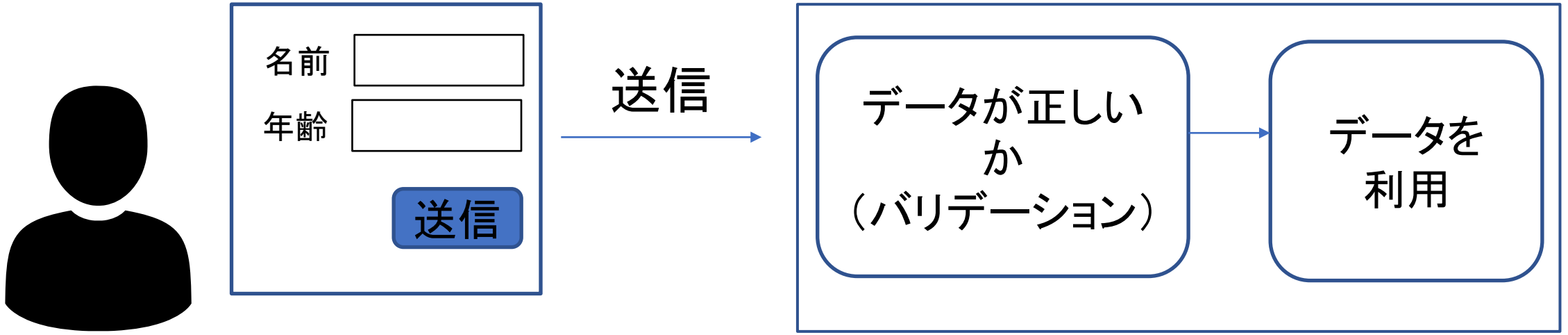
Django基本講座３
(formの利用)

Formとは

Djangoでユーザの入力をサーバに送信して、データが正しいかチェックして処理を行う機能

・HTMLのフォームを簡単に作成できる
・フォームの値をチェックできる
・モデルに合わせた形でフォームを作成して、モデルからDBにデータ挿入ができる

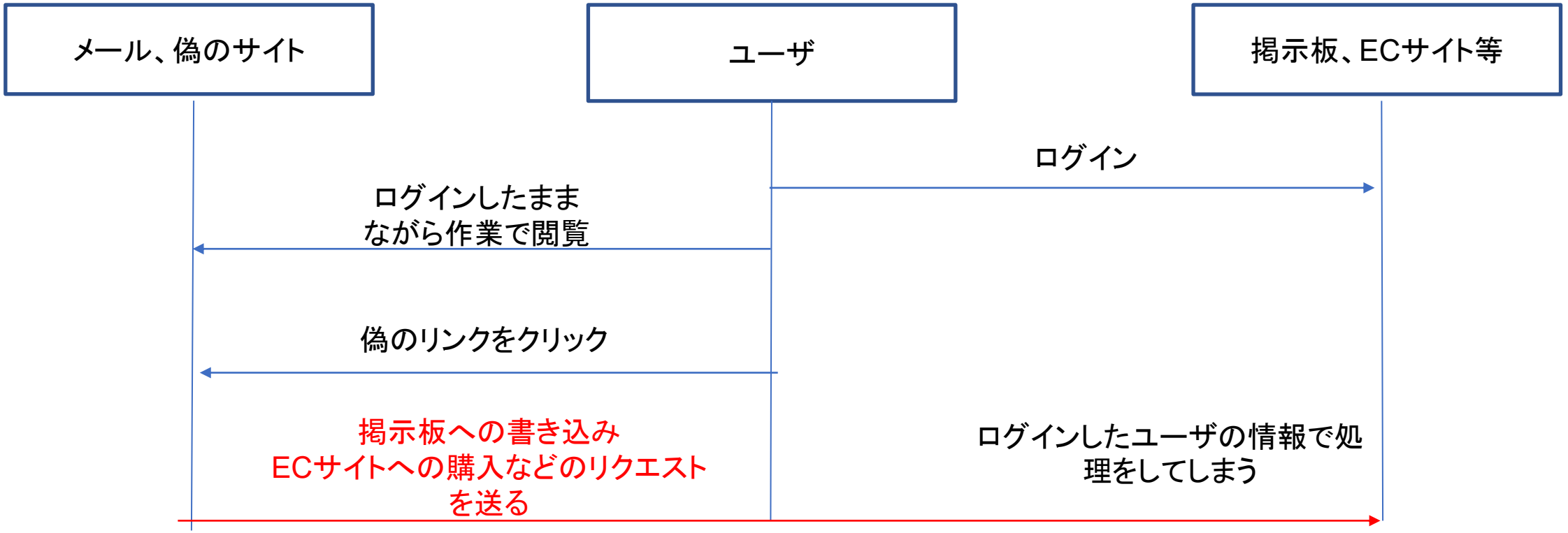


フォームの作成(from django import forms)

```
# フォームクラス側
class UserForm(forms.Form):
    name = forms.CharField(label='Your name', max_length=100) # 文字型でフィールドを作成
    age = forms.IntegerField() ) # 数値型でフィールドを作成
    mail = forms.EmailField() ) # メールアドレス型でフィールドを作成

# ビュー側
form = forms.UserInfo()
return render(request, 'template_path', context={'form': form}) # テンプレートにFormを渡す

# テンプレート側
<form action="/your-name/" method="post">
    {% csrf_token %} # クロスサイトリクエストフォージェリ対策
    {{ form }} # formを表示
    <input type="submit" value="Submit"> # 送信用のボタン
</form>

{{ form.as_p }}: <p>タグでラップして表示
{{ form.as_ul }}: <li>タグでラップして表示
{{ form.as_table }}: <tr>タグでラップして表示
```

攻撃手法(クロスサイトリクエストフォージェリ(CSRF))

メールやSNS、他のサイトを通じて、標的となるページに強制的にリクエストを送らせて、掲示板の書き込み、オンラインショッピングでの注文をさせる



対策・・・重要な処理の場合にはパスワードを要求させる。**csrf_token**を利用する(乱数文字列で対象の掲示板、ECサイトでの遷移しか処理を受け付けないようにする仕組み)。**CAPTCHA**を利用してプログラムでの処理でなく人間の入力であることを保証する

フォームの作成(from django import forms)

```
from django import forms

# テンプレート側
<form action="/your-name/" method="post">
    {% csrf_token %} # クロスサイトリクエストフォージェリ対策
    {{ form }} # formを表示
    <input type="submit" value="Submit"> # 送信用のボタン
</form>

# ビュー側で送られてきたデータを扱う

if request.method == 'POST': # リクエストメソッドがPOSTの場合
        form = forms.UserInfo(request.POST) # リクエストを処理してFormの型に変換
        if form.is_valid(): # formの中のフィールドの値が正しいかチェックする
            form.cleaned_data['subject'] # フォームの中の情報を扱う
```

フォームのフィールド一覧

フィールドには以下のような型があります。
*) 参考: https://docs.djangoproject.com/ja/3.1/ref/forms/fields/

BooleanField	**入力は**チェックボックス。TRUE、Falseの２値を入れられる。
CharField	入力はテキストボックス。文字列型。最大、最小の長さを設定できる。
ChoiceField	入力はセレクトボックス。複数の要素から一つを選ぶ
MultipleChoiceField	複数の項目から複数の要素が選択できる
DateField, DatetimeField	入力は日付、タイムスタンプ。日付型、タイムスタンプ型のカラムが作成される
EmailField	入力はテキストボックス。文字型でメールアドレスとして正しいかチェックする
FileField	ファイルのアップロードの際に用いられる。デフォルトでVARCHAR(100)作成される
DecimalField	数値または、文字列を利用して、正確な数値を格納したい場合に用いられる。
FloatField	数値または、文字列を利用して、浮動小数点数を格納する。
IntegerField	数値または、文字列を利用して、整数を入力する
SlugField	テキストボックスで入力。文字、数値、ハイフン、アンダースコアの文字列を入力
URLField	入力はテキストボックス。URLを入力。
UUIDField	入力はテキストボックス。UUIDの値を入れる。

フォームのカスタマイズ

フォームのラベルを変えたい場合、引数にlabelを追加する
field = forms.CharField(**label=**'名前’)

特定のフォームの入力を必須でなくす場合、required=Falseとする
field = forms.URLField(r**equired=**False)

特定のフォームのウィジェット（入力）を変更する場合、widget=とする
field = forms.CharField(**widget=**forms.Textarea)

初期値を設定する場合、initial=とする
field = forms.Charfleld(initial=‘○○’)

プレースホルダーと追加したい場合、widgetを指定して、placeholder: を要素として追加する
form = forms.EmailField(widget=forms.TextInput(attrs={'placeholder': ‘〇〇@mail.com'}))

文字列の最大、最小の長さを指定したい場合、max_length, min_lengthを指定する
name = forms.CharField(max_length=10)

フォームにID, クラスを追加してレイアウト変更

1. widgetを指定して、要素を追加する

```
form = forms.EmailField(widget=forms.TextInput(attrs={‘class‘: ‘○○’, ‘id’=‘××’}))
```

2. formクラスのコンストラクタ__init__をオーバーライドして、クラスを定義する。

```
def __init__(self, *args, **kwargs):
        super(Form, self).__init__(*args, **kwargs)
        self.fields[‘job’].widget.attrs[‘id’] = ‘id_job‘ # id名を指定
        self.fields[‘hobbies’].widget.attrs[‘class’] = ‘class_hobbies’ # clss名を指定
```

フォームのバリデーション

ユーザがフォームの入力を間違えた場合に、そのチェックを行い、ユーザに誤りを伝える機能

Djangoではこのバリデーション機能をカスタマイズすることができる。

1. フィールドごとのバリデーション用のメソッドを定義する
バリデーションのメソッドは、clean_フィールド名という名前で作成すると自動で実行される
def **clean**_フィールド名(self):
　○○

2. フィールドに対して、validators=[]で属性を追加する
field ~ forms.CharField(validators = [○○]) # ○○には、djangoのライブラリ内のクラス
(https://docs.djangoproject.com/ja/3.1/ref/validators/#built-in-validators)
自作の関数を入れる

3. cleanメソッドを定義して、複数のフィールドの値をチェックする
def clean(self):
　○○

ModelForm

ModelとFormを連携させて利用したい場合に用いる。

 # model側

```
class Post(models.Model):
    pass
```

 # Form側

```
class PotModelForm(forms.ModelForm):

    # fieldのカスタマイズを記載

    class Meta:
        model = Modelの名前
        fields = フィールドを配列指定 or '__all__'で全指定 # 表示するフィールド
        exclude = フィールドを配列指定 or '__all__'で全指定 # 表示しないフィールド
```

 # view側

```
form = forms.PostModelForm(request.POST or None)
if form.is_valid():
    form.save() # データを保存
```

ModelForm(saveメソッドをカスタマイズ)

ModelFormのsaveメソッドを変更することもできます。

それには、ModelFormにdef save(self, *args, **kwargs):を定義（オーバーライド）して中を実装すればよい。

このカスタマイズすることで中に例えば、以下のような処理を行うことができる。

・値の一部の書き替え
・実行される前にログ出力をする
・別のテーブルにもbk用にレコードを入れる
など

ModelForm(formの各要素を表示する)

テンプレート上にFormを表示する場合、
これまで{{form.as_p}}, {{form.as_table}のような形で自動的にformがきれいに表示されるようにしてきましたが、これをカスタマイズして1つ1つの要素を出したい場所に出せるようにしたいと思います。

これをすることで、自分の好きな形で、Formを表示してレイアウトも自由に変更できるようになります。

{{ form.name }} **# フィールドnameを表示する**
{{ form.name.label }} **# フィールドnameのラベルを表示する**
{{ form.name.errors }} **# フィールドnameに対するエラーを表示する**
{{ form.non_field_errors }} **# バリデーションチェックで発生したフィールド単体でないエラーを発生させる**

{{ form.errors }} **# 各フィールドに対するエラーを全て格納する。**

 # 各フィールドのエラーをループさせて、一か所に表示する

```
 {% if form.errors %}
     {% for key, value in form.errors.items %}
     <p>{{key}}: {{value.as_text}}</p>
     {% endfor %}
   {% endif %}
```

FormのHTMLを外だしする

HTMLの要素を別のHTMLファイルに記載して、それを読み込むことで、HTMLの要素を表示することができます。

other.html
○○

main.html
{% include “other.html” %} **# includeを使うとother.htmlの内容を読み込んで表示することができる。**

{% include “other.html” with var1 = ‘○○’ %} **# withを使うと、var1=‘○○’で値を渡すことができる。**

Formset(from django.forms import formset_factory)を利用する

Formsetを利用すると、同じFormを複数個、簡単に画面上に表示することができる。

 # view側(Formsetの作成)
TestFormset = formset_factory(SampleForm, formset=○○, **extra**=〇) # extraでFormを画面上にいくつ表示するか定義
formset = TestFormset()

 # Template側
{{ formset.as_p }} # formsetを表示
or
{{ formset.management_form }} **# フォームの情報を格納しており、formsetをループで利用するのに必要**
{% for form in formset %}
  {{ form }}
{% endfor %}

 # view側(データの取り出し)
if formset.is_valid():
    for form in formset:
        print(form.cleaned_data)

ModelFormset(from django.forms import modelformset_factory)を利用する

ModelFormSetで作成されたFormを用いて、一辺にデータを挿入することができる。

 # view側(Formsetの作成)
TestFormset = modelformset_factory(Model, form=ModelForm, **extra**=〇) **# 対象のモデルとフォームを指定**
or
TestFormset = modelformset_factory(Model, fields='__all__', **extra**=〇) **# 対象のモデルと表示するフィールドを指定**
formset = TestFormset()

 # Template側
{{ formset.as_p }} # formsetを表示
or
{{ formset.management_form }} **# フォームの情報を格納しており、formsetをループで利用するのに必要**
{% for form in formset %}
  {{ form }}
{% endfor %}

 # view側(データの取り出し)
if formset.is_valid():
    formset.save() **# フォームセット保存**

ファイルのアップロード

HTMLからアップロードされたファイルのデータは、**request.FILES**に格納される

Modelには、FileField, ImageFieldを用いて、ファイルの格納先のパスを保存する。
ファイルの格納先はDB内でなくファイルシステム内で、DB内のレコードを削除してもファイルそのものは削除されない。

settings.pyに設定を記載
MEDIA_URL = '/media/' # メディアファイルを公開する際のURL
MEDIA_ROOT = os.path.join(BASE_DIR, 'media') # ファイルの保存先

 # テンプレート
```
 <form method="post" enctype="multipart/form-data"> # ファイルをアップロードする
    {% csrf_token %}
    <input type="file" name="myfile">
    <button type="submit">Upload</button>
  </form>
```

 # View(from django.core.files.storage import FileSystemStorage)
myfile = request.FILES['myfile']
fs = FileSystemStorage() **# ファイル保存用のインスタンス**
filename = fs.save(myfile.name, myfile) **# ファイルを保存する**
uploaded_file_url = fs.url(filename) **# ファイルのURL**

Modelを用いたファイルのアップロード

画像を直接画面先から表示するようにするには、プロジェクトのsettings.pyに以下の記述を追加する。

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns +=  static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

これは、通常DEBUGモード(開発環境)でのみ付け加えるオプション(本番は、apache, nginxと言ったWebサーバにメディアの設定は記載する)

```
# models.py
class Image(models.Model):
    description = models.CharField(max_length=255, blank=True)
    path = models.FileField(upload_to='documents/') # アップロードするファイルの格納先を保存する
(upload_to='documents/%Y/%m/%d/'とすると、documents/年/月/日で保存できる)

# views.py
def model_form_upload(request):
    if request.method == 'POST':
        form = Image Form(request.POST, request.FILES) # 保存するファイルを第2引数に置く
        if form.is_valid():
            form.save()
            return redirect('home')
```

Modelを用いたファイルのアップロード

 # template

```
<form method="POST" enctype="multipart/form-data">
      {% csrf_token %}
      {{ form.as_p }}
       <input type="submit" value="保存">
   </form>
```

 # 画像を表示する

```
<img src="{{ user.picture.url }}">
```

問題

テーブルを作成して、フォームを用いたデータの追加、削除、更新を行う演習をします。

1. 右下のようなstudentsテーブルを作成しましょう。
2. ModelFormを用いてデータを挿入しましょう
3. FormとModelを用いて、すでに挿入されたデータを更新、削除する画面を作成しましょう
4. ModelFormSetを用いて、一度に3件大量のデータを挿入しましょう

students
id(PK) age(int) name(VARCHAR(50)) grade(int) picture(file)