TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Manejando los parámetros en el enrutamiento.
- Configuración del enrutamiento.
- Procesamiento del enrutamiento.

PASO DE PARÁMETROS EN EL ENRUTAMIENTO

1. *Agregar un archivo de enrutamiento de base de datos.*

Para usar múltiples bases de datos en Django, solo especifique cada una en `settings.py`:

```
DATABASES = {
 'default': {
 'NAME': 'app_data',
 'ENGINE': 'django.db.backends.postgresql',
 'USER': 'django_db_user',
 'PASSWORD': os.environ['LOCAL_DB_PASSWORD']
 },
 'users': {
 'NAME': 'remote_data',
 'ENGINE': 'django.db.backends.mysql',
 'HOST': 'remote.host.db',
 'USER': 'remote_user',
 'PASSWORD': os.environ['REMOTE_DB_PASSWORD']
 }
}
```

Utilice un archivo `dbrouters.py` para especificar qué modelos deben operar en qué bases de datos para cada clase de operación. Por ejemplo, para datos remotos almacenados en `remote_data`, es posible que desee lo siguiente:

```
class DbRouter(object):
 """
 A router to control all database operations on models in the
 auth application.
 """
 def db_for_read(self, model, **hints):
 """
```

```
    Attempts to read remote models go to remote database.
    """
    if model._meta.app_label == 'remote':
    return 'remote_data'
    return 'app_data'
    def db_for_write(self, model, **hints):
    """
    Attempts to write remote models go to the remote database.
    """
    if model._meta.app_label == 'remote':
    return 'remote_data'
    return 'app_data'
    def allow_relation(self, obj1, obj2, **hints):
    """
    Do not allow relations involving the remote database
    """
    if obj1._meta.app_label == 'remote' or \
    obj2._meta.app_label == 'remote':
    return False
    return None
    def allow_migrate(self, db, app_label, model_name=None, **hints):
    """
    Do not allow migrations on the remote database
    """
    if model._meta.app_label == 'remote':
    return False
    return True
```

Finalmente, agregue su `dbrouter.py` a `settings.py`:

```
DATABASE_ROUTERS = ['path.to.DbRouter', ]
```

Para especificar diferentes bases de datos en el código:

El `obj.save()` normal `obj.save()` usará la base de datos predeterminada, o si se usa un enrutador de base de datos, usará la base de datos como se especifica en `db_for_write`. Puedes anularlo usando:

```
obj.save(using='other_db')
obj.delete(using='other_db')
```

Del mismo modo, para la lectura:

```
MyModel.objects.using('other_db').all()
```

2. *Enrutamiento de URL*

Después de definir, convertir, pasar parámetros, y nombrar una URL determinada, el proceso de encontrar la función de procesamiento correspondiente es asociar la URL y la función de procesamiento.

CONFIGURACIÓN DE ENRUTAMIENTO DE DJANGO

La variable `ROOT_URLCONF` en el archivo settings.py especifica el nombre del archivo de enrutamiento global. Este es el archivo de enrutamiento de entrada del proyecto.

```
ROOT_URLCONF = "<nombre del proyecto> .urls"
```

En el archivo `urls.py`, use la variable `urlpatterns` para indicar el enrutamiento. La variable es un tipo de lista, y el elemento está compuesto por: `path()` o `re_path()`. `ruta()` maneja el enrutamiento de cadenas. `re_path()` maneja el enrutamiento de expresiones regulares.

PROCESO DE ENRUTAMIENTO DE DJANGO

1. Busque la variable `urlpatterns` en el archivo de enrutamiento global.
2. Según el orden, haga coincidir los elementos en `urlpatterns` con la URL, uno por uno.
3. Después de encontrar el primer elemento coincidente, dejará de buscar, y ejecute la función de procesamiento en función del resultado coincidente.
4. Si no se encuentra una coincidencia, o se produce una excepción, Django hará el manejo de errores.

El enrutamiento de Django no considera el método de solicitud HTTP, solo las rutas basadas en la URL. Siempre que la URL sea la misma, `POST` o `GET` apuntarán a la misma función de procesamiento. Pero puede hacer juicios básicos antes de procesar funciones a través de decoradores.

```
from django.views.decorators.http import require_http_methods
 @require_http_methods(["GET", "POST"])
def index(request):
        pass
```

RUTA DE LA FUNCIÓN DE ENRUTAMIENTO()

La función `path()` tiene cuatro parámetros; dos obligatorios: ruta y vista; y dos opcionales: `kwargs` y `name`.

- `route:` patrón de coincidencia de URL. Cuando Django responde a una solicitud, hará coincidir elementos según la ruta desde el primer elemento de `urlpatterns`.
- `view:` el nombre de la función de procesamiento. Cuando Django encuentra un patrón coincidente, llamará a esta función de procesamiento, y pasará un objeto `HttpRequest`.
- `kwargs:` se puede pasar cualquier número de parámetros de palabras clave a la función de procesamiento de destino, como un diccionario.
- `name:` nombra el patrón de URL.

TRES FORMATOS DE RUTA

1. Formato de cadena exacto, similar a: `artículos / 2003 /`

Una URL precisa coincide con una función de procesamiento, adecuada para responder a URL estáticas.

2. Formato de conversión de Django, similar a: `artículos / <int: año> /`

Una plantilla de URL, mientras coincide con la URL, obtiene un lote de variables como parámetros y la pasa a la función de procesamiento.

3. Formato de expresión regular, similar a: `artículos /? P <año> [0-9] {4} /`

Con la ayuda de la rica sintaxis de las expresiones regulares, se puede expresar un tipo de URL (no una), y las variables se pueden extraer como parámetros de la función de procesamiento a través de `<>`.

Dos tipos de vista:

- Función de procesamiento.
- La función `include()`. Al contener otra información de enrutamiento, la ruta segmentada se combina para formar la ruta total.