str(reslen)+resstr)
main()
```

5.1.5 k_file_copy：拷贝文件

```
def k_file_copy(src_file: KOSFile, dst_file: KOSFile, offset: int = None,
length: int = None) -> int:
```

```
拷贝文件，将src_file文件指定的内容拷贝追加到dst_file中
:param src_file:
  源文件对象
:param dst_file:
  目标文件对象
:param offset:
  源文件位置偏移，单位字节，默认从0开始
:param length:
  读取长度，单位字节，默认尽可能的读取偏移后的全部内容，但须遵循k_file_append中的限制
:return:
  返回实际拷贝的长度，单位字节
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

```

```
def main()
    ofile = nact.k_file_open("/data/news-short.txt",)
    cfile = nact.k_file_open("/data/cnews.txt",nact.FileOpenMode.OVERWRITE)

    # nact.k_file_copy(ofile,cfile)
    nact.k_file_copy(ofile,cfile,10,10)
main()
```

5.2 资料库

5.2.1 k_material_semantic_search：资料库语义搜索

```
def k_material_semantic_search(query: str, priority: DataPriority = None,
top_n: int = 5,file_id_list: List[str] = None) -> List[KOSMaterialData]:
```

```
根据用户查询语句，语义搜索资料库
:param query:
:param top_n:
:param priority:
  资料优先级，类型为DataPriority
:param file_id_list:
  指定文件范围搜索
:return:
  返回匹配的资料数据列表
```

5.2.2 k_material_search_by_tags：根据标签搜索资料库

```
def k_material_search_by_tags(tags: List[str], priority: DataPriority = None,
top_n: int = 5) -> List[KOSMaterialData]:
```

```
根据标签精确搜索资料库
:param priority:
  资料优先级，类型为DataPriority
:param tags:
:param top_n:
:return:
  返回匹配的资料数据列表
```

5.2.3 k_material_update_tags：更新资料库标签

```
def k_material_update_tags(materia_data: KOSMaterialData, tags: List[str]):
```

```
更新某个资料数据的标签
:param materia_data:
:param tags:
:return:
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    nact.k_message_send("start")
    file = nact.k_file_open("/data/news-short.txt")
    meta_space= nact.k_data_dehydration(file)
    semantic_feature = nact.k_semantic_compute_feature("中信")
    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur

    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)
    mdata_list=nact.k_meta_space_search(meta_space,semantic_feature,file)

```

```
    nact.k_message_send("search over"+str(mdata_list))

    tag_list=["理财", "笔记本", "耳机", "路由器", "电脑", "打印机", "扬声器", "U盘"]
    for idx,mdata in enumerate(mdata_list):
        nact.k_message_send(str(idx))

        nact.k_message_send(nact.k_meta_data_get_text(mdata))
        nact.k_meta_data_update_tags(mdata,[tag_list[idx%8]])
        nact.k_message_send("tag update:"+tag_list[idx%8])

        idx_tag = nact.k_semantic_compute_feature(tag_list[idx%8])

        idx_mdata_list=nact.k_meta_space_search(meta_space,idx_tag,file)
        for idx_mdata in idx_mdata_list:
            nact.k_message_send("update:"+str(idx)+":"+nact.k_meta_data_get_text

main()

```

6. 运算

6.1 LLM

6.1.1 k_semantic_chat：大模型对话

```
def k_semantic_chat(prompt: str) -> str:
```

```
语义对话，类似于大模型的对话
:param prompt:
  提示词
:return:
  对话结果
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_chat("今天好无聊")
    nact.k_print(res)
main()
```

6.2 提示词工程

6.2.1 k_prompt_with_json_format：提示词添加JSON输出修饰

```
def k_prompt_with_json_format(prompt: str, json_format: dict) -> str:
```

```
给提示词添加JSON格式输出要求，以便对话结果可以被k_prompt_parse_json_output解析成JSON对
:param prompt:
  需要增强的提示词
:param json_format:
  需要输出的JSON格式说明，如{"result": "xxxx"}
:return:
  返回增强后的提示词
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_prompt_with_json_format("北京天气怎么样",{"city":"北京","weather":"
    nact.k_print(res)
main()
```

6.2.2 k_prompt_parse_json_output：从对话输出中解析JSON对象

```
def k_prompt_parse_json_output(output: str) -> dict:
```

```
从对话结果中解析JSON对象
:param output:
  对话结果
:return:
  解析后的json对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_prompt_parse_json_output('''
    "北京天气怎么样",{"city":"北京","weather":"sunny"}
    ''')
    nact.k_print(str(res))
main()
```

6.3 语义计算

6.3.1 k_semantic_compute_feature：特征计算

```
def k_semantic_compute_feature(query: str) -> KOSFeature:
```

```
计算数据的特征属性
:param query:
  要计算的输入文本
:return:
  返回的特征属性
```

6.3.2 k_semantic_compute_tags：打标签

```
def k_semantic_compute_tags(query: str, tags: List[str]) -> List[str]:
```

```
根据限定标签列表，给输入打标签
:param query:
  输入的文本内容
:param tags:
  限定的标签列表
:return:
  该输入文本对应的标签列表，如果与限定标签不匹配，则会返回空
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_compute_tags("北京天气怎么样",["weather",'query'])
    nact.k_print(str(res))
main()
```

6.3.3 k_semantic_summarize：语义总结

```
def k_semantic_summarize(content: str, requirements: str = None) -> str:
```

```
总结文本内容，若提供要求则表示需要根据用户的要求来进行针对性的总结
:param content:
  需要总结的长文本
:param requirements:
  用户要求
:return:
  返回总结后的结果
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_summarize('''作为中信银行在资产管理领域的重要布局,信银理财秉
    nact.k_print(str(res))
main()
```

6.3.4 k_semantic_summarize_answer：语义总结

```
def k_semantic_summarize_answer(content: str, question: str) -> str:
```

```
根据提问总结答案，用户问答类
:param content:
```

```
    需要总结的内容
:param question:
    问题
:return:
    返回答案
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_summarize_answer('''作为中信银行在资产管理领域的重要布局,信
    nact.k_print(str(res))
main()
```

6.3.5 k_semantic_analyse_file：语义理解文件内容

```
def k_semantic_analyse_file(file: KOSFile):
```

```
对文件进行语义理解，并将提取的语义存储在资料库中
:param file:
    要语义理解的文件对象
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
```

```
    ofile = nact.k_file_open("/data/news.txt")
    res=nact.k_semantic_analyse_file(ofile)
    nact.k_print(str(res))
main()
```

6.3.6 k_semantic_rephrase：语义重写

```
def k_semantic_rephrase(demand: str, content: str, prev_context: str = '',
next_context: str = '') -> str:
```

```
根据要求语义重写内容，为了更清晰的理解该内容，可以提供该内容的上下文
:param demand:
  改写要求
:param content:
  要改写的内容
:param prev_context:
  上文
:param next_context:
  下文
:return:
  改写后的内容
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_rephrase("把八戒改为贾宝玉，沙僧改为林黛玉，注意人物性格",'''孙
    nact.k_print(str(res))
main()
```

7. 网络

7.1 互联网

7.1.1 k_web_get_page_content：获取网页内容

```
def k_web_get_page_content(page_url: str) -> str:
```

```
获取网页文本内容
:param page_url:
  网页url，必须是http://或者https://
:return:
  返回网页文本内容
```

7.1.2 k_web_search_and_summarize：网络搜索并总结

```
def k_web_search_and_summarize(keyword: str) -> str:
```

```
根据关键字搜索互联网，并对返回搜索结果进行总结
:param keyword:
  搜索关键字
:return:
  返回搜索结果的总结内容。若返回为None，则说明未搜索到有关内容
```

7.1.3 k_web_search：网络搜索

```python
def k_web_search(keyword: str) -> List[KOSWebSearchTopic]:
```

```
根据关键字搜索互联网，并返回搜索结果列表
:param keyword:
  搜索关键字
:return:
  返回搜索结果列表。若返回为None，则说明未搜索到有关内容
```

8. 系统

8.1 k_sleep：休眠

```python
def k_sleep(ms: int):
```

```
让ACT休眠一段时间
:param ms:
  休眠时间，单位毫秒
```

8.2 k_print：打印输出

```python
def k_print(output: str):
```

```
ACT打印输出，会输出到用户调试对话窗口中。格式如：ACT>> 2023-12-15 16:36:52 [95c4c7e56
其中，[]中的为日志trace id，当有异常的时候可以将此id发给技术支持人员排查。
注意：此打印输出只在调试状态才有效，正式发布之后会屏蔽该打印
:param output:
  要打印的输出
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
    res=nact.k_semantic_chat("今天好无聊")
    nact.k_print("helloworld")
main()
```

8.3 k_assert：断言

```
def k_assert(cond: bool):
```

```
断言某个条件，如果为False，则抛异常，终止ACT执行
:param cond:
  要断言的条件
```

示例：

```
# -*- coding: utf-8 -*-
from kOS import nact

def main():
```

```
    # 文件上传
    file = nact.k_file_open("/data/news-short.txt")
    # 生成元空间：多维展开
    meta_space = nact.k_data_dehydration(file)

    query="把中信银行改为兔不二科技"
    # 语义特征提取：即：标签
    semantic_feature = nact.k_semantic_compute_feature(query)

    nact.k_message_send("feature get"+semantic_feature.query+str(semantic_featur
    # 修改元空间：关联度计算
    nact.k_meta_space_correlation_compute(meta_space, semantic_feature, file)


    # 搜索 待修改元空间：根据 语义特征 搜索元空间.
    ready_edit_meta_data_list = nact.k_meta_space_search(meta_space, semantic_fe
    nact.k_message_send(len(ready_edit_meta_data_list))

    nact.k_assert(False)
    nact.k_message_send("assert fail")
main()
```