1. Apacche 是老牌的静态服务器，在并发功能上有所欠缺，同时处理1万个并发请求吃力。Nignx是俄罗斯开发的开源新的静态服务器，更加轻便有高并发性。 静态服务器是指服务器设计本身是返回静态内容，但由于可以通过服务器后面的例如Python的语言产生动态网页。 Tomcat是是编译JSP\Servlet的容器，设计本身是通过提供给JSP运行环境提供动态网页。
2. Perl的telnet模块去telnet另一台机器的执行一个指令的时候，timeout为超时描述。如果telnet到另一台机器上执行某个脚本，且该脚本每隔一段时间都有信息打印，那么该telnet仍然会超时。Timeout意味着telnet的指令要在该timeout范围内完成。
3. Subprocess模块的call方法是阻塞方式，在call的方法完成后才继续执行。Popen方法是非阻塞性方法，会直接继续执行语句。
4. 反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后再将请求转发给内部网络上的服务器。而不让internet上的链接请求直接让内部网络上的服务器接收处理。
5. Nagios是一款开源的免费网络监视工具，能有效监控Windows、Linux和Unix的[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA" \t "_blank)状态，交换机[路由器](https://baike.baidu.com/item/%E8%B7%AF%E7%94%B1%E5%99%A8)等网络设备，打印机等。在系统或服务状态异常时发出邮件或短信报警第一时间通知网站运维人员，在状态恢复后发出正常的邮件或短信通知。
6. Django 在处理网页错误404,500,403的时候，有自己原生的view处理也可以使用自定义的view处理。基本过程是，当DEBUG = True的时候， Django不能设置成使用指定的view处理404,500,403，错误, Django会直接将错误信息，以及错误栈显示在网页上供调试。

而当DEBUG=False的时候，Django将回去处理这些错误并且根据原生或者自定义的网页封装错误信息给用户。

--在DEBUG=FALSE下，Django原生的处理HTTP错误为404使用默认的[page\_not\_found](https://docs.djangoproject.com/en/stable/ref/views/" \l "the-404-page-not-found-view)，处理500使用默认的[server\_error](https://docs.djangoproject.com/en/stable/ref/views/" \l "the-500-server-error-view)，处理403使用[permission\_denied](https://docs.djangoproject.com/en/stable/ref/views/" \l "the-403-http-forbidden-view)，处理400使用[bad\_request](https://docs.djangoproject.com/en/stable/ref/views/" \l "the-400-bad-request-view)。对应的将返回到404.html,500.html,403.html,400.html中进行渲染，返还给用户。并且发送邮件。

--所以我们在自定义错误界面的时候，可以有两个方面。第一我们可以自定义错误函数即是custom error handler函数（例如handle505=某自定义函数，从而不使用server\_error函数，server\_error默认的行为是基本没什么行为，不会给500.HTML提供任何变量，以保证不会在生成错误界面的时候再产生错误）。另一方面我们可以自己写模板，从而产生自定义的错误页面。我们可以选择自定义一个方面，可以自定义两个方面。需要注意的是如果只自定义了HTML，例如提供了500.HTML而不去写自定义的handle500，那么在500.html中是无法使用任何变量信息的，因为默认的server\_error不会给模板传入任何信息）

1. Django的用户认证在settings.py中的[AUTHENTICATION\_BACKENDS](https://docs.djangoproject.com/en/1.9/ref/settings/" \l "std:setting-AUTHENTICATION_BACKENDS) 进行认证。Django自带的认证为django.contrib. auth.backends.ModelBackend。 我们还可以添加其他认证例如LDAP认证。 当我们向认证中加入多个认证时，一个用户登录时，Django会从上到下认证。如果成功则马上返回，如果不成功则到下一个认证backend去试图认证。 登陆成功后，用户使用什么backend进行的认证信息，会保存在会话(session)中。
2. Django中如果以Database作为session的backend.那么如果用户手动logout退出，对应的Session想会被清除。如果用户直接关闭浏览器，那么session将永远留在数据库中。Django提供一个自动清除的命令为. /manage.py clearsessions
3. Django的session都需要通过SessionMiddleWave中间件实现。然后我们选择Session的方式。Django有四种Session方式，分别是database-backed session(Django最常用的方式，也是默认方式), cached session, file-based session, 还有cookie-based session。
4. Django中可以设置两种风格的Sessioin失效时间。默认情况下是持久性的会话（Persistent sessions）意思是当用户关闭浏览器以后，服务器上的Session不会失效直到失效时间到达。我们也可以通过 [SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE](https://docs.djangoproject.com/en/2.0/ref/settings/#std:setting-SESSION_EXPIRE_AT_BROWSER_CLOSE)这个属性去设置，当浏览器关闭的时候，会话马上失效。
5. Django自带的登陆验证系统中，有自带的User类，这个类含有一些基本的属性。例如用户名，用户密码，最后登陆日期，权限,所属组，是否active等。除此之外如果我们还想定义一些User额外的属性，可以以AbstractBBaseUser作为基类继续自行扩展。
6. LDAP（Lightweight Directory Access Protocol）协议是 轻量化的访问控制协议。默认情况下，标准端口非保密认证使用389，保密认证例如SSL/TLS通信采用636端口。
7. 市面上有许多产品实现了LDAP协议，例如UnboundID Directory Server， Oracle Internet Directory等
8. LDAP有两种方式从LDAP服务器中去寻找被验证的用户名和密码。

第一种是Search/Bind另外一种是Direct Binding. Search/Bind的过程为首先我们连接到LDAP服务器，我们从给定的参数路径中（例如"ou=users,dc=example,dc=com"）去寻找用户id然后使用密码去验证。在连接到LDAP服务器的时候如果我们不设置AUTH\_LDAP\_BIND\_DN和AUTH\_LDAP\_BIND\_PASSWORD，那么我们作为匿名去连接LDAP服务器，搜索需要被验证的用户。如果我们设置AUTH\_LDAP\_BIND\_DN和AUTH\_LDAP\_BIND\_PASSWORD则我们作为认证用户去LDAP服务器中搜索被验证的用户。

第二种是直接绑定（Direct Binding）

例如AUTH\_LDAP\_USER\_DN\_TEMPLATE="uid=%(user)s,ou=users,dc=example,dc=com"，这样的话我们变等于直接在com🡪example🡪ou🡪uid找寻用户，指定了路径不需要LDAP搜索。而且我们也可以看出越后的指定的节点越高层，最后一个节点为根节点。

搜索集合（Search Unions）

例如：

import ldap

from django\_auth\_ldap.config import LDAPSearch, LDAPSearchUnion

AUTH\_LDAP\_USER\_SEARCH = LDAPSearchUnion(

LDAPSearch("ou=users,dc=example,dc=com", ldap.SCOPE\_SUBTREE, "(uid=%(user)s)"),

LDAPSearch("ou=otherusers,dc=example,dc=com", ldap.SCOPE\_SUBTREE, "(uid=%(user)s)"),

)

1. 我们在这里使用LDAPSearch Union而不是LDAPSe arch去返回用户。LDAPSearch只能指一条路径搜索用户，而LDAPSearch Union内可以指定多条LDAPSe arch的搜索路径。

备注： dc: Domain Component , DN: Distinguished Name , ou: Organizational unit

1. LDAP中的 base dn 为LDAP Server指定查找用户的起点。
2. Python的try…except模块，一般在except模块阶段，使用except ExceptionClass, e. 其中Exception为具体的异常类型，如果直接Exception则是捕获所有异常。e是异常的实例，从而可以在except块中填入e.message。如果直接随意的except: 那么只是隐藏了异常，让程序不报错，没有关于异常的任何信息，不利于调试。
3. Django中每次进行post请求以后，服务器都会赋予一个新的CSRF值，此次如果使用以前的csrf值再次提交则会出错。
4. Python的psutil模块可以直接查看机器的内存情况，CPU情况，磁盘情况等。
5. Django的 DateTimeField 和DateField有属性为，auto\_now以及auto\_now\_add. Auto\_now是每次当此entry进行save()的时候，自动更改该事件域，auto\_now\_add是每次创建的时候更改该时间域。需要注意的是设置为auto\_now或者auto\_now\_add以后，不能后自行通过save去保存，最多只能通过update()进行手动更改。
6. Celery消除特定队列的消息：

celery -Q queue\_name purge

celery purge # 清除所有队列的消息

23. 给用户发送邮件的时候，除了一般的纯文字以外，还可以加入HTML内容使得页面更具有交互性。但是注意，邮件中加入HTML并不能和单独写一个单独的HTML一样，因为用户的邮件系统各异，每个邮件系统有自己的过滤条件。他们无法像正常浏览器那样解析邮件中的HTML语言，主要的编写原则为：

1. 使用style属性来编写样式，邮件不接受外联，内联, class这样的css写法。

2. img 一般邮件系统都会默认不显示img元素，所以尽量不用。

3. float, position这些让元素脱离标准流的语句邮件系统都不会认。要布局就是用table元素。

4. javascript语句都不认。

24. Django将数据字典传给模板产生Html页面，我们也可以通过语句获得template语句然后让他使用数据字典渲染，然后得到渲染后的字符串结果。

25. Django组织将会选择某些Django版本，并且对他们进行特别长时间的支持。这些选中的版本为LTS(Long-Term Support)例如feature号为1.11的Django版本为LTS.

26. Django的版本格式为A.B.C, 其中A.B为feature版本编号例如1.9,1.10。feature号大概每八个月更新一次， C为patch版本例如1.9.4中的4。

所以1,9.4为feature号是1.9,patch编号为4.而1.9.5同样是1.9feautre号，但是patch号为5.

27. Django某些特性的淘汰策略为，在同一A版本中将兼容，在下一个A+1版本中将淘汰。例如，在4.2中淘汰feature F, 那么4.2, 4.3, 4.4中都会有警告信息，5.0为最后一个兼容的版本,5.1以后将无法使用。

28. Django在每个app下面，即在./manage.py startapp xx的时候，会在app的目录下创建一个tests.py文件。这个文件将会被./manage.py test自动pickup，去执行一些测试。

29.一般情况下，一些特性语法过期的提示 (deprecation warnings) 会被隐藏，我们需要在执行. /manage.py test的时候，告诉Django进行汇报，不要静默。我们可以有两种方法，第一种是python -Wall manage.py test做一次自动化测试，且通过-Wall暴露出警告信息或者是设置环境变量PYTHONWARNINGS=all xx.test tests –capture=no去执行。

30. 升级前，我们最好通过所有测试，并且解决所有deprecation warnings，然后升级。

31. Django在生产环境中，我们将DEBUG=False, 这样会加快网站的速度。但此时如果出错，我们便看不到DEBUG的也面。所以我们一般是设置DEBUG=FALSE，以后配置EMAIL\_HOST，SERVER\_EMAIL，ADMINS字段在settings.py。这样如果用户在使用网站出错的时候，用户看到的是500Internal Error, 但是Django会将出错的信息发送到我们填入的邮箱，我们便可以马上收到出错的信息进行改错。

32. 在Django数据库语法中，表名(条件) 等于 表名.objects.get\_or\_create(条件)

33. assertEqual(A, B)方法。如果A与B相等则不报错，如果A与B不相等则报错。

34. 执行Django框架的自动测试时：

python manage.py test APP名。

Django会自动寻找App名下的tests.py。

Django会自动执行App名下的类是django.test. Testcase子类中的，以test名字开头的方法并且执行。

Django在执行测试的时候，会创建一个沙盒数据库，在test方法中需要改变数据库中的资料都在这个沙盒数据库中，所以不会影响外面的数据库。并且Django在执行每个测试方法的时候，都会重置一次沙盒数据库。并且需要注意的是，也正是因为使用的是沙盒数据库，所以无法在生产数据库中取出任何数据，在Test中除非自己创建数据才能搜索得到，如果直接去检索数据那么肯定是空的。

Django自动化测试的最后通常会写入一句assertxxx条件作为判断测试是否成功的标准。例如assertEqual, assertContains, assertQuerysetEqual等

Django自动化测试还可以模拟用户访问某个url （使用django.test.Client）,检测是否可以成功获得返回的网页。

### 35.Django的值APPEND\_SLASH 默认为True, 意思是如果在urls.py中没有任何一个url可以匹配的时候，将会在请求的url地址后面加上一个/然后再进行一次匹配。（注意的是，如果在最后有一个万能匹配，然后导向到404页面，那么这个将不生效，因为在第一次Django认为有一个可以匹配就是万能匹配，不会在其后面加上/再次匹配。）

### 36. Static File与Template File在Django中是两个不同的东西。有各自的设置。

### 通常使用Django所推崇的部署静态文件的方式是：

### 首先：Static File在部署服务器上，通过在app中注册使用django.contrib.staticfiles来启动使用。

然后：将静态文件有明显的属于某个APP的放置在各个App的static/文件夹下，将共有的文件放在额外的静态文件中，在settings.py中的字段为STATICFILES\_DIRS

### 然后：在settings.py中指定STATIC\_URL。这个字段的主要作用是在试图加载静态文件的时候，给静态文件加个前缀。更重要的功能是，将这个前缀设置在Apache 的Alias 中，将这个前缀与STATIC做成一个别名。

### 然后进行./manage.py collectstatic进行部署。Collectstatic会将STATIC\_DIRS所设置的额外目录或者在settings.py中INSTALLED\_App所列出的所有app中的static/文件夹下的文件或者文件夹拷贝到STATIC\_ROOT所指定的绝对路径中。

### 从而在加载静态文件的时候，由于前缀的存在,Apache会mapping到STATIC\_DIRS寻找文件。

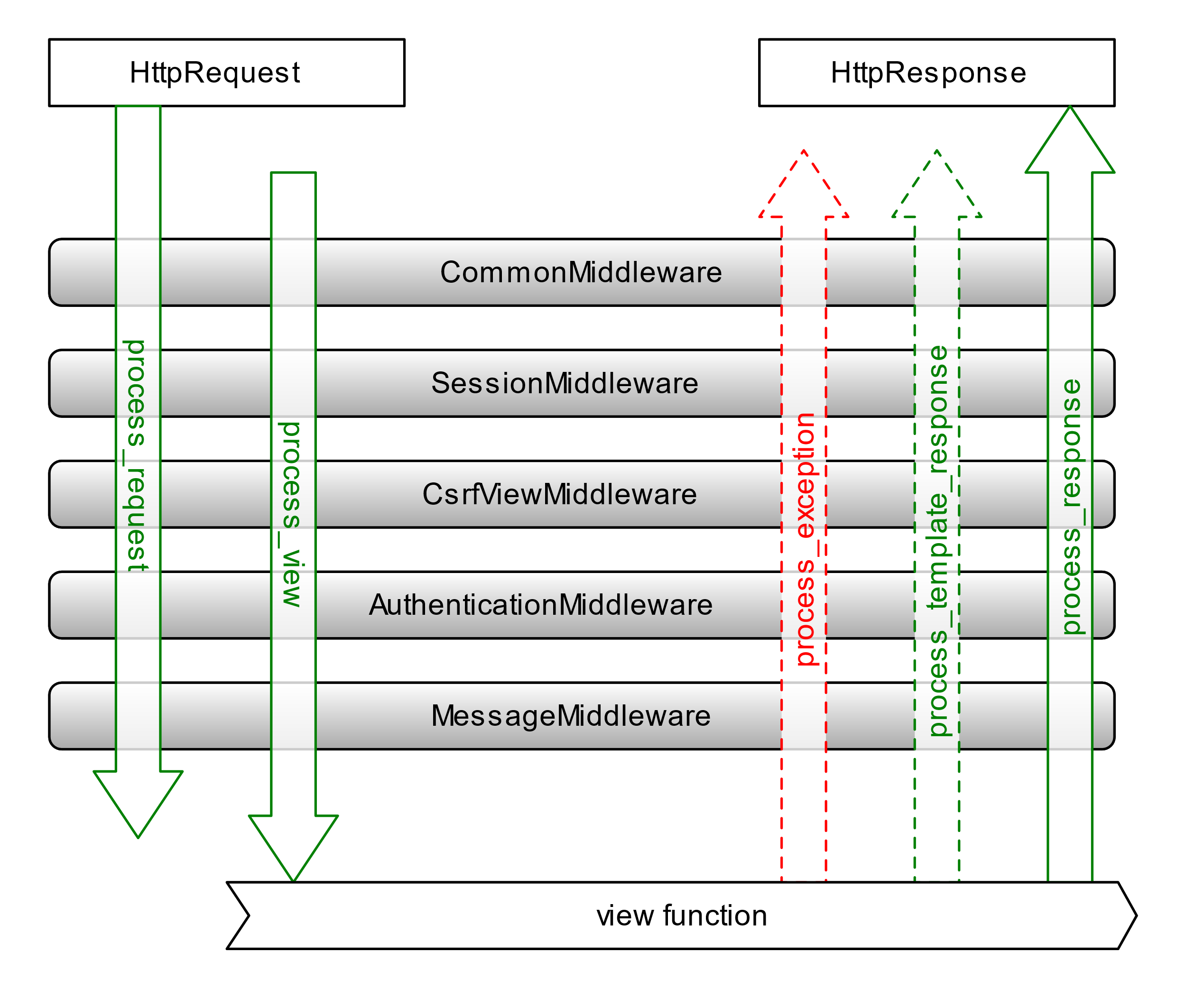
### 从而在扩展其他Django控件，例如DRF的时候，在INSTALLED\_APP中注册组件以后，一执行collectstatics变可以将DRF自带的CSS,JS文件（用于browsers api）自动拷贝到STATIC\_ROOT中，而不是手动拷贝或者再从网上下载。与此同时，当我们改变我们自己编写的JS文件以后，可需要collectstatics以后，才会放到serverd的静态文件中去。

### 37.当django.template.context\_processors.static启动时，RequestContext才会带有STATIC\_URL

### 38.Celery如果不指定队列，那么queue的名称为celery.

### 39.Django中间件

### Django中间件的流程为：



最底层为view方法。首先HttpRequest从最顶层进入，然后这个request按在INSTALL\_APPS定义的中间件顺序被处理，逐层传递。如果某个一层中间件需要的变量，在上一层提供，那么上一层的中间件在INSTALLED\_APPS中定义必须在下一层的前面。到达view以后，将返回response，response按逆顺序经过中间层返回到Browser中。

--钩子方法 (Hook function),在中间件的传递中，由HttpRequest到view方法中有两个勾子方法，从View返回response到Browser当中有3个钩子方法。钩子方法可以定义在中间件中，并且钩子方法只会在对应的过层中出发，例如process\_exception不会再从httpRequest到view中触发，只会在view返回response到browser触发。钩子方法的行为可以对request或者response对象进行加工，然后返回到下一层或者上一层，也可以直接截断传递到view或者browser中，跳过剩余的层数。

--中间件其实就是一个Python的类，然后加工的行为就是在这个类中定义特定的方法而已。

### --

### process\_exception()方法，这个方法在响应从view到browser的阶段时，产生异常时会被触发。这个方法的返回类型有两种，如果是None,则将会继续执行默认的异常处理，例如发送404邮件，500邮件等。如果返回是一个响应对象，那么该方法所在的中间件，再向上的中间件将不会被经过。

--

在Debug=False,默认处理中，Django使用默认的模板或者在模板文件夹中文件名为500.html的模板，不传达任何context参数（为了减少生成500页面时候的错误）。如果Debug-False的时候，不会有这样的行为。并且Django只当500.html只是个普通模板，不会在异常的时候默认调用。

40.Django Cache Framework。

Django提供了一个Cache Framework这个Framework的作用在于，对于生成一些耗时的页面的时候，可以根据上次的运算结果直接给出结果，而不需要每个HTTPREQUEST都要经历views.py中的计算，template渲染，从而减少计算的时间以及资源。

Django提高了多种层次，不同方式的Cache方式。在层次上，我们可以为某个app进行cache, 对某个特定的view进行cache对整个site进行cache。在方式上，我们可以选择将cache的结果放在内存中，将cache的结果放在数据库中，对cache的结果放在文件中等等。不同的cache方式会有不同的响应速度，对cache的设置放置在settings.py中。

41. 如果使用Database Cache:

在settings.py中，设置如下内容：

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.db.DatabaseCache',

'LOCATION': 'my\_cache\_table',

}

}

其中BACKED的值表示使用Django中的Database Cache,另外location的值表明，将会在数据库中创建名为my\_cache\_table的表作为存放cache的表。

最后使用python manage.py createcachetable 命令，在数据库中创建Cache表。

***P. S*** *上面我们只对Cache进行了最简单的设置，我们还可以对*Cache进行更加精确的设置。例如可以在cache中存放多少个条目（默认为300）, 默认的TIMEOUT时间（以秒为单位，默认为300秒，None则为永不过期，0代表永远过期，其实就是弃用Cache）等等的选项。并且不同通的Cache将会有各自特别的选项可以设置。详情参照文档。

42.经过41的设置以后，我们设置以什么backend作为cache的存储。现在我们还要指定让什么作为Cache.

如果我们让某个view函数以Cache的形式进行响应的时候，我们有两种方式。第一种是在view函数上以@cache\_page修饰器进行修饰，第二种是我们在urls.py文件对url进行指定的时候加上可选参数cache\_page. 这种方式可以使得cache系统的代码完全独立于view函数，使view函数的代码更加整洁。

43.在manage.py shell的交互模式下我们还可以使用from django.core.cache import cache中的cache对象，来获取访问Django项目中的cache.进行添加或者删除。

44. 有时候Django的网页无反应，是因为后台的数据库中的其中一个数据库进程卡住。这个进程阻止了所有其他SQL进程的执行。需要手动删除。

45.Template默认情况下，在Django中每次接受HTTP请求以后，都会在硬盘读取然后渲染然后返回给用户。所以模板的改变不需要重启服务器。但是在1.11以后，在DEBUG=FALSE的情况下，Django会默认开启为模板缓存，从而短时间下一次请求相同页面模板的时候，不需要再到硬盘读取，同样的，这样改变template以后需要重启，清除缓存。

46.

在Django模板中使用，extends标记去扩展某个基本模板，使用includes标记来加入某个扩展模板到本模板中。扩展模板中和原模版一样，可以含有HTML代码，可以有JS代码。并且扩展模板中所能读到的数据一般是传给本模板中的数据。

在Views函数中，可以引入一个新的python文件，那个文件内的函数接受request参数，返回http响应。那么这个引入的文件中的python函数也可以处理Http请求，相当于从外部扩充了Views.py函数。适合做plugins的请求响应。

47.

django.setup()函数能够让我们独立的Python Script去加载Django的模板以及通过Django的ORM去访问我们的数据库。

if \_\_name\_\_ == '\_\_main\_\_':

import django

django.setup()

48.

Python的默认添加寻找路径是本目录，但是不包含上层目录。

如果我们需要扩展寻找的引入路径时候，

1.可以使用sys.path.append()去扩展目录。这样需要对每个运行的脚本都需要添加。

2.直接改变本机器的默认寻找目录，方法为：

在site-packages添加一个路径文件，如mypkpath.pth

使用编辑器打开，比如我用的是notepad++， 写上你要加入的模块文件所在的目录名称，多个使用换行即可

检查是否成功，在python下>>>import sys>>>sys.path

49.

Django中null=True与blank=True的区别：

null=True是Django ORM实际将数据存储到数据库的时候用的，表达此域在数据库中是否可以为空。

blank=True是Django做表单认证的时候用的，只是表达Django在验证含有此域表单的时候，是否可以不传值。

50.

Django数据库变化：

Makemigrations将会为检查并且生成所有App的Migrations

Makemigrations [app\_lable.] 将只会生成app所对应的app以及所依赖的migrations

P.s如果通过通过makemigrations后，各个App下的migrations目录以及数据库中的django\_migrations应该一致。此时如果通过外部的SQL文件导入数据时候，如果包含创建表格语句，还是会让数据库中的表格更改导致不一致情况。

51.

Django根据测试服务器以及生产服务器的个性化配置。

Django没有专门为测试服务器和生产服务器进行特殊的配置机制。但是由于配置文件settings.py是由Python写成，所以已经提高的足够的自由度。我们可以在settings.py中根据不同需要，以Python的语法引入额外的配置，例如import \* from …settings\_local.py。也可以在settings.py中根据不同情况，例如先用Python获取hostname然后根据变量，覆盖原始设置等等。

52.

Django选用数据库的时候，如果选用了MySQL并且以MyISAM作为数据引擎(例如采用了MySQL5.5.5版本以下，默认使用MyISAM作为数据引擎) 的时候，由于MyISAM并不支持外键限制，所以外键限制只会通过Django的ORM层来保证。如果我们采用Django ORM语句来操作数据库，则可以保持外键限制，但是如果直接使用SQL语句或者通过其他方式绕过ORM对数据进行改变，没有了Django的ORM逻辑，那么外键限制不会存在。

Django在2.2版本以后支持MySQL 5.6以后包括MySQL8, Django的1.11版本支持MySQL 5.5.x – 5.7.x, MySQL8不支持。

53.

Django默认情况下使用的是Django特有的模板语言，Django template language (DTL)， 虽然语法和Jinja2很像但是不是Jinja2.例如DTL中自带模板|dictsort和Jinja2的|dictsort用法不一样。

Django可以配置成使用Jinja2而不适用自带的模板语言。

54.