

Pydantic 2

Описание

Библиотека валидации (проверка на соответствие типов) и трансформации (автоматическое приведение к нужным типам и форматам) данных.

Модели наследуются от класса `BaseModel`. Модель описывает набор полей, представляющих структуру данных и условия валидации.

Установка

```
pip install pydantic
```

Типизация:

- Простая: указание типа, например, `name: str`.
- Объект `Field()`: дополнительные параметры, например, значения по умолчанию, ограничения и т.д.

Внутри класса можно комбинировать способы типизации.

```
from pydantic import BaseModel, Field

class User(BaseModel):
    name: str
    email: str = Field(..., alias='email_address')
```

Валидация:

- Минимальная валидация: встроенные типы Python (например, `str`, `int`).
- Валидаторы: например `EmailStr` для проверки email-адресов. Требуется установка дополнительных зависимостей: `pydantic[email]` или `pydantic[all]`.
- `@field_validator` — добавляет логику валидации поля. Вызывается при создании или изменении модели.

```
from pydantic import BaseModel, field_validator

class User(BaseModel):
    age: int
```

```
@field_validator('age')
def check_age(cls, value):
    if value < 18:
        raise ValueError('Возраст должен быть больше 18 лет')
    return value
```

- `@computed_field` — вычисляемое поле на основе данных в модели. Его можно использовать для автоматической генерации значений, а также для валидации.

```
from pydantic import BaseModel, computed_field

class User(BaseModel):
    name: str
    surname: str

    @computed_field
    def full_name(self) -> str:
        return f"{self.name} {self.surname}"
```

- `@model_validator` - валидация всей модели.

Работает со всей моделью (а не с отдельными полями), может изменять данные перед валидацией (`mode='before'`) или после (`mode='after'`), полезен для комплексных проверок, где одно поле зависит от другого, может возвращать новую версию модели (если нужно модифицировать данные).

```
@model_validator(mode='before')
def validate_before(cls, data: dict):
    if 'username' not in data:
        data['username'] = "guest_" + str(data.get('id', 0))
    return data
```

```
@model_validator(mode='after')
def validate_after(self):
    if self.age < 18 and self.is_premium:
        raise ValueError("Minors cannot have premium accounts!")
    return self
```

При проверке `before` передается класс, при `after` - объект. Можно делать два валидатора: `before` для подстановки вычисляемых значений и `after` для финальной проверки

Интеграция с SQLAlchemy:

Для настройки используется параметр ConfigDict с флагом from_attributes=True.

```
from datetime import date
from pydantic import BaseModel, ConfigDict

class User(BaseModel):
    id: int
    name: str = 'John Doe'
    birthday_date: date

    config = ConfigDict(from_attributes=True)
```

Для создания модели Pydantic из объекта ORM используется метод from_orm.

```
user = User.from_orm(orm_instance)
```

Ссылки:

[Основа для текста](#)

Revision #4

Created 16 November 2024 18:52:54 by Admin

Updated 7 April 2025 16:23:56 by Admin