장고 차근차근 시작하기 2/E

당신의 파이썬/장고 페이스메이커가 되겠습니다. ☺

EP-17. 장고 logging과 SQL 로깅 처리

로그

- 특정 형식으로 현 상황을 기록하는 문자열 기록
- 로깅을 파이썬에서 기본 지원
- 로그 레벨
  - DEBUG : 개발환경에서 디버깅을 목적
  - INFO : 분석이 필요한 유용한 상태 정보. ex) 주요 기능의 시작/종료
  - WARNING : 중요도가 낮은 문제를 발생할 가능성. ex) 403 Forbidden, admin 침입시도
  - **ERROR** : 상용 환경의 에러. ex) 500/400 에러
  - **CRITICAL** : 급하게 주의가 요구되는 심각한 상황, ex) 내부 API 서비스에 접근 불가

logging 모듈 (공식문서)

- 파이썬 logging 모듈을 통해 지원
- logging.config.dictConfig 포맷을 사용
  - Loggers : named bucket별 수행할 핸들러 지정
  - Handlers : 핸들러별 수행할 핸들러 클래스 지정
  - Filters : 핸들러가 호출된 조건
  - Formatters : 로그 문자열 포맷
    - LogRecord 속성을 지원

django/utils/log.py

```
# Default logging for Django. This sends an email to the site admins on every
# HTTP 500 error. Depending on DEBUG, all other log records are either sent to
# the console (DEBUG=True) or discarded (DEBUG=False) by means of the
# require_debug_true filter.
DEFAULT_LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse',
        },
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    'formatters': {
        'django.server': {
            '()': 'django.utils.log.ServerFormatter',
            'format': '[{server_time}] {message}',
            'style': '{',
        }
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
        },
        'django.server': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'django.server',
        },
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        }
     },
    'loggers': {
        'django': {
            'handlers': ['console', 'mail_admins'],
            'level': 'INFO',
        },
        'django.server': {
            'handlers': ['django.server'],
            'level': 'INFO',
            'propagate': False,
        },
    }
}
```

named bucket

- 마침표로 parent/child 계층 구분
  - ex) django.security.csrf 로그 : django.security와 django에 전파
- 부모 namespace로의 전파를 막을려면
  - 해당 handler 설정에서 propagate=False 설정
- 장고에서 사용 중인 named bucket
  - django / django.contrib.gis
  - django.db.backends / django.db.backends.schema
  - django.request / django.security.csrf
  - django.server / django.template

예제) 로깅

- named bucket을 지정하여, 현 모듈에서 쓸 logger 객체 획득하고, logger.debug("...") 등으로 로깅 ➔ LOGGING 설정 필요

```python
# myapp/views.py
import logging
logger = logging.getLogger(__name__)  # __name__ => "myapp.views"

def post_list(request):
    logger.error('Something went wrong!')
```

```
λ allieus [Dev/django-dev/django] at  master ✔
→ grep -rn logging.getLogger *
contrib/sessions/backends/db.py:38:                logger = logging.getLogger('django.security.%s' % e.__class__.__name__)
contrib/sessions/backends/file.py:87:                    logger = logging.getLogger('django.security.%s' % e.__class__.__name__)
contrib/sessions/backends/base.py:114:                logger = logging.getLogger('django.security.%s' % e.__class__.__name__)
contrib/gis/geos/libgeos.py:18:logger = logging.getLogger('django.contrib.gis')
contrib/gis/forms/widgets.py:10:logger = logging.getLogger('django.contrib.gis')
contrib/gis/gdal/libgdal.py:10:logger = logging.getLogger('django.contrib.gis')
contrib/gis/admin/widgets.py:11:logger = logging.getLogger('django.contrib.gis')
contrib/gis/db/backends/mysql/schema.py:7:logger = logging.getLogger('django.contrib.gis')
core/handlers/exception.py:74:        security_logger = logging.getLogger('django.security.%s' % exc.__class__.__name__)
core/handlers/base.py:13:logger = logging.getLogger('django.request')
core/servers/basehttp.py:22:logger = logging.getLogger('django.server')
db/backends/utils.py:13:logger = logging.getLogger('django.db.backends')
db/backends/base/schema.py:14:logger = logging.getLogger('django.db.backends.schema')
middleware/csrf.py:21:logger = logging.getLogger('django.security.csrf')
template/base.py:97:logger = logging.getLogger('django.template')
test/runner.py:29:        self.logger = logging.getLogger('django.db.backends')
test/utils.py:674:    logger = logging.getLogger(logger_name)
test/utils.py:818:        self.logger = logging.getLogger('django')
utils/log.py:12:request_logger = logging.getLogger('django.request')
views/generic/base.py:13:logger = logging.getLogger('django.request')
```

주요 logging.handlers.* (공식문서)

- StreamHandler (stream=None) : sys.stdout, sys.stderr 등의 파일객체로의 출력
  - FileHandler (filename, mode='a', encoding=None, delay=False) : 디스크 파일로의 출력.
- NullHandler : 어떠한 포맷팅/출력도 수행하지 않습니다.
- RotatingFileHandler (filename, mode='a', maxBytes=0, backupCount=0, encoding=None, delay=False)
  - maxBytes 단위로 로그 파일을 생성하되, 최대 backupCount 개수만큼만 유지
- TimedRotatingFileHandler (filename, when='h', interval=1, backupCount=0, encoding=None, delay=False, utc=False, atTime=None)
  - 지정 시간 단위로 로그 파일을 생성하되, 최대 backupCount 개수만큼만 유지
- SocketHandler (host, port) : 지정 서버/포트로의 TCP 출력
  - DatagramHandler (host, port) : UDP 소켓
- SysLogHandler (address=('localhost', SYSLOG_UDP_PORT), facility=LOG_USER, socktype=socket.SOCK_DGGRAM)
  - 유닉스의 로그 인터페이스인 syslog 서버로의 출력
- SMTPHandler (mailhost, fromaddr, toaddrs, subject, credentials=None, secure=None, timeout=1.0)
  - SMTP 프로토콜을 활용하여 이메일 발송
- HTTPHandler (host, url, method='GET', secure=False, credentials=None, context=None)
  - GET 혹은 POST 방식으로 출력

예제) SQL 내역을 로깅하기

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'filters': ['require_debug_true'],
            'class': 'logging.StreamHandler',
        },
        'write_to_file': {
            'level': 'DEBUG',
            'filters': ['require_debug_true'],
            'class': 'logging.FileHandler',
            'filename': 'db.log',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['console', 'write_to_file'],
            'level': 'DEBUG',
        },
    }
}
```

```
# PowerShell
Get-Content ./db.log -Wait -Tail 10
```

```
# Mac/Linux Shell
tail -f ./db/log
```


인생은 짧습니다.
파이썬/장고를 쓰세요.