## Explicación Detallada de Todas las Funciones del Crawler

### Función: limpiar_html()

```
def limpiar_html(texto_html: str) -> str:
    if not texto_html:
        return ""
    texto = re.sub(r"<.*?>", " ", texto_html)
    texto = re.sub(r"\s+", " ", texto)
    return texto.strip()
```

Esta función recibe un texto que contiene HTML y devuelve solo el contenido textual.

- Si el texto es vacío o None, devuelve una cadena vacía para evitar errores.
- Usa una expresión regular para reemplazar todas las etiquetas HTML por espacios.
- Luego elimina espacios repetidos y limpia los bordes con strip().

La función se usa para procesar tanto el texto de los enlaces como el contenido HTML completo antes de clasificarlo.

### Función: descargar_html()

```
def descargar_html(url: str) -> str:
    try:
        resp = requests.get(url, timeout=10)
        resp.raise_for_status()
        return resp.text
    except Exception as e:
        print(f"[ERROR] No se pudo descargar {url}: {e}")
        return
```

Descarga el contenido HTML de la URL indicada.

- requests.get(url, timeout=10): realiza la petición HTTP.
- raise_for_status(): lanza una excepción si la página devolvió error.
- Si todo va bien, resp.text contiene el HTML.
- Si ocurre algún error, lo imprime y devuelve una cadena vacía.

Esto permite que el programa principal compruebe si la descarga falló.

### Función: generar_resumen_desde_url()

```
def generar_resumen_desde_url(url: str, fallback_text: str = "") -> str:
    texto = ""
    try:
        downloaded = trafilatura.fetch_url(url)
        if downloaded:
            texto = trafilatura.extract(
                downloaded,
                include_comments=False,
                include_tables=False,
                favor_recall=False
            ) or ""
    except Exception:
        texto = ""
```

```
    texto = (texto or "").strip()

    if not texto:
        if fallback_text:
            texto = fallback_text
        else:
            return "Contenido no accesible automáticamente para esta URL."

    frases = re.split(r'(?<=[.!?])\s+', texto)
    frases_buenas = [f.strip() for f in frases if len(f.strip()) > 40]

    if frases_buenas:
        resumen = " ".join(frases_buenas[:2])
    else:
        resumen = " ".join(frases[:2])

    if len(resumen) > 320:
        resumen = resumen[:317] + "..."

    try:
        resumen_es = translator.translate(resumen, dest="es").text
    except:
        resumen_es = resumen

    return resumen_es
```

Genera un resumen del contenido real de una URL y lo traduce al español.

1. Obtención del texto:
- Usa trafilatura.fetch_url() para descargar contenido.
- Usa trafilatura.extract() para obtener solo el texto útil.

2. Manejo de fallos:
- Si no hay texto, usa fallback_text (texto del enlace) o devuelve un mensaje fijo.

3. Creación del resumen:
- Divide el texto en frases.
- Selecciona las dos primeras frases largas o, si no hay, las dos primeras frases normales.
- Limita el resumen a 320 caracteres.

4. Traducción:
- Traduce usando googletrans → cliente NO oficial de Google Translate.
- Si falla, devuelve el resumen sin traducir.

El resumen se genera localmente; Google Translate se usa solo para traducir.

## Funciones de Base de Datos

```
def inicializar_db():
    conn = sqlite3.connect(DB_PATH)
    cur = conn.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS links (id INTEGER PRIMARY KEY AUTOINCREMENT,
source_url TEXT, link_url TEXT, link_text TEXT, summary TEXT)")
    conn.commit()
    conn.close()
```

```
def guardar_enlace(source_url, link_url, link_text, summary):
    conn = sqlite3.connect(DB_PATH)
    cur = conn.cursor()
    cur.execute("INSERT INTO links (source_url, link_url, link_text, summary) VALUES (?,
?, ?, ?)", (source_url, link_url, link_text, summary))
    conn.commit()
    conn.close()

def obtener_enlaces(limit=200):
    conn = sqlite3.connect(DB_PATH)
    cur = conn.cursor()
    cur.execute("SELECT source_url, link_url, summary FROM links ORDER BY id DESC LIMIT
?", (limit,))
    rows = cur.fetchall()
    conn.close()
    return rows
```

Aquí se gestiona la base de datos SQLite donde se almacenan los enlaces analizados.

1. inicializar_db():
- Crea la tabla si no existe.
- La tabla almacena: URL origen, URL encontrada, texto del enlace y resumen.

2. guardar_enlace():
- Inserta un registro con toda la información procesada de un enlace.

3. obtener_enlaces():
- Recupera hasta 'limit' enlaces para mostrarlos en el informe HTML.

## Función: generar_vista_html()

```
def generar_vista_html(output_file="resultado.html", limit=200):
    filas = obtener_enlaces(limit)
    path = Path(output_file)
    html = "<html>... tabla con enlaces y resúmenes ...</html>"
    Path(output_file).write_text(html, encoding="utf-8")
```

Genera un archivo HTML que sirve como informe visual del crawler.

- Obtiene las filas desde la base de datos.
- Construye una página HTML con estilos CSS y una tabla.
- Inserta cada enlace y su resumen dentro de la tabla.
- Guarda el archivo final en disco.

Este archivo es el resultado visual del proyecto.

## Función: entrenar_modelo_ia()

```
def entrenar_modelo_ia():
    documentos = [...]
    etiquetas = [...]
    vectorizer = TfidfVectorizer(ngram_range=(1,2), max_df=0.85, sublinear_tf=True,
stop_words="english")
    X = vectorizer.fit_transform(documentos)
    modelo = MultinomialNB(alpha=0.5)
```

```
    modelo.fit(X, etiquetas)
    return vectorizer, modelo
```

Entrena un modelo de clasificación de texto muy básico.

- Usa TF-IDF para convertir texto en números.
- Usa Naive Bayes para clasificar.
- Las categorías entrenadas son: noticias, deportes, tienda y tecnología.

El modelo es sencillo pero suficiente para identificar el tipo de página descargada.

## Función: clasificar_contenido_html()

```
def clasificar_contenido_html(html, vectorizer, modelo):
    texto = limpiar_html(html).lower()
    if not texto:
        return "desconocido"
    X_new = vectorizer.transform([texto])
    return modelo.predict(X_new)[0]
```

Clasifica el HTML completo de la página en una categoría.

- Limpia el HTML para convertirlo en texto limpio.
- Lo transforma en vector con el mismo vectorizer del entrenamiento.
- Predice la categoría usando el modelo.

Posibles resultados: noticias, deportes, tienda, tecnología o desconocido.

## Función: main()

```
def main():
    url = input("Introduce la URL inicial: ")
    html = descargar_html(url)
    matches = list(LINK_REGEX.finditer(html))[:50]
    inicializar_db()

    for m in matches:
        link_url = m.group("url")
        link_text = limpiar_html(m.group("text"))
        link_url_abs = urljoin(url, link_url) if link_url.startswith("/") else link_url
        resumen = generar_resumen_desde_url(link_url_abs, fallback_text=link_text)
        guardar_enlace(url, link_url_abs, link_text, resumen)

    generar_vista_html("supernenas.html")
    vectorizer, modelo = entrenar_modelo_ia()
    categoria = clasificar_contenido_html(html, vectorizer, modelo)
    print("Categoría detectada:", categoria)

if __name__ == "__main__":
    main()
```

La función central que dirige todo el proceso.

1. Pide una URL al usuario.
2. Descarga la página.
3. Extrae enlaces con la expresión regular.

4. Inicializa la base de datos.
5. Para cada enlace:
   - Obtiene su URL y texto.
   - Normaliza la URL si es relativa.
   - Genera un resumen.
   - Guarda la información en la base de datos.

6. Genera un HTML con todos los resultados.
7. Entrena el modelo IA y clasifica la página.

Es el flujo completo del programa.