Django сигналы позволяют различным частям приложения сообщать друг другу о событиях. Это механизм обратного вызова, который позволяет одной части кода реагировать на изменения в другой части. Вот основные компоненты и этапы работы с сигналами в Django:

**Основные компоненты сигналов**

1. **Сигналы**: События, на которые можно подписаться. Django предоставляет встроенные сигналы, и вы также можете создать свои собственные.

Примеры встроенных сигналов:

* + django.db.models.signals.pre\_save
  + django.db.models.signals.post\_save
  + django.db.models.signals.pre\_delete
  + django.db.models.signals.post\_delete

1. **Получатели (receivers)**: Функции, которые выполняются в ответ на сигналы.
2. **Отправка сигналов**: Сигналы могут быть отправлены при выполнении определенных действий в коде.

**Пример использования сигналов**

**Шаг 1: Импорт необходимых модулей**

python

Копировать код

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.contrib.auth.models import User

**Шаг 2: Определение функции-получателя**

python

Копировать код

@receiver(post\_save, sender=User)

def user\_saved(sender, instance, created, \*\*kwargs):

if created:

print(f'Новый пользователь создан: {instance}')

else:

print(f'Пользователь обновлен: {instance}')

**Шаг 3: Подключение функции-получателя к сигналу**

Функция user\_saved автоматически подключается к сигналу post\_save для модели User благодаря декоратору @receiver.

**Создание собственных сигналов**

Вы также можете создать свои собственные сигналы:

**Шаг 1: Импортирование необходимых модулей**

python

Копировать код

from django.dispatch import Signal

**Шаг 2: Определение своего сигнала**

python

Копировать код

my\_signal = Signal(providing\_args=["instance", "created"])

**Шаг 3: Определение функции-получателя и подключение к сигналу**

python

Копировать код

@receiver(my\_signal)

def my\_handler(sender, \*\*kwargs):

instance = kwargs['instance']

created = kwargs['created']

if created:

print(f'Новый объект создан: {instance}')

else:

print(f'Объект обновлен: {instance}')

**Шаг 4: Отправка сигнала**

python

Копировать код

from django.dispatch import receiver

# Отправка сигнала

my\_signal.send(sender=MyModel, instance=my\_instance, created=True)

**Важные моменты**

* **Декораторы**: Использование декораторов (@receiver) помогает избежать ошибок при подключении функций-получателей к сигналам.
* **Аргументы**: Обратите внимание на аргументы, передаваемые в функцию-получатель. Они зависят от типа сигнала.
* **Очистка**: Не забудьте отсоединить сигнал, если он больше не нужен, чтобы избежать утечек памяти.

**Заключение**

Django сигналы — мощный инструмент для обеспечения реактивности приложения и упрощения логики обратных вызовов.

***js***

**1. Методы объектов**

* **Object.create(proto, [propertiesObject])**: Создает новый объект с указанным прототипом и свойствами.
* **Object.assign(target, ...sources)**: Копирует значения всех перечисляемых свойств из одного или более исходных объектов в целевой объект.
* **Object.keys(obj)**: Возвращает массив имен собственных перечисляемых свойств объекта.
* **Object.values(obj)**: Возвращает массив значений собственных перечисляемых свойств объекта.
* **Object.entries(obj)**: Возвращает массив пар [key, value] собственных перечисляемых свойств объекта.
* **Object.freeze(obj)**: Замораживает объект, предотвращая добавление, удаление или изменение свойств.
* **Object.seal(obj)**: Запечатывает объект, предотвращая добавление новых свойств и делая все существующие свойства не конфигурируемыми.
* **Object.getOwnPropertyDescriptor(obj, prop)**: Возвращает дескриптор свойства для указанного свойства объекта.
* **Object.defineProperty(obj, prop, descriptor)**: Определяет новое или изменяет существующее свойство непосредственно на объекте, возвращая объект.

**2. Методы массивов**

* **Array.from(arrayLike, [mapFn, [thisArg]])**: Создает новый массив из массивоподобного или итерируемого объекта.
* **Array.isArray(obj)**: Проверяет, является ли объект массивом.
* **Array.of(element0, element1, ..., elementN)**: Создает новый массив с переменным числом элементов, независимо от числа или типа аргументов.
* **[].concat(array2, array3, ..., arrayN)**: Объединяет два или более массива.
* **[].every(callback(element[, index[, array]])[, thisArg])**: Проверяет, удовлетворяют ли все элементы массива условию, заданному в передаваемой функции.
* **[].filter(callback(element[, index[, array]])[, thisArg])**: Создает новый массив со всеми элементами, прошедшими проверку в передаваемой функции.
* **[].forEach(callback(currentValue[, index[, array]])[, thisArg])**: Выполняет переданную функцию один раз для каждого элемента массива.
* **[].indexOf(searchElement[, fromIndex])**: Возвращает первый индекс, по которому данный элемент может быть найден в массиве, или -1, если элемент отсутствует.
* **[].join([separator])**: Объединяет все элементы массива в строку.
* **[].map(callback(currentValue[, index[, array]])[, thisArg])**: Создает новый массив с результатами вызова указанной функции для каждого элемента массива.
* **[].pop()**: Удаляет последний элемент из массива и возвращает его значение.
* **[].push(element1, ..., elementN)**: Добавляет один или более элементов в конец массива и возвращает новую длину массива.
* **[].reduce(callback(accumulator, currentValue[, index[, array]])[, initialValue])**: Применяет функцию к аккумулятору и каждому значению массива (слева направо) для свертки его до одного значения.
* **[].shift()**: Удаляет первый элемент из массива и возвращает его значение.
* **[].slice([begin[, end]])**: Возвращает новый массив, содержащий копию части исходного массива.
* **[].some(callback(element[, index[, array]])[, thisArg])**: Проверяет, удовлетворяет ли какой-либо элемент массива условию, заданному в передаваемой функции.
* **[].sort([compareFunction])**: Сортирует элементы массива на месте и возвращает отсортированный массив.
* **[].splice(start[, deleteCount[, item1[, item2[, ...]]]])**: Изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые.
* **[].unshift(element1, ..., elementN)**: Добавляет один или более элементов в начало массива и возвращает новую длину массива.

**3. Методы строк**

* **String.charAt(index)**: Возвращает символ, стоящий на указанной позиции.
* **String.charCodeAt(index)**: Возвращает числовой код символа, стоящего на указанной позиции.
* **String.concat(string2, ..., stringN)**: Объединяет текст из двух или более строк и возвращает новую строку.
* **String.includes(searchString[, position])**: Определяет, содержит ли одна строка другую строку.
* **String.indexOf(searchValue[, fromIndex])**: Возвращает индекс первого вхождения указанного значения, или -1, если не найдено.
* **String.lastIndexOf(searchValue[, fromIndex])**: Возвращает индекс последнего вхождения указанного значения, или -1, если не найдено.
* **String.match(regexp)**: Соответствует строке с регулярным выражением.
* **String.replace(searchValue, replaceValue)**: Заменяет часть строки на другую строку.
* **String.search(regexp)**: Ищет совпадения между регулярным выражением и строкой.
* **String.slice(beginIndex[, endIndex])**: Извлекает часть строки и возвращает новую строку.
* **String.split([separator[, limit]])**: Разбивает объект String на массив строк.
* **String.startsWith(searchString[, position])**: Определяет, начинается ли строка с символов другой строки.
* **String.substr(start[, length])**: Возвращает указанное количество символов из строки, начиная с указанной позиции.
* **String.substring(indexStart[, indexEnd])**: Возвращает подстроку строки между двумя индексами.
* **String.toLowerCase()**: Преобразует строку в нижний регистр.
* **String.toUpperCase()**: Преобразует строку в верхний регистр.
* **String.trim()**: Удаляет пробельные символы с начала и конца строки.

**4. Методы чисел**

* **Number.isFinite(value)**: Определяет, является ли переданное значение конечным числом.
* **Number.isInteger(value)**: Определяет, является ли переданное значение целым числом.
* **Number.isNaN(value)**: Определяет, является ли переданное значение NaN.
* **Number.isSafeInteger(value)**: Определяет, является ли значение безопасным целым числом (в пределах безопасного диапазона чисел в JavaScript).
* **Number.parseFloat(string)**: Преобразует строку в число с плавающей запятой.
* **Number.parseInt(string[, radix])**: Преобразует строку в целое число.

**5. Методы работы с датами**

* **Date.now()**: Возвращает числовое значение, соответствующее текущему времени (количество миллисекунд с 1 января 1970 года 00:00:00 UTC).
* **Date.parse(dateString)**: Разбирает строковое представление даты и возвращает количество миллисекунд с 1 января 1970 года 00:00:00 UTC.
* **Date.UTC(year, month[, day[, hour[, minute[, second[, millisecond]]]]])**: Возвращает число, представляющее собой количество миллисекунд с 1 января 1970 года 00:00:00 UTC.
* **date.getDate()**: Возвращает день месяца.
* **date.getDay()**: Возвращает день недели (0 для воскресенья, 1 для понедельника и т.д.).
* **date.getFullYear()**: Возвращает год.
* **date.getHours()**: Возвращает часы.
* **date.getMilliseconds()**: Возвращает миллисекунды.
* **date.getMinutes()**: Возвращает минуты.
* **date.getMonth()**: Возвращает месяц (0 для января, 1 для февраля и т.д.).
* **date.getSeconds()**: Возвращает секунды.
* **date.getTime()**: Возвращает числовое значение времени (количество миллисекунд с 1 января 1970 года 00:00:00 UTC).
* **date.getTimezoneOffset()**: Возвращает разницу в минутах между местным и UTC временем.
* **date.setDate(day)**: Устанавливает день месяца.
* **date.setFullYear(year[, month[, day]])**: Устанавливает полный год.
* **date.setHours(hours[, minutes[, seconds[, milliseconds]]])**: Устанавливает часы.
* **date.setMilliseconds(milliseconds)**: Устанавливает миллисекунды.
* **date.setMinutes(minutes[, seconds[, milliseconds]])**: Устанавливает минуты.
* **date.setMonth(month[, day])**: Устанавливает месяц.
* **date.setSeconds(seconds[, milliseconds])**: Устанавливает секунды.
* **date.setTime(time)**: Устанавливает числовое значение времени (количество миллисекунд с 1 января 1970 года 00:00:00 UTC).

**6. Методы работы с регулярными выражениями**

* **RegExp.exec(string)**: Выполняет поиск соответствия в строке и возвращает результат в виде массива или null.
* **RegExp.test(string)**: Проверяет соответствие строки регулярному выражению и возвращает true или false.

**1. Встроенные функции**

* **abs(x)**: Возвращает абсолютное значение числа.
* **all(iterable)**: Возвращает True, если все элементы в итерируемом объекте истинны (или если итерируемый объект пустой).
* **any(iterable)**: Возвращает True, если хотя бы один элемент в итерируемом объекте истинный.
* **bin(x)**: Преобразует целое число в двоичную строку.
* **bool([x])**: Преобразует значение в булево значение.
* **bytes([source[, encoding[, errors]]])**: Возвращает новый объект bytes.
* **callable(object)**: Проверяет, является ли объект вызываемым (функцией, методом и т.д.).
* **chr(i)**: Возвращает строку, представляющую символ Unicode для указанного целого числа.
* **classmethod(function)**: Преобразует метод в метод класса.
* **compile(source, filename, mode, flags=0, dont\_inherit=False, optimize=-1)**: Компилирует исходный код в объект кода.
* **complex([real[, imag]])**: Создает комплексное число.
* **delattr(object, name)**: Удаляет атрибут объекта.
* **dict()**: Создает новый словарь.
* **dir([object])**: Возвращает список атрибутов и методов объекта.
* **divmod(a, b)**: Возвращает пару (a // b, a % b).
* **enumerate(iterable, start=0)**: Возвращает объект enumerate.
* **eval(expression[, globals[, locals]])**: Выполняет выражение Python из строки.
* **exec(object[, globals[, locals]])**: Выполняет динамическое выполнение Python-кода.
* **filter(function, iterable)**: Конструирует итератор из элементов, для которых функция возвращает истину.
* **float([x])**: Преобразует строку или число в число с плавающей запятой.
* **format(value[, format\_spec])**: Форматирует значение.
* **frozenset([iterable])**: Возвращает неизменяемый объект frozenset.
* **getattr(object, name[, default])**: Возвращает значение атрибута объекта.
* **globals()**: Возвращает словарь текущего пространства имен глобальной области видимости.
* **hasattr(object, name)**: Проверяет, имеет ли объект указанный атрибут.
* **hash(object)**: Возвращает хэш значение объекта.
* **help([object])**: Вызов встроенной системы справки.
* **hex(x)**: Преобразует целое число в шестнадцатеричную строку.
* **id(object)**: Возвращает "адрес" объекта в памяти.
* **input([prompt])**: Запрашивает ввод пользователя.
* **int([x[, base]])**: Преобразует строку или число в целое число.
* **isinstance(object, classinfo)**: Проверяет, является ли объект экземпляром указанного класса или кортежа классов.
* **issubclass(class, classinfo)**: Проверяет, является ли класс подклассом указанного класса или кортежа классов.
* **iter(object[, sentinel])**: Возвращает итератор для объекта.
* **len(s)**: Возвращает длину объекта.
* **list([iterable])**: Создает новый список.
* **locals()**: Возвращает словарь текущего пространства имен локальной области видимости.
* **map(function, iterable, ...)**: Применяет функцию к каждому элементу итерируемых объектов и возвращает итератор с результатами.
* **max(iterable, \*[, key, default])**: Возвращает наибольший элемент.
* **memoryview(obj)**: Возвращает объект memory view.
* **min(iterable, \*[, key, default])**: Возвращает наименьший элемент.
* **next(iterator[, default])**: Возвращает следующий элемент итератора.
* **object()**: Возвращает новый, базовый объект.
* **oct(x)**: Преобразует целое число в восьмеричную строку.
* **open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)**: Открывает файл и возвращает соответствующий объект файла.
* **ord(c)**: Возвращает числовое представление символа.
* **pow(x, y[, z])**: Возвращает (x \*\* y) % z.
* **print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)**: Выводит объекты в поток вывода.
* **property(fget=None, fset=None, fdel=None, doc=None)**: Возвращает свойство.
* **range([start,] stop[, step])**: Возвращает объект range.
* **repr(object)**: Возвращает строковое представление объекта.
* **reversed(seq)**: Возвращает обратный итератор.
* **round(number[, ndigits])**: Округляет число до указанного количества десятичных знаков.
* **set([iterable])**: Возвращает новый объект set.
* **setattr(object, name, value)**: Устанавливает значение атрибута объекта.
* **slice(stop)** или **slice(start, stop[, step])**: Возвращает объект slice.
* **sorted(iterable, \*, key=None, reverse=False)**: Возвращает новый отсортированный список.
* **staticmethod(function)**: Преобразует метод в статический метод.
* **str([object])**: Возвращает строковое представление объекта.
* **sum(iterable, /, start=0)**: Возвращает сумму элементов итерируемого объекта.
* **super([type[, object-or-type]])**: Возвращает объект прокси для базового класса.
* **tuple([iterable])**: Возвращает новый объект tuple.
* **type(object)** или **type(name, bases, dict)**: Возвращает тип объекта или создает новый тип.
* **vars([object])**: Возвращает **dict** атрибут объекта.
* **zip(\*iterables)**: Создает итератор, который агрегирует элементы из каждой из переданных итерируемых объектов.

**2. Методы строк**

* **str.capitalize()**: Преобразует первый символ строки в верхний регистр.
* **str.casefold()**: Преобразует строку в нижний регистр (более агрессивно, чем str.lower()).
* **str.center(width[, fillchar])**: Возвращает отцентрованную строку.
* **str.count(sub[, start[, end]])**: Возвращает количество неперекрывающихся вхождений подстроки.
* **str.encode(encoding="utf-8", errors="strict")**: Кодирует строку в байты.
* **str.endswith(suffix[, start[, end]])**: Проверяет, заканчивается ли строка указанным суффиксом.
* **str.expandtabs(tabsize=8)**: Возвращает копию строки, где все символы табуляции заменены пробелами.
* **str.find(sub[, start[, end]])**: Возвращает наименьший индекс, по которому находится подстрока.
* \*\*str.format(\*args, **kwargs)**: Выполняет форматирование строки.
* **str.format\_map(mapping)**: Выполняет форматирование строки с использованием словаря.
* **str.index(sub[, start[, end]])**: Возвращает наименьший индекс, по которому находится подстрока, вызывает ValueError, если подстрока не найдена.
* **str.isalnum()**: Проверяет, состоит ли строка только из буквенно-цифровых символов.
* **str.isalpha()**: Проверяет, состоит ли строка только из буквенных символов.
* **str.isascii()**: Проверяет, состоит ли строка только из ASCII символов.
* **str.isdecimal()**: Проверяет, состоит ли строка только из десятичных символов.
* **str.isdigit()**: Проверяет, состоит ли строка только из цифровых символов.
* **str.isidentifier()**: Проверяет, является ли строка допустимым идентификатором.
* **str.islower()**: Проверяет, все ли символы строки в нижнем регистре.
* **str.isnumeric()**: Проверяет, состоит ли строка только из числовых символов.
* **str.isprintable()**: Проверяет, все ли символы строки печатные или строка пустая.
* **str.isspace()**: Проверяет, состоит ли строка только из пробельных символов.
* **str.istitle()**: Проверяет, является ли строка заглавной.
* **str.isupper()**: Проверяет, все ли символы строки в верхнем регистре.
* **str.join(iterable)**: Возвращает строку, являющуюся конкатенацией строк в итерируемом объекте.
* **str.ljust(width[, fillchar])**: Возвращает выровненную по левому краю строку.
* **str.lower()**: Возвращает строку в нижнем регистре.
* **str.lstrip([chars])**: Возвращает копию строки с удаленными начальными символами.
* **str.maketrans(x[, y[, z]])**: Возвращает таблицу сопоставлений символов для использования в методе translate().
* **str.partition(sep)**: Разделяет строку на три части вокруг указанного разделителя.
* **str.replace(old, new[, count])**: Возвращает копию строки с заменами.
* **str.rfind(sub[, start[, end]])**: Возвращает наибольший индекс, по которому находится подстрока.
* **str.rindex(sub[, start[, end]])**: Возвращает наибольший индекс, по которому находится подстрока, вызывает ValueError, если подстрока не найдена.
* **str.rjust(width[, fillchar])**: Возвращает выровненную по правому краю строку.
* **str.rpartition(sep)**: Разделяет строку на три части вокруг указанного разделителя, начиная с конца строки.
* **str.rsplit(sep=None, maxsplit=-1)**: Разделяет строку на список строк.
* **str.rstrip([chars])**: Возвращает копию строки с удаленными конечными символами.
* **str.split(sep=None, maxsplit=-1)**: Разделяет строку на список строк.
* **str.splitlines([keepends])**: Разделяет строку на список строк, разрываемых по символам перевода строки.
* **str.startswith(prefix[, start[, end]])**: Проверяет, начинается ли строка с указанного префикса.
* **str.strip([chars])**: Возвращает копию строки с удаленными начальными и конечными символами.
* **str.swapcase()**: Возвращает строку с измененным регистром.
* **str.title()**: Возвращает строку с заглавными начальными буквами каждого слова.
* **str.translate(table)**: Возвращает копию строки с замененными символами по таблице сопоставлений.
* **str.upper()**: Возвращает строку в верхнем регистре.
* **str.zfill(width)**: Возвращает строку, дополненную нулями слева до указанной ширины.

**3. Методы списков**

* **list.append(x)**: Добавляет элемент в конец списка.
* **list.extend(iterable)**: Расширяет список, добавляя все элементы итерируемого объекта.
* **list.insert(i, x)**: Вставляет элемент на указанную позицию.
* **list.remove(x)**: Удаляет первый элемент из списка, значение которого равно x.
* **list.pop([i])**: Удаляет элемент в указанной позиции и возвращает его.
* **list.clear()**: Удаляет все элементы из списка.
* **list.index(x[, start[, end]])**: Возвращает наименьший индекс, по которому находится указанный элемент.
* **list.count(x)**: Возвращает количество вхождений элемента в списке.
* **list.sort(\*, key=None, reverse=False)**: Сортирует элементы списка на месте.
* **list.reverse()**: Разворачивает элементы списка на месте.
* **list.copy()**: Возвращает копию списка.

**4. Методы словарей**

* **dict.clear()**: Удаляет все элементы из словаря.
* **dict.copy()**: Возвращает копию словаря.
* **dict.fromkeys(iterable, value=None)**: Создает новый словарь с ключами из итерируемого объекта и значением по умолчанию.
* **dict.get(key, default=None)**: Возвращает значение по ключу, если ключ есть в словаре, иначе возвращает значение по умолчанию.
* **dict.items()**: Возвращает представление элементов словаря.
* **dict.keys()**: Возвращает представление ключей словаря.
* **dict.pop(key[, default])**: Удаляет элемент по ключу и возвращает его значение. Если ключ не найден, возвращает значение по умолчанию.
* **dict.popitem()**: Удаляет и возвращает пару (ключ, значение) из словаря. Пары возвращаются в произвольном порядке.
* **dict.setdefault(key, default=None)**: Возвращает значение по ключу, если ключ есть в словаре. Если ключа нет, вставляет его с указанным значением.
* **dict.update([other])**: Обновляет словарь, добавляя пары (ключ, значение) из другого словаря или итерируемого объекта.
* **dict.values()**: Возвращает представление значений словаря.

**5. Методы множеств**

* **set.add(elem)**: Добавляет элемент в множество.
* **set.clear()**: Удаляет все элементы из множества.
* **set.copy()**: Возвращает копию множества.
* **set.difference(\*others)**: Возвращает разность множеств.
* **set.difference\_update(\*others)**: Удаляет все элементы множества, присутствующие в других множествах.
* **set.discard(elem)**: Удаляет элемент из множества, если он присутствует.
* **set.intersection(\*others)**: Возвращает пересечение множеств.
* **set.intersection\_update(\*others)**: Оставляет только элементы, присутствующие в обоих множествах.
* **set.isdisjoint(other)**: Возвращает True, если множества не имеют общих элементов.
* **set.issubset(other)**: Возвращает True, если все элементы множества присутствуют в другом множестве.
* **set.issuperset(other)**: Возвращает True, если все элементы другого множества присутствуют в данном множестве.
* **set.pop()**: Удаляет и возвращает элемент из множества.
* **set.remove(elem)**: Удаляет элемент из множества. Вызывает KeyError, если элемент не найден.
* **set.symmetric\_difference(other)**: Возвращает симметрическую разность множеств.
* **set.symmetric\_difference\_update(other)**: Обновляет множество, оставляя только элементы, присутствующие в одном из множеств, но не в обоих.
* **set.union(\*others)**: Возвращает объединение множеств.
* **set.update(\*others)**: Добавляет элементы из других множеств в текущее множество.