



파이썬 – HW6

임베디드스쿨1기

lv1과정

2020. 09. 15

박성환

1. 연습(@staticmethod vs @classmethod)

1. 특징

- 클래스에서 직접 접근할 수 있는 메소드
- 다른언어와 다르게 인스턴스에서 static/class method에 접근 가능
- **인스턴스 메소드** -> 첫번째 인자 : 객체 자신 **self** 입력
classmethod -> 첫번째 인자 : **class** 를 입력(**cls**)
staticmethod -> 추가되는 인자 x

2. 구조

- staticmethod는 **self**를 사용x
- classmethod는 **cls**를 사용하여 클래스 속성을 인자로 받음
(instancemethod가 **self**를 사용하여 객체를 받는것과 같은 구조)

```
class CustomClass:

    # instance method
    def add_instance_method(self, a,b):
        return a + b

    # classmethod
    @classmethod
    def add_class_method(cls, a, b):
        return a + b

    # staticmethod
    @staticmethod
    def add_static_method(a, b):
        return a + b
```

1. 예습(@staticmethod vs @classmethod)

3. 예제

- 아래는 Linux 클래스는 Windows 클래스를 상속받아 모든 코드를 실행시킬 수 있는 코드임.
- 이 때 **클래스 메소드**는 cls로 현재 자기가 실행된 클래스 속성을 가져오고,
- 정적 메소드**는 부모클래스(Windows)의 속성을 가져오게 됨

```
class Windows:
    os = "window10"          # 클래스 속성

    def __init__(self):
        self.out = "OS: " + self.os

    @staticmethod
    def static_os():
        return Windows()

    @classmethod
    def class_os(cls):
        return cls()

    def os_output(self):
        print(self.out)

class Linux(Windows):
    os = "Linux"             # 클래스 속성

a = Linux.static_os()
a.os_output()

b = Linux.class_os()
b.os_output()

결과값 : OS: window10
결과값 : OS: Linux
```

4. 차이

상속에서 차이가 남

staticmethod: 부모클래스의 속성값을 가져옴

classmethod: cls인자를 활용하여 cls클래스 속성을 가져옴

```
class Person:
    default= "아빠"

    def __init__(self):
        self.data = self.default

    @classmethod
    def class_person(cls):
        return cls()

    @staticmethod
    def static_person():
        return Person()

class WhatPerson(Person):
    default = "엄마"

person1 = WhatPerson.class_person()    # return 엄마
person2 = WhatPerson.static_person()    # return 아빠
```

1. 예습(@staticmethod vs @classmethod)

4. 예습

```
1 """
2 staticmethod(function) 형태는 파이썬스럽지 않은 형태의 생성으로
3 최신 버전에는 @staticmethod 형태의 decorator를 사용함
4
5 @staticmethod
6 def func(args,...)
7 """
8 class CntManager:
9     cnt = 0
10    def __init__(self):
11        CntManager.cnt += 1
12    def staticPrintCnt():
13        print("Instance cnt:", CntManager.cnt)
14    sPrintCnt = staticmethod(staticPrintCnt)
15
16    def classPrintCnt(cls):
17        print("Instance cnt:", cls.cnt)
18    cPrintCnt = classmethod(classPrintCnt)
19
20 a, b, c = CntManager(), CntManager(), CntManager()
21
22 CntManager.sPrintCnt()
23 b.sPrintCnt()
24
25 CntManager.cPrintCnt()
26 c.cPrintCnt()
```

변경 전

```
1 """
2 staticmethod(function) 형태는 파이썬스럽지 않은 형태의 생성으로
3 최신 버전에는 @staticmethod 형태의 decorator를 사용함
4
5 @staticmethod
6 def func(args,...)
7 """
8 class CntManager:
9     cnt = 0
10    def __init__(self):
11        CntManager.cnt += 1
12
13    @staticmethod
14    def staticPrintCnt():
15        print("Instance cnt:", CntManager.cnt)
16        # self, cls 같은것으로 받는것이 아니기 때문에
17        # class 내의 변수에 접근하고 싶으면 클래스이름.변수로 접근해야함(별로)
18
19    @classmethod
20    def classPrintCnt(cls): # cls 로 class를 전달받음
21        print("Instance cnt:", cls.cnt)
22
23 a, b, c = CntManager(), CntManager(), CntManager()
24
25 CntManager.staticPrintCnt()
26 b.staticPrintCnt()
27
28 CntManager.classPrintCnt()
29 c.classPrintCnt()
```

변경 후

2-1. fd(=File Descriptor)

1. 정의

- 리눅스 혹은 유닉스 계열의 시스템에서 프로세스(Process)가 파일(File)을 다룰 때 사용하는 개념
- 파일 디스크립터(File Descriptor)는 프로세스에서 특정 파일에 접근할 때 사용하는 추상적인 값
파일 디스크립터는 일반적으로 0이 아닌 정수값을 갖는다.
- 프로세스가 이미 존재하는 파일을 `open()` 함수를 이용해 열거나 `creat()` 함수를 이용해 새로운 파일을 생성해달라고 커널에 요청하면 커널은 필요한 동작을 수행하고 파일 디스크립터 값을 리턴해준다.
- 이 파일 디스크립터는 프로세스가 `read()`, `write()` 함수를 수행할 때 인자로 사용되며, 어느 파일에 `read()`, `write()` 요청을 수행할 지를 구분하는 값으로 사용된다.
- 리눅스, 유닉스 시스템에서 일반적으로 0, 1, 2번 파일 디스크립터를 특수한 목적으로 사용한다. 0번 파일 디스크립터는 표준입력(stdin), 1번은 표준 출력(stdout), 2번은 표준 오류(stderr)에 매핑되어 있다. 물론 이 값들은 나중에 `dup()`, `dup2()` 함수나 `fcntl()` 함수 등을 이용해 변경할 수 있다.

2. 요약

- 리눅스에서 파일을 `open`과 같은 함수로 열면 파일 디스크립터(fd)를 리턴
- 프로그램이 파일을 액세스 할때 할당된 파일 디스크립터를 사용
- 리눅스에서는 모든 파일, 하드웨어 장치, 파이프, 소켓 등을 파일로 취급
- 파일 디스크립터 테이블 : 파일 오픈시 시스템은 파일에 대한 정보를 가질 구조체를 할당, 테이블의 인덱스 값이 파일 디스크립터

2-2. 커널소스(task_struct)

1. 커널 소스 확인

① elixir.bootlin 접속

The screenshot shows the elixir.bootlin website interface. At the top, there is a navigation bar with links: HOME, ENGINEERING, TRAINING, DOCS, COMMUNITY, and COMPANY. A banner for "bootlin" with the tagline "Elixir Cross Referencer" is displayed. A sidebar on the left shows a tree view of Linux kernel versions, with "v5.8.9" selected. The main content area displays a list of kernel source files and directories, including Documentation, LICENSES, arch, block, certs, crypto, drivers, fs, include, init, ipc, kernel, lib, mm, net, samples, scripts, security, sound, tools, usr, virt, COPYING, CREDITS, Kbuild, Kconfig, MAINTAINERS, and Makefile. The file sizes for some files are listed: COPYING (496 bytes), CREDITS (99752 bytes), Kbuild (1327 bytes), Kconfig (555 bytes), MAINTAINERS (561445 bytes), and Makefile (61003 bytes).

2-2. 커널소스(task_struct)

1. 커널 소스 확인

② task_struct 검색 -> sched.h 클릭

The screenshot shows the bootlin Elixir Cross Referencer interface. At the top, there's a navigation bar with links: HOME, ENGINEERING, TRAINING, DOCS, COMMUNITY, COMPANY. A banner for 'bootlin Elixir Cross Referencer' is visible. On the left, a sidebar shows a tree view of kernel versions from v2 to v5.9, with v5.8.9 selected. The main content area shows the search results for 'task_struct'. It indicates 'Defined in 4 files:' and lists the following files with line numbers (as a struct):

- include/linux/sched.h, line 629 (as a struct)
- tools/include/linux/lockdep.h, line 26 (as a struct)
- tools/testing/selftests/bpf/progs/strobemeta.h, line 14 (as a struct)
- tools/testing/selftests/bpf/progs/test_core_reloc_kernel.c, line 24 (as a struct)

Below this, it says 'Referenced in 1318 files:' and lists various kernel files and their reference counts, such as arch/alpha/include/asm/elf.h (3 times), arch/alpha/include/asm/machvec.h (2 times), arch/alpha/include/asm/mmu_context.h (4 times), arch/alpha/include/asm/processor.h (3 times), arch/alpha/include/asm/switch_to.h (3 times), arch/alpha/include/asm/syscall.h (line 7), arch/alpha/include/asm/thread_info.h (line 18), arch/alpha/kernel/asm-offsets.c (5 times), arch/alpha/kernel/process.c (6 times), arch/alpha/kernel/proto.h (3 times), arch/alpha/kernel/ptrace.c (11 times), arch/alpha/kernel/smp.c (3 times), arch/alpha/kernel/traps.c (line 147), arch/arc/include/asm/bug.h (2 times), arch/arc/include/asm/current.h (line 16), arch/arc/include/asm/dsp-impl.h (2 times), arch/arc/include/asm/fpu.h (3 times), and arch/arc/include/asm/mmu_context.h (2 times).

2-2. 커널소스(task_struct)

1. 커널 소스 확인

③ task_struct 정의 찾기

```
629 struct task_struct {
630 #ifdef CONFIG_THREAD_INFO_IN_TASK
631     /*
632      * For reasons of header soup (see current_thread_info()), this
633      * must be the first element of task_struct.
634      */
635     struct thread_info      thread_info;
636 #endif
637     /* -1 unrunnable, 0 runnable, >0 stopped: */
638     volatile long           state;
639
640     /*
641      * This begins the randomizable portion of task_struct. Only
642      * scheduling-critical items should be added above here.
643      */
644     randomized_struct_fields_start
645
646     void                    *stack;
647     refcount_t              usage;
648     /* Per task flags (PF_*), defined further below: */
649     unsigned int            flags;
650     unsigned int            ptrace;
651
652 #ifdef CONFIG_SMP
653     int                     on_cpu;
654     struct __call_single_node wake_entry;
655 #ifdef CONFIG_THREAD_INFO_IN_TASK
656     /* Current CPU: */
657     unsigned int            cpu;
658 #endif
659     unsigned int            wakee_flips;
660     unsigned long            wakee_flip_decay_ts;
661     struct task_struct      *last_wakee;
662
```


2-2. 커널소스(files_struct)

1. 커널 소스 확인

- ④ task_struct 정의 내에 files_struct 구조의 포인터 변수로 *files 존재함 확인
- ⑤ files_struct 클릭하여 정의 확인 (include/linux/fdtable.h)

```
916 #ifdef CONFIG_DETECT_HUNG_TASK
917     unsigned long      last_switch_count;
918     unsigned long      last_switch_time;
919 #endif
920     /* Filesystem information: */
921     struct fs_struct     *fs;
922
923     /* Open file information: */
924     struct files_struct  *files;
925
926     /* Namespaces: */
927     struct nsproxy       *nsproxy;
928
929     /* Signal handlers: */
930     struct signal_struct *signal;
931     struct sighand_struct __rcu *sighand;
932     sigset_t             blocked;
933     sigset_t             real_blocked;
934     /* Restored if set_restore_sigmask() was used: */
935     sigset_t             saved_sigmask;
936     struct sigpending     pending;
937     unsigned long        sas_ss_sp;
938     size_t               sas_ss_size;
939     unsigned int          sas_ss_flags;
940
941     struct callback_head  *task_works;
942
943 #ifdef CONFIG_AUDIT
944 #ifdef CONFIG_AUDITSYSCALL
945     struct audit_context  *audit_context;
946 #endif
947 #endif
```

2-2. 커널소스(files_struct)

1. 커널 소스 확인

- ⑥ files_struct 구조체 확인 가능
- ⑦ files_operation 클릭하여 해당 정의 확인 (include/linux/fs.h)

```
45  /*
46   * Open file table structure
47   */
48  struct files_struct {
49      /*
50       * read mostly part
51       */
52      atomic_t count;
53      bool resize_in_progress;
54      wait_queue_head_t resize_wait;
55
56      struct fdtable __rcu *fdt;
57      struct fdtable fdtab;
58
59      /*
60       * written part on a separate cache line in SMP
61       */
62      spinlock_t file_lock ____cacheline_aligned_in_smp;
63      unsigned int next_fd;
64      unsigned long close_on_exec_init[1];
65      unsigned long open_fds_init[1];
66      unsigned long full_fds_bits_init[1];
67      struct file __rcu * fd_array[NR_OPEN_DEFAULT];
68  };
69
70  struct file_operations;
71  struct vfsmount;
72  struct dentry;
73
74  #define rcu_dereference_check_fdt(files, fdt) \
75      rcu_dereference_check((fdt), lockdep_is_held(&(files->file_lock)))
76
77  #define files_fdt(files) \
78      rcu_dereference_check_fdt((files), (files->fdt))
79
80  /*
81   * The caller must ensure that fd table isn't shared or hold rcu or file lock
82   */
83  static inline struct file *__fcheck_files(struct files_struct *files, unsigned int fd)
84  {
85      struct fdtable *fdt = rcu_dereference_raw(files->fdt);
86
87      if (fd < fdt->max_fds) {
88          fd = array_index_nospec(fd, fdt->max_fds);
89          return rcu_dereference_raw(fdt->fd[fd]);
90      }
91      return NULL;
92  }
```

2-2. 커널소스(file_operation)

1. 커널 소스 확인

⑧ file_operations 구조체 정의 확인

- 내부에 함수 포인터 집합들을 확인할 수 있음

```
1837 struct file_operations {
1838     struct module *owner;
1839     loff_t (*llseek) (struct file *, loff_t, int);
1840     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
1841     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
1842     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
1843     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
1844     int (*iopoll) (struct kiocb *kiocb, bool spin);
1845     int (*iterate) (struct file *, struct dir_context *);
1846     int (*iterate_shared) (struct file *, struct dir_context *);
1847     __poll_t (*poll) (struct file *, struct poll_table_struct *);
1848     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
1849     long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
1850     int (*mmap) (struct file *, struct vm_area_struct *);
1851     unsigned long mmap_supported_flags;
1852     int (*open) (struct inode *, struct file *);
1853     int (*flush) (struct file *, fl_owner_t id);
1854     int (*release) (struct inode *, struct file *);
1855     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
1856     int (*fasync) (int, struct file *, int);
1857     int (*lock) (struct file *, int, struct file_lock *);
1858     ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
1859     unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
1860     int (*check_flags) (int);
1861     int (*flock) (struct file *, int, struct file_lock *);
1862     ssize_t (*splice_write) (struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
1863     ssize_t (*splice_read) (struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
1864     int (*setlease) (struct file *, long, struct file_lock **, void **);
1865     long (*fallocate) (struct file *file, int mode, loff_t offset,
1866                       loff_t len);
1867     void (*show_fdinfo) (struct seq_file *m, struct file *f);
1868 #ifdef CONFIG_MMU
1869     unsigned (*mmap_capabilities) (struct file *);
1870 #endif
1871     ssize_t (*copy_file_range) (struct file *, loff_t, struct file *,
1872                               loff_t, size_t, unsigned int);
1873     loff_t (*remap_file_range) (struct file *file_in, loff_t pos_in,
1874                               struct file *file_out, loff_t pos_out,
1875                               loff_t len, unsigned int remap_flags);
1876     int (*fadvise) (struct file *, loff_t, loff_t, int);
1877 } __randomize_layout;
```

llseek

loff_t (*llseek) (struct file *file, loff_t offset, int whence)

파일 포인터를 **offset** 값으로 갱신합니다.

read

ssize_t (*read) (struct file *file, char __user *buf, size_t count, loff_t *offset);

파일 오프셋(**offset**) 위치에서 **count** 바이트만큼 읽습니다. 이 동작을 수행하면서 파일 오프셋인 ***offset**은 업데이트됩니다.

write

ssize_t (*write) (struct file *file, const char __user buf*, size_t count, loff_t offset*);

파일의 오프셋(***offset**) 위치에 **count** 바이트만큼 **buf**에 있는 데이터를 써줍니다.

poll

unsigned int (*poll) (struct file *, struct poll_table_struct *);
파일 동작을 점검하고 파일에 대한 동작이 발생하기 전까지 휴면 상태에 진입합니다.

ioctl

long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);

2-3. 세부구조

1. 아래 사이트에 정리가 굉장히 잘 되어 있는데 몇 번 더 읽고 내용 정리하겠습니다.

<http://rousalome.egloos.com/9994364>

2. 내가 이해한 개괄적인 사항(틀릴 수 있음)

- 큰 틀에서 생각해보면 파일 열기, 생성 등의 함수 호출을 통하여 객체 생성될 때마다 테이블에 파일 디스크립터가 생성 등록되어 앞으로 파일 디스크립터를 통해 프로세스가 해당 파일에 접근한다는 것(즉, 파일 디스크립터가 인터럽트 핸들러 등록하는 거랑 비슷한 개념인 것 같다, Index로 관리하는..., fd_array(파일 객체 테이블))
- 파일 시스템 구조를 보면 유저 영역에서 read(), write() 함수를 유저가 호출하면 시스템 콜을 통해 커널 영역으로 접근하고 커널 함수가 호출되어 동작하게 되며 커널함수는 파일 디스크립터에 해당하는 파일의 함수를 호출하여 동작시키게 됨.(즉, 일종의 추상화 개념으로 생각하면 될 것 같다, 함수 포인터가 반드시 필요하다는게 느껴짐)



감사합니다.