Руководство по написанию сценариев для Zaplet

Содержание

Введение

Сценарии в Zaplet — это последовательность HTTP-запросов, определенная в YAML-файле, которая позволяет автоматизировать и воспроизводить тестирование API. Сценарии могут содержать переменные, условия, ожидаемые ответы и другие элементы для создания гибких и мощных тестов.

Основные преимущества использования сценариев:

- Воспроизводимое тестирование API
- Последовательное выполнение связанных запросов
- Извлечение и использование данных из предыдущих ответов
- Валидация ответов
- Условное выполнение шагов

Файлы сценариев Zaplet имеют расширение `.zpl` и используют синтаксис YAML.

Структура файла сценария

Основные элементы

Файл сценария состоит из следующих основных элементов:

```yaml
name: Имя сценария
description: Описание сценария
repeat: 1
continue_on_error: false

environment:
  key1: value1
  key2: value2

steps:
  - name: Шаг 1
    description: Описание шага 1
    request:
      # Определение запроса
    expected_response:
      # Ожидаемый ответ
    variables:
      # Переменные для извлечения

  - name: Шаг 2
    # ...
```

Обязательные элементы:

- **name**: имя сценария (строка)
- **steps**: список шагов (массив)

Необязательные элементы:

- **description**: описание сценария (строка)
- **repeat**: количество повторений сценария (целое число или `infinite`)
- **continue_on_error**: продолжать выполнение при ошибке (логическое значение)
- **environment**: глобальные переменные (объект)

Формат YAML

Сценарии используют синтаксис YAML, который имеет следующие основные правила:

- Отступы используются для обозначения вложенности (обычно 2 пробела)
- `-` используется для обозначения элементов массива
- `key: value` используется для определения пар ключ-значение
- Строки могут быть заключены в одинарные (`'`) или двойные (`"`) кавычки
- Многострочные строки могут быть определены с помощью `|` или `>`

Пример:

```yaml
# Это комментарий
простая_строка: значение
строка_в_кавычках: "значение с специальными символами"
многострочная_строка: |
  Первая строка
  Вторая строка
  Третья строка
массив:
  - элемент1
  - элемент2
  - элемент3
вложенный_объект:
  ключ1: значение1
  ключ2: значение2
```

Шаги сценария

Структура шага

Каждый шаг в сценарии представляет собой отдельный HTTP-запрос с дополнительными параметрами:

```yaml
- name: Имя шага
  description: Описание шага
  request:
    # Определение запроса
  expected_response:
    # Ожидаемый ответ (опционально)
  variables:
    # Переменные для извлечения (опционально)
  condition: # Условие выполнения (опционально)
  delay: 1000 # Задержка перед выполнением в миллисекундах (опционально)
```

Обязательные элементы шага:

- **name**: имя шага
- **request**: определение HTTP-запроса

Необязательные элементы шага:

- **description**: описание шага
- **expected_response**: ожидаемый ответ для валидации
- **variables**: определение переменных для извлечения из ответа
- **condition**: условие выполнения шага
- **delay**: задержка перед выполнением шага в миллисекундах

Определение запроса

Запрос определяется с помощью следующих параметров:

```
request:
  method: GET  # HTTP-метод (GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS)
  url: https://api.example.com/endpoint  # URL запроса
  headers:  # Заголовки запроса (опционально)
    Content-Type: application/json
    Authorization: Bearer token
  body: '{"key": "value"}'  # Тело запроса (опционально)
  query_params:  # Параметры запроса (опционально)
    param1: value1
    param2: value2
  timeout: 30  # Таймаут запроса в секундах (опционально)
```

Обязательные параметры:

- **url**: URL запроса
- **method**: HTTP-метод (по умолчанию GET)

Необязательные параметры:

- **headers**: заголовки запроса
- **body**: тело запроса
- **query_params**: параметры запроса
- **timeout**: таймаут запроса в секундах

Примеры разных типов запросов:

GET-запрос:

```
request:
  method: GET
  url: https://api.example.com/users
  headers:
    Authorization: Bearer token
```

POST-запрос с JSON-данными:

```
request:
  method: POST
  url: https://api.example.com/users
  headers:
    Content-Type: application/json
    Authorization: Bearer token
  body: '{"name": "John", "email": "john@example.com"}'
```

PUT-запрос:

```
request:
  method: PUT
  url: https://api.example.com/users/1
  headers:
    Content-Type: application/json
    Authorization: Bearer token
  body: '{"name": "John Updated", "email": "john@example.com"}'
```

DELETE-запрос:

```
request:
  method: DELETE
  url: https://api.example.com/users/1
  headers:
    Authorization: Bearer token
```

Ожидаемые ответы

Для проверки ответа на запрос можно определить ожидаемый ответ:

```
expected_response:
  status_code: 200  # Ожидаемый код состояния
  headers:  # Ожидаемые заголовки (опционально)
    Content-Type: application/json
  body: '{"success": true}'  # Ожидаемое тело ответа (опционально)
```

Все элементы ожидаемого ответа являются необязательными, но для проведения валидации необходимо указать хотя бы один из них.

Переменные

Переменные в сценариях Zaplet позволяют:

- Хранить общие значения для использования в нескольких шагах
- Извлекать значения из ответов на запросы
- Создавать динамические запросы, зависящие от предыдущих ответов

Окружение и глобальные переменные

Глобальные переменные сценария определяются в секции `environment`:

```
environment:
  base_url: https://api.example.com
  auth_token: my_secret_token
  api_version: v1
```

Эти переменные доступны во всех шагах сценария.

Использование переменных

Переменные можно использовать в следующих местах:

- URL запроса
- Заголовки запроса
- Тело запроса
- Условия выполнения

Синтаксис для использования переменных: `${имя_переменной}`

Примеры использования переменных:

В URL:

```
request:
  method: GET
  url: ${base_url}/users/${user_id}
```

В заголовках:

```
request:
  method: GET
  url: ${base_url}/users
  headers:
    Authorization: Bearer ${auth_token}
```

В теле запроса:

```
request:
  method: POST
  url: ${base_url}/users
  headers:
    Content-Type: application/json
  body: '{"name": "${user_name}", "email": "${user_email}"}'
```

Извлечение переменных из ответов

Переменные могут быть извлечены из ответов на запросы с использованием секции `variables`:

```
variables:
  user_id: $.id
  access_token: $.data.access_token
  total_count: $.meta.total
  content_type: header.Content-Type
  status: status_code
  response: body
```

Каждая переменная имеет имя и правило извлечения, определяющее, как значение будет извлечено из ответа.

JSONPath

Для извлечения значений из JSON-ответов используется синтаксис JSONPath:

```
variables:
  user_id: $.id               # Значение поля id в корне JSON
  token: $.data.access_token  # Вложенное поле data.access_token
  first_item: $.items[0].name # Имя первого элемента в массиве items
  count: $.meta.total         # Значение поля meta.total
```

Примеры выражений JSONPath:

- `$.field` - поле в корне JSON
- `$.parent.child` - вложенное поле
- `$.items[0]` - первый элемент массива
- `$.items[0].name` - поле первого элемента массива

Извлечение заголовков

Для извлечения значений из заголовков ответа используется префикс `header.`:

```
variables:
  content_type: header.Content-Type
  location: header.Location
  custom_header: header.X-Custom-Header
```

Извлечение по регулярному выражению

Для извлечения значений из текста ответа с использованием регулярных выражений:

```
variables:
  token: regex:token=([a-zA-Z0-9]+)
  number: regex:"id":(\d+)
```

В этом случае из ответа будет извлечено значение первой группы захвата в регулярном выражении.

Условное выполнение

Zaplet позволяет выполнять шаги сценария условно, в зависимости от значений переменных или результатов предыдущих шагов.

Синтаксис условий

Условия определяются с помощью параметра `condition` в шаге:

```
- name: Выполнение запроса при определенном условии
  condition: status_code == 200
  request:
    # ...
```

Операторы сравнения

Поддерживаемые операторы сравнения:

- `==` - равно
- `!=` - не равно
- `>` - больше
- `<` - меньше
- `>=` - больше или равно
- `<=` - меньше или равно

Примеры условий

```
condition: user_id != ""      # Выполнить, если user_id не пустой
condition: status_code == 200 # Выполнить, если код состояния равен 200
condition: count > 0          # Выполнить, если count больше 0
condition: token != null      # Выполнить, если token не равен null
```

Управление выполнением

Повторение сценария

Сценарий может быть выполнен несколько раз с использованием параметра `repeat`:

```
name: Повторяющийся сценарий
repeat: 5  # Повторить сценарий 5 раз
# ...
```

Для бесконечного повторения:

```
name: Бесконечный сценарий
repeat: infinite  # Повторять сценарий бесконечно
# ...
```

Задержки между шагами

Для добавления задержки перед выполнением шага используйте параметр `delay`:

```
- name: Шаг с задержкой
  delay: 1000  # Задержка 1000 миллисекунд (1 секунда)
  request:
    # ...
```

Обработка ошибок

По умолчанию, выполнение сценария прекращается при первой ошибке. Для изменения этого поведения используйте параметр `continue_on_error`:

```
name: Сценарий с продолжением при ошибках
continue_on_error: true
# ...
```

Валидация ответов

Валидация ответов позволяет проверить, соответствует ли ответ ожидаемому, и при необходимости прервать выполнение сценария.

Проверка кода состояния

Для проверки кода состояния HTTP:

```
expected_response:
  status_code: 200
```

Проверка заголовков

Для проверки наличия и значения определенных заголовков:

```
expected_response:
  headers:
    Content-Type: application/json
    Cache-Control: no-cache
```

Проверка тела ответа

Для проверки тела ответа:

```
expected_response:
  body: '{"success": true}'
```

Если тело ответа в формате JSON, выполняется структурное сравнение. Это означает, что порядок полей и форматирование не имеют значения.

Продвинутые техники

Цепочка запросов

Одно из самых мощных свойств сценариев Zaplet — возможность выполнять последовательность связанных запросов, где каждый последующий запрос использует данные, полученные из предыдущих ответов:

```
steps:
  - name: Авторизация
    request:
      method: POST
      url: ${base_url}/auth
      body: '{"username": "user", "password": "pass"}'
    variables:
      token: $.access_token

  - name: Получение данных
    request:
      method: GET
      url: ${base_url}/data
      headers:
        Authorization: Bearer ${token}
```

Динамическое формирование URL

Переменные можно использовать для динамического формирования URL:

```
steps:
  - name: Получение списка ресурсов
    request:
      method: GET
      url: ${base_url}/resources
    variables:
      resource_id: $.data[0].id

  - name: Получение конкретного ресурса
    request:
      method: GET
      url: ${base_url}/resources/${resource_id}
```

Динамические заголовки

Заголовки запросов также могут быть динамическими:

```
steps:
  - name: Авторизация
    request:
      method: POST
      url: ${base_url}/auth
      body: '{"username": "user", "password": "pass"}'
    variables:
      token: $.access_token

  - name: Запрос с токеном
    request:
      method: GET
      url: ${base_url}/protected-resource
      headers:
        Authorization: Bearer ${token}
        X-Request-ID: ${request_id}
```

Преобразование ответов

Переменные могут быть использованы для преобразования и манипуляции данными между запросами:

```
steps:
  - name: Получение данных
    request:
      method: GET
      url: ${base_url}/data
    variables:
      item_id: $.data.id

  - name: Обновление данных
    request:
      method: PUT
      url: ${base_url}/data/${item_id}
      body: '{"id": "${item_id}", "status": "updated"}'
```

Примеры сценариев

Авторизация и получение данных

```yaml
name: Авторизация и получение данных пользователя
description: Авторизация через API и получение профиля пользователя
repeat: 1
continue_on_error: false

environment:
  base_url: https://api.example.com
  username: test_user
  password: test_password

steps:
  - name: Авторизация
    description: Получение токена авторизации
    request:
      method: POST
      url: ${base_url}/auth/login
      headers:
        Content-Type: application/json
      body: '{"username": "${username}", "password": "${password}"}'
    variables:
      token: $.data.access_token
      user_id: $.data.user_id

  - name: Получение профиля пользователя
    description: Запрос на получение данных профиля
    request:
      method: GET
      url: ${base_url}/users/${user_id}
      headers:
        Authorization: Bearer ${token}
    expected_response:
      status_code: 200
    condition: token != ""
```

CRUD операции

```yaml
name: CRUD операции
description: Создание, чтение, обновление и удаление ресурса
repeat: 1
continue_on_error: false

environment:
  base_url: https://api.example.com
  auth_token: my_test_token

steps:
  - name: Создание ресурса
```

```
    description: Создание нового ресурса
    request:
      method: POST
      url: ${base_url}/resources
      headers:
        Authorization: Bearer ${auth_token}
        Content-Type: application/json
      body: '{"name": "Test Resource", "description": "Test description"}'
    variables:
      resource_id: $.id

  - name: Получение ресурса
    description: Проверка созданного ресурса
    request:
      method: GET
      url: ${base_url}/resources/${resource_id}
      headers:
        Authorization: Bearer ${auth_token}
    expected_response:
      status_code: 200

  - name: Обновление ресурса
    description: Изменение данных ресурса
    request:
      method: PUT
      url: ${base_url}/resources/${resource_id}
      headers:
        Authorization: Bearer ${auth_token}
        Content-Type: application/json
      body: '{"name": "Updated Resource", "description": "Updated
description"}'
    expected_response:
      status_code: 200

  - name: Проверка обновления
    description: Проверка обновленных данных
    request:
      method: GET
      url: ${base_url}/resources/${resource_id}
      headers:
        Authorization: Bearer ${auth_token}
    expected_response:
      status_code: 200
    variables:
      resource_name: $.name

  - name: Удаление ресурса
    description: Удаление созданного ресурса
    request:
      method: DELETE
      url: ${base_url}/resources/${resource_id}
      headers:
```

```
        Authorization: Bearer ${auth_token}
    expected_response:
      status_code: 204
    condition: resource_name == "Updated Resource"
```

Пагинация и обработка списков

```
name: Пагинация и обработка списков
description: Пример работы с пагинацией и обработки списков данных
repeat: 1
continue_on_error: false

environment:
  base_url: https://api.example.com
  auth_token: my_test_token

steps:
  - name: Первая страница
    description: Получение первой страницы данных
    request:
      method: GET
      url: ${base_url}/items?page=1&limit=10
      headers:
        Authorization: Bearer ${auth_token}
    variables:
      total_pages: $.meta.total_pages
      first_item_id: $.data[0].id

  - name: Вторая страница
    description: Получение второй страницы данных
    request:
      method: GET
      url: ${base_url}/items?page=2&limit=10
      headers:
        Authorization: Bearer ${auth_token}
    condition: total_pages > 1
    variables:
      second_page_item_id: $.data[0].id

  - name: Получение отдельного элемента
    description: Получение данных конкретного элемента
    request:
      method: GET
      url: ${base_url}/items/${first_item_id}
      headers:
        Authorization: Bearer ${auth_token}
    expected_response:
      status_code: 200
```

Многошаговый бизнес-процесс

```
name: Многошаговый бизнес-процесс
```

```yaml
description: Пример имитации бизнес-процесса через API
repeat: 1

environment:
  base_url: https://api.example.com
  auth_token: my_test_token

steps:
  - name: Создание заказа
    description: Инициализация нового заказа
    request:
      method: POST
      url: ${base_url}/orders
      headers:
        Authorization: Bearer ${auth_token}
        Content-Type: application/json
      body: '{"customer_id": "cust123", "items": [{"product_id": "prod1",
"quantity": 2}]}'
    variables:
      order_id: $.order_id

  - name: Добавление адреса доставки
    description: Указание адреса доставки для заказа
    request:
      method: PUT
      url: ${base_url}/orders/${order_id}/shipping
      headers:
        Authorization: Bearer ${auth_token}
        Content-Type: application/json
      body: '{"address": "123 Main St", "city": "Example City", "postal_code":
"12345"}'
    expected_response:
      status_code: 200

  - name: Выбор способа оплаты
    description: Добавление информации об оплате
    request:
      method: PUT
      url: ${base_url}/orders/${order_id}/payment
      headers:
        Authorization: Bearer ${auth_token}
        Content-Type: application/json
      body: '{"payment_method": "credit_card", "card_last4": "1234"}'
    expected_response:
      status_code: 200

  - name: Подтверждение заказа
    description: Финализация заказа
    request:
      method: POST
      url: ${base_url}/orders/${order_id}/confirm
      headers:
```

```
      Authorization: Bearer ${auth_token}
  expected_response:
    status_code: 200
  variables:
    status: $.status

- name: Проверка статуса заказа
  description: Проверка успешного создания заказа
  request:
    method: GET
    url: ${base_url}/orders/${order_id}
    headers:
      Authorization: Bearer ${auth_token}
  expected_response:
    status_code: 200
    body: '{"order_id": "${order_id}", "status": "confirmed"}'
  condition: status == "confirmed"
```

Рекомендации и лучшие практики

Организация файлов сценариев

1. **Используйте логичную структуру каталогов**:

```
scenarios/
├── auth/
│   ├── login.zpl
│   └── refresh_token.zpl
├── users/
│   ├── create_user.zpl
│   └── update_user.zpl
└── orders/
    ├── create_order.zpl
    └── process_order.zpl
```

2. **Имена файлов должны отражать содержание**:
   - Используйте snake_case для имен файлов
   - Давайте говорящие имена: `create_user.zpl`, а не `scenario1.zpl`

3. **Разделяйте большие сценарии**:
   - Большие сценарии лучше разделять на несколько меньших
   - Используйте общие переменные окружения

Управление переменными

1. **Используйте осмысленные имена переменных**:
   - `user_id` вместо `id`
   - `access_token` вместо `token`

2. **Группируйте связанные переменные в environment**:

```
environment:
  # API конфигурация
  base_url: https://api.example.com
  api_version: v1

  # Данные пользователя
  username: test_user
  password: test_password
```

3. **Определяйте повторно используемые значения как переменные**:
    - Используйте переменные для повторяющихся значений
    - Это упрощает поддержку и обновление сценариев

Отладка сценариев

1. **Начинайте с простых сценариев**:
    - Создайте и отладьте простой сценарий с одним шагом
    - Постепенно добавляйте больше шагов и сложностей
2. **Включите подробное логирование**:
    - Установите уровень логирования `debug` или `trace` для отладки
    - Проверяйте логи для выявления ошибок и проблем
3. **Проверяйте переменные**:
    - Добавьте промежуточные шаги для проверки значений переменных
    - Используйте условия для проверки ожидаемых значений
4. **Изолируйте проблемы**:
    - При возникновении ошибки, изолируйте проблемный шаг
    - Проверьте запрос вручную с помощью команд Zaplet

Это руководство охватывает основные аспекты написания сценариев для Zaplet. По мере использования инструмента вы обнаружите дополнительные возможности и техники, которые можно применить для тестирования вашего API.