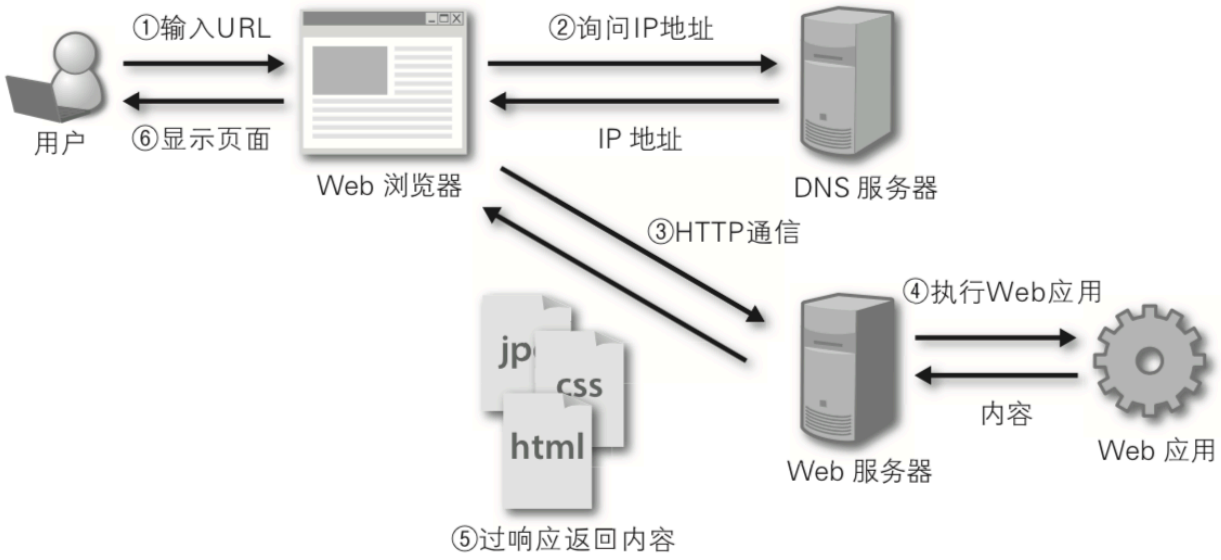
Django知识点概述

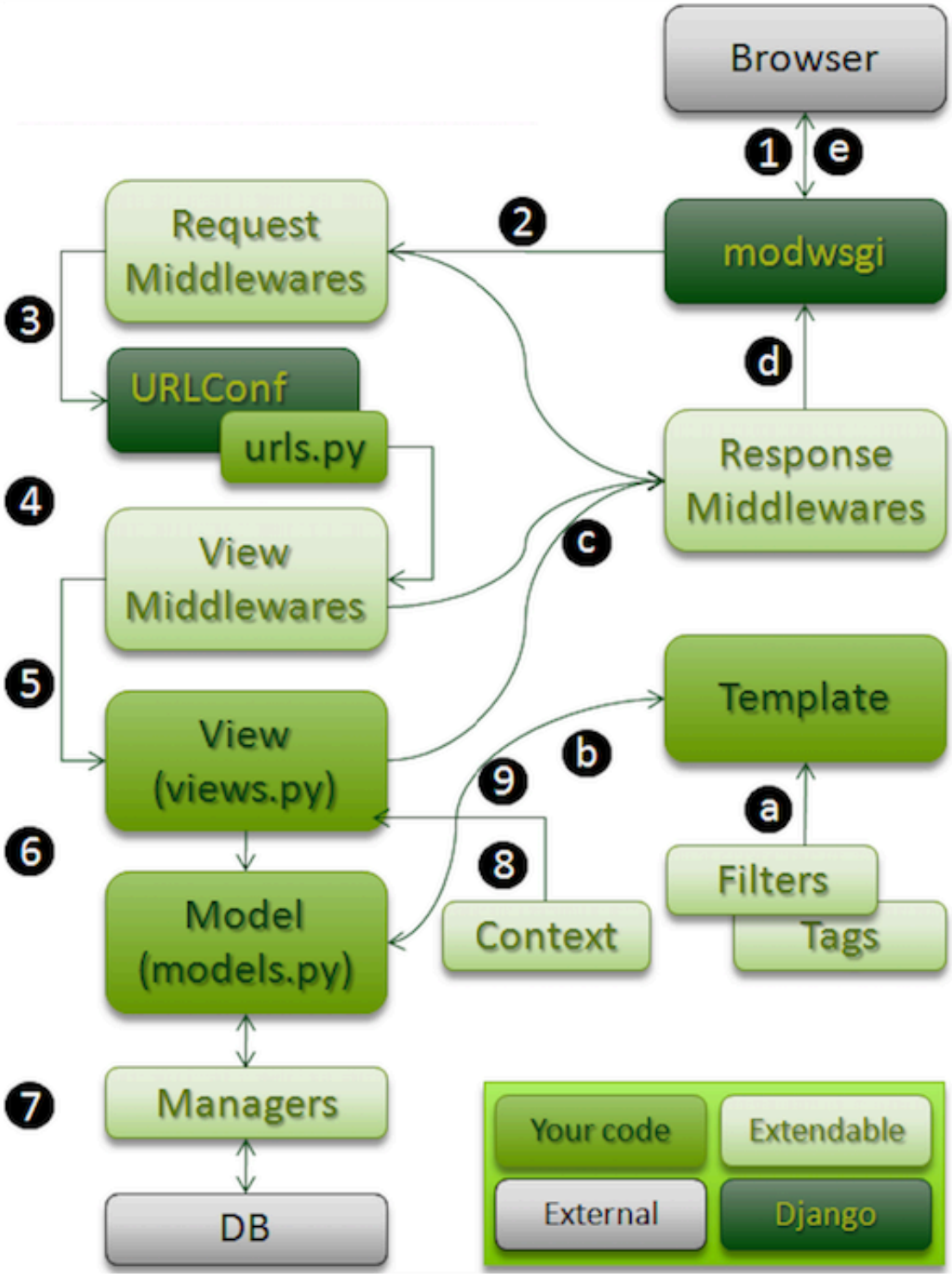
Web应用



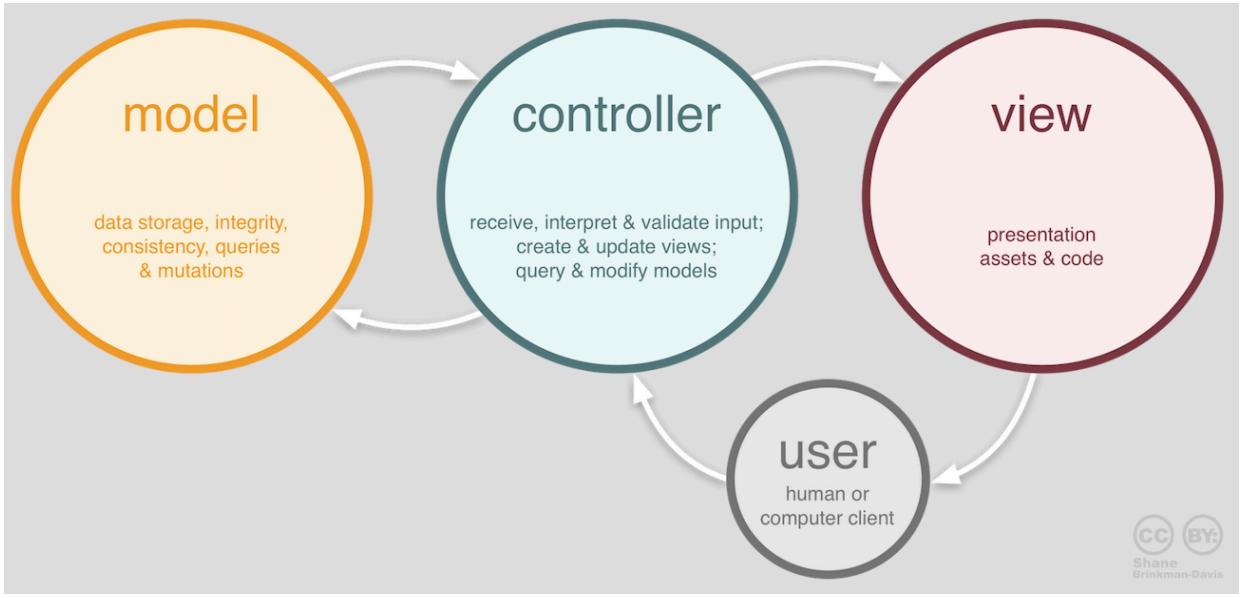
问题1：描述一个Web应用的工作流程。

问题2：描述一下项目的物理架构。

问题3：描述Django项目的工作流程。


Browser
1
e
Request Middlewares
2
modwsgi
3
URLConf
urls.py
d
Response Middlewares
4
View Middlewares
c
5
Template
View (views.py)
b
9
a
6
8
Filters
Model (models.py)
Context
Tags
7
Managers
Your code
Extendable
External
Django
DB

MVC架构模式



问题1：为什么要使用MVC架构模式?

问题2：MVC架构中每个部分的作用?

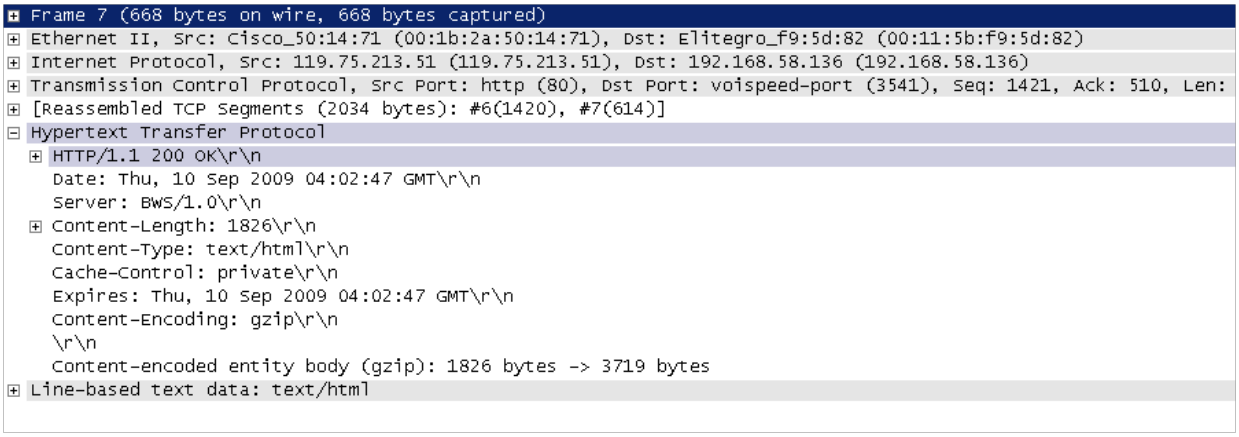
HTTP请求和响应

HTTP请求(请求行+请求头+[空行+消息体])

```
Frame 4 (563 bytes on wire, 563 bytes captured)
Ethernet II, Src: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82), Dst: Cisco_50:14:71 (00:1b:2a:50:14:71)
Internet Protocol, Src: 192.168.58.136 (192.168.58.136), Dst: 119.75.213.51 (119.75.213.51)
Transmission Control Protocol, Src Port: voispeed-port (3541), Dst Port: http (80), Seq: 1, Ack: 1, Len: 509
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: zh-cn\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727)\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: www.baidu.com\r\n
    Connection: Keep-Alive\r\n
    [truncated] Cookie: BAIDUID=72675E110453F51BEAC13B6277CE022F:FG=1; BDLFONT=0; BDUSS=VFJenJQbGZuS21EM1dja3Vyv
    \r\n
```

1. HTTPRequest对象的属性和方法：
    - `method`
    - `path` / `get_full_path()`
    - `scheme` / `is_secure()` / `get_host()` / `get_port()`
    - `META` / `COOKIES`
    - `GET` / `POST` / `FILES`
    - `get_signed_cookie()`
    - `is_ajax()`
    - `body` / `content_type` / `encoding`
2. 中间件添加的属性：
    - `session` / `user` / `site`

HTTP响应(响应行+响应头+空行+消息体)



1. HttpResponse对象的属性和方法：
    - `set_cookie()` / `set_signed_cookie()` / `delete_cookie()`
    - `__setitem__` / `__getitem__` / `__delitem__`
    - `charset` / `content` / `status_code`
2. JsonResponse对象

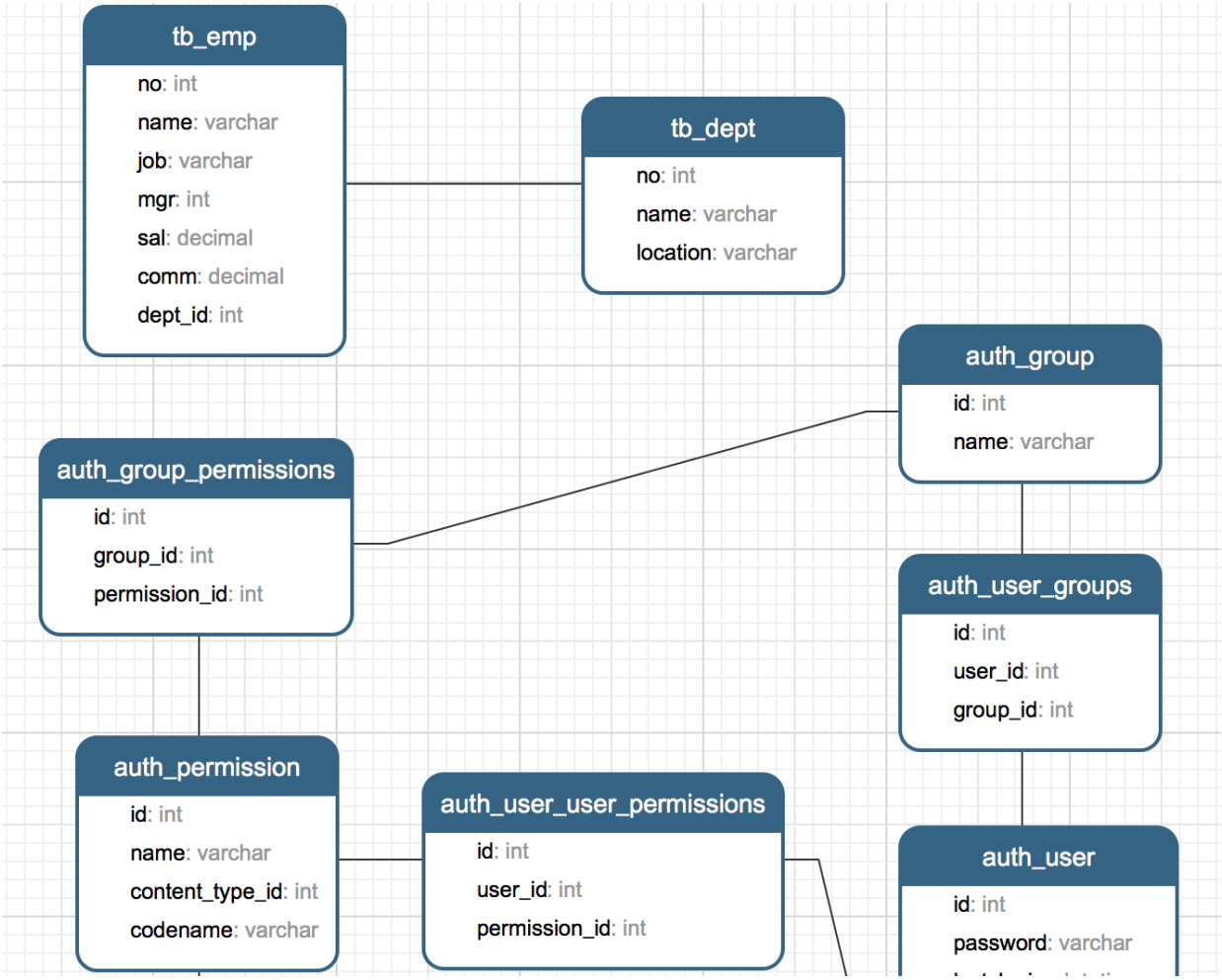
```
>>> from django.http import JsonResponse
>>> response = JsonResponse({'foo': 'bar'})
>>> response.content

>>> response = JsonResponse([1, 2, 3], safe=False)
```

数据模型(Model)

问题1：关系型数据库表的设计应该注意哪些问题?

问题2：关系型数据库中数据完整性指的是什么?

问题3：ORM是什么以及解决了什么问题?



1. Field及其子类的属性：
    - 通用选项：
        - `db_column` / `db_tablespace`
        - `null` / `blank` / `default`
        - `primary_key`
        - `db_index` / `unqiue`
        - `choices` / `help_text` / `error_message` / `editable` / `hidden`
    - 其他选项：
        - CharField：`max_length`
        - DateField: `auto_now` / `auto_now_add`
        - DecimalField: `max_digits` / `decimal_places`
        - FileField: `storage` / `upload_to`
        - ImageField: `height_field` / `width_field`
2. ForeignKey的属性：
    - 重要属性：
        - `db_constraint`
        - `on_delete`
        - `related_name`
    - 其他属性：
        - `to_field`

        - limit_choices_to
        - swappable
3. Model的属性和方法
    - objects
    - pk
    - save() / delete()
    - from_db() / get_XXX_display() / clean() / full_clean()
4. QuerySet的方法
    - get() / all() / values()
    - count() / order_by() / exists() / reverse()
    - filter() / exclude()
        - exact / iexact：精确匹配/忽略大小写的精确匹配查询
        - contains / icontains / startswith / istartswith / endswith / iendswith：基于like`的模糊查询
        - in：集合运算
        - gt / gte / lt / lte：大于/大于等于/小于/小于等于关系运算
        - range：指定范围查询（SQL中的between…and…）
        - year / month / day / week_day / hour / minute / second：查询时间日期
        - isnull：查询空值（True）或非空值（False）
        - search：基于全文索引的全文检索
        - regex / iregex：基于正则表达式的模糊匹配查询
        - aggregate() / annotate()
        - Avg / Count / Sum / Max / Min
    - first() / last()
    - only() / defer()
    - create() / update() / raw()
5. Q对象和F对象

```
>>> from django.db.models import Q
>>> Emp.objects.filter(
...     Q(name__startswith='张'),
...     Q(sal__lte=5000) | Q(comm__gte=1000)
... ) # 查询名字以“张”开头且工资小于等于5000或补贴大于等于1000的员工
<QuerySet [<Emp: 张三丰>]>
```

```
reporter = Reporters.objects.filter(name='Tintin')
reporter.update(stories_filed=F('stories_filed') + 1)
```

视图函数(Controller)

如何设计视图函数

1. 用户的每个操作对应一个视图函数。
2. 每个视图函数构成一个事务边界。
    - 事务的概念。
    - 事务的ACID特性。
    - 事务隔离级别。
    - Django中的事务控制。
        - 给每个请求绑定事务环境（反模式）。

        ```
        ATOMIC_REQUESTS = True
        ```

        - 使用事务装饰器。

        ```
        @transaction.non_atomic_requests
        @transaction.atomic
        ```

        - 使用上下文语法。

        ```
        with transaction.atomic():
        ```

        - 关闭自动提交。

        ```
        AUTOCOMMIT = False
        ```

        ```
        transaction.commit()
        transaction.rollback()
        ```

URL配置

1. 调试模式下生效的URL。
2. 使用命名捕获组。
3. URL配置不关心请求使用的方法。
4. 如果使用`url`函数捕获的参数都是字符串。
5. 可以使用`include`函数引入其他URL配置，捕获的参数会向下传递。
6. 在`url`和`path`函数甚至是`include`函数中都可以用字典向视图传入额外的参数，如果参数与捕获的参数同名，则使用字典中的参数。
7. 可以用`reverse`函数实现URL的逆向解析（从名字解析出URL），在模板中也可

以用{% url %}实现同样的操作。

模板(View)

后端渲染

1. 模板的配置和渲染函数。

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'), ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

 'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

```
resp = render(request, 'foo.html', {'foo': 'bar'})
```

2. 模板遇到变量名的查找顺序。
    - 字典查找（foo['bar']）
    - 属性查找（foo.bar）
    - 方法调用（foo.bar()）
        - 方法不能又必须的参数
        - 在模板中不能够给方法传参
        - 如果方法的alters_data被设置为True则不能调用，模型对象动态生成的delete()和save()方法都设定了alters_data = True。
    - 列表索引查找（foo[0]）
3. 模板标签的使用。
    - {% if %} / {% else %} / {% endif %}
    - {% for %} / {% endfor %}
    - {% ifequal %} / {% endifequal %} / {% ifnotequal %} / {% endifnotequal %}
    - {# comment #} / {% comment %} / {% endcomment %}

4. 过滤器的使用
    - `lower` / `upper` / `first` / `last` / `truncatewords:<n>` / `date`/ `time` / `length` / `pluralize` / `ljust` / `rjust` / `add` / `capfirst` / `center` / `cut:<ch>` / `title` / `urlencode` / `default_if_none:<string>` / `filesizeformat` / `join:<ch>` / `slice:<ch>` / `slugify` / `wordwrap` / `yesno`
5. 模板的包含和继承
    - `{% include %}` / `{% block %}`
    - `{% extends %}`
6. 模板加载器
7. 自定义模板指令（不需要掌握，浪费时间）。

前端渲染

1. 前端模板引擎。
2. 前端MV*框架。

其他视图

1. 如何处置生成的内容。

    ```
    response['content-disposition'] = 'attachment;
    filename=xyz.abc'
    ```

2. 生成CSV / Excel / PDF。
    - 大文件的流式处理：`StreamingHttpResponse`。
    - 生成PDF：需要安装`reportlab`。
    - 生成Excel：需要安装`openpyxl`。

中间件

问题1：中间件背后的设计理念是什么?

问题2：中间件有哪些不同的实现方式?

```
def simple_middleware(get_response):

    def middleware(request):

        response = get_response(request)

        return response

    return middleware
```

```
class MyMiddleware:

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):

        response = self.get_response(request)

        return response

    def process_view(self, request, view_func, view_args,
view_kwargs):
        response = view_func(*view_args, **view_kwargs)
        return response
```

```
from django.utils.deprecation import MiddlewareMixin


class MyMiddleware(MiddlewareMixin):

    def process_request(self, request):
        pass

    def process_view(self, request, view_func, view_args,
view_kwargs):
        pass

    def process_response(self, request, response):
        return response
```

问题3：阐述中间件的执行顺序。

问题4：描述Django内置的中间件的执行顺序。

Django官方文档 - 中间件 - 中间件顺序。

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'common.middlewares.block_sms_middleware',
]
```

表单

1. 用法：通常不要用来生成页面上的表单控件（耦合度太高不容易定制），主要用来验证数据。
2. Form的属性和方法：
    - `is_valid()` / `is_multipart()`
    - `errors` / `fields` / `is_bound` / `changed_data` / `cleaned_data`
    - `add_error()` / `has_errors()` / `non_field_errors()`
    - `clean()`
3. Form.errors的方法：
    - `as_data()` / `as_json()` / `get_json_data()`

问题1：Django中的Form和ModelForm有什么作用？

问题2：表单上传文件时应该注意哪些问题？

Cookie和Session

问题1：使用Cookie能解决什么问题？

问题2：Cookie和Session之间关系是什么？

Session的配置

1. Session对应的中间件：`django.contrib.sessions.middleware.SessionMiddleware`。
2. Session引擎。
    - 基于数据库

```
INSTALLED_APPS = [
    'django.contrib.sessions',
]
```

- 基于缓存

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
SESSION_CACHE_ALIAS = 'default'
```

- 基于文件（基本不考虑）
- 基于Cookie

```
SESSION_ENGINE =
'django.contrib.sessions.backends.signed_cookies'
```

2. Cookie相关的配置。

```
SESSION_COOKIE_NAME =
SESSION_COOKIE_AGE =
SESSION_EXPIRE_AT_BROWSER_CLOSE =
SESSION_SAVE_EVERY_REQUEST =
SESSION_COOKIE_HTTPONLY = True
```

3. session的属性和方法。
   - `session_key` / `session_data` / `expire_date`
   - `__getitem__` / `__setitem__` / `__delitem__` / `__contains__`
   - `set_expiry()` / `get_expiry_age()` / `get_expiry_date()`
   - `flush()`
   - `set_test_cookie()` / `test_cookie_worked()` / `delete_test_cookie()`
4. session的序列化。

```
SESSION_SERIALIZER =
```

   - JsonSerializer（默认）
   - PickleSerializer（关于这种方式的安全漏洞，请参考[《Python Pickle的任意代码执行漏洞实践和Payload构造》]。）

缓存

1. 配置缓存。

```
CACHES = {
```

```
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': [
            'redis://120.77.222.217:6379/0',
            'redis://120.77.222.217:6380/0',
            'redis://120.77.222.217:6381/0',
            # 'redis://120.77.222.217:6382/0',
        ],
        # 'KEY_PREFIX': 'fang',
        # 'TIMEOUT': 120,
        'OPTIONS': {
            'CLIENT_CLASS':
'django_redis.client.DefaultClient',
            'CONNECTION_POOL_KWARGS': {
                'max_connections': 100,
            },
            # 'PASSWORD': '1qaz2wsx',
            # 'COMPRESSOR':
'django_redis.compressors.zlib.ZlibCompressor'
        }
    }
}
```

2. 全站缓存。

```
MIDDLEWARE_CLASSES = [
    'django.middleware.cache.UpdateCacheMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.cache.FetchFromCacheMiddleware',
]

CACHE_MIDDLEWARE_ALIAS =
CACHE_MIDDLEWARE_SECONDS =
CACHE_MIDDLEWARE_KEY_PREFIX =
```

3. 视图层缓存。

```
from django.views.decorators.cache import cache_page


@cache_page()
def my_view(request):
```

```
from django.views.decorators.cache import cache_page

urlpatterns = [
    url(r'^foo/([0-9]{1,2})/$', cache_page(60 * 15)(my_view)),
]
```

4. 模板片段缓存。
    - {% load cache %}
    - {% cache %} / {% endcache %}
5. 使用底层API访问缓存。

```
>>> from django.core.cache import cache
>>> cache.set('my_key', 'hello, world!', 30)
>>> cache.get('my_key')
>>> cache.clear()
```

```
>>> from django.core.cache import caches
>>> cache1 = caches['myalias']
>>> cache2 = caches['myalias']
>>> cache1 is cache2
True
```

日志

日志级别

NOTSET < DEBUG < INFO < WARNING < ERROR < FETAL

日志配置

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'simple': {
            'format': '%(asctime)s %(module)s.%(funcName)s: %
(message)s',
            'datefmt': '%Y-%m-%d %H:%M:%S',
        },
        'verbose': {
            'format': '%(asctime)s %(levelname)s [%(process)d-%
(threadName)s] '
```

```
                      '%(module)s.%(funcName)s line %(lineno)d: %
(message)s',
            'datefmt': '%Y-%m-%d %H:%M:%S',
        }
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'level': 'DEBUG',
        },
        'inf': {
            'class': 'logging.handlers.TimedRotatingFileHandler',
            'filename': 'info.log',
            'when': 'W0',
            'backupCount': 7,
            'formatter': 'simple',
            'level': 'INFO',
        },
        'err': {
            'class': 'logging.handlers.TimedRotatingFileHandler',
            'filename': 'error.log',
            'when': 'D',
            'backupCount': 31,
            'formatter': 'verbose',
            'level': 'WARNING',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
        'inform': {
            'handlers': ['inf'],
            'level': 'DEBUG',
            'propagate': True,
        },
        'error': {
            'handlers': ['err'],
            'level': 'DEBUG',
            'propagate': True,
        }
    }
}
```

日志配置官方示例。

日志分析

1. Linux相关命令：head、tail、grep、awk、uniq、sort

```
tail -1000 access.log | awk '{print $1}' | uniq -c | sort -r
```

2. 实时日志文件分析：Python + 正则表达式 + Crontab
3. 《Python日志分析工具》。

RESTful

问题1：RESTful架构到底解决了什么问题?

问题2：你在项目中是如何应用RESTful架构的?

安全保护

Django提供的安全保护

1. 跨站脚本攻击
2. 跨站身份伪造
3. SQL注射攻击
4. 点击劫持攻击
5. CSRF令牌和小工具

```
{% csrf_token %}
```

```
@csrf_exempt
@csrf_protect
@require_csrf_token
@ensure_csrf_cookie
```

配置中的敏感信息

用户敏感信息的保护

1. 哈希摘要
2. 加密和解密

配置HTTPS

性能相关

网站优化两大定律：

1. 尽可能的使用缓存 - 牺牲空间换取时间（普适策略）。
2. 能推迟的都推迟 - 使用消息队列将并行任务串行来缓解服务器压力。

Django框架

1. 配置缓存来缓解数据库的压力，并有合理的机制应对缓存穿透和缓存雪崩。
2. 开启模板缓存来加速模板的渲染。

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'), ],
        # 'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

 'django.contrib.messages.context_processors.messages',
            ],
            'loaders': [(
                'django.template.loaders.cached.Loader', [

 'django.template.loaders.filesystem.Loader',

 'django.template.loaders.app_directories.Loader',
                ], ),
            ],
        },
    },
]
```

3. 用惰性求值、迭代器、`defer()`、`only()`等缓解内存压力。

数据库

1. 用ID生成器代替自增主键（性能更好、适用于分布式环境）。
2. 避免不必要的外键列（除非必须保证参照完整性），更不要使用触发器之类的机制。
3. 使用索引来优化查询性能。
4. 使用存储过程。
5. 使用`explain`来分析查询性能。

6. 使用慢查询日志来发现性能低下的查询。

其他

（待补充）