Spring HTTP 客戶端

要回答「如何使用 FeignClient、Spring RestClient 或 Spring WebClient」的查詢，我將解釋每個 Spring 工具如何用來發送 HTTP 要求，並專注於它們的設置和基本使用。這些都是 Spring 生態系統的一部分，用於與 RESTful 服務進行互動，但根據應用程序的需求，它們有不同的用途。以下，我將提供每個工具的清晰、逐步指南，並附上簡單的示例。

---

1. 使用 FeignClient

FeignClient 是由 Spring Cloud 提供的宣告式 REST 客戶端。它允許你通過註解定義 HTTP 客戶端作為接口，特別適合於微服務架構，你需要調用其他服務。

使用 FeignClient 的步驟

1. **添加依賴**：在項目中包含 Spring Cloud 依賴。如果你使用 Maven，將 Spring Cloud 開發者的 Feign 添加到你的 `pom.xml` 中：

   ```xml
   <dependency>
       <groupId>org.springframework.cloud</groupId>
       <artifactId>spring-cloud-starter-openfeign</artifactId>
   </dependency>
   ```

   確保你也有一個 Spring Cloud 的依賴管理塊，指定一個兼容的版本。

2. **啟用 Feign 客戶端**：在你的主應用程序類或配置類上使用 `@EnableFeignClients` 註解來啟用 Feign 支持：

   ```java
   import org.springframework.boot.SpringApplication;
   import org.springframework.boot.autoconfigure.SpringBootApplication;
   import org.springframework.cloud.openfeign.EnableFeignClients;

   @SpringBootApplication
   @EnableFeignClients
   public class MyApplication {
       public static void main(String[] args) {
           SpringApplication.run(MyApplication.class, args);
       }
   }
   ```

3. **定義 FeignClient 接口**：創建一個使用 `@FeignClient` 註解的接口，指定服務名稱或 URL，並定義與 REST 端點對應的方法：

```java
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import java.util.List;

@FeignClient(name = "user-service", url = "http://localhost:8080")
public interface UserClient {
    @GetMapping("/users")
    List<User> getUsers();
}
```

這裡，`name` 是客戶端的邏輯名稱，`url` 是目標服務的基礎 URL。`@GetMapping` 註解映射到 `/users` 端點。

4. **注入和使用客戶端**：在你的服務或控制器中自動注入接口並調用其方法，就像它們是本地的：

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class UserService {
    @Autowired
    private UserClient userClient;

    public List<User> fetchUsers() {
        return userClient.getUsers();
    }
}
```

關鍵點

- FeignClient 默認是同步的。
- 它適合於微服務，當你省略 `url` 並讓 Spring Cloud 解析它時，特別適合於服務發現（例如 Eureka）。
- 可以通過回退或斷路器（例如 Hystrix 或 Resilience4j）添加錯誤處理。

---

2. 使用 Spring RestClient

Spring RestClient 是 Spring Framework 6.1 中引入的同步 HTTP 客戶端，作為已被廢棄的 `RestTemplate` 的現代替代品。它提供了一個流暢的 API 來構建和執行請求。

使用 RestClient 的步驟

1. **依賴**：RestClient 包含在 `spring-web` 中，這是 Spring Boot 的 `spring-boot-starter-web` 的一部分。通常不需要額外的依賴：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2. **創建 RestClient 實例**：使用其靜態 `create()` 方法或使用構建器進行自定義來實例化 `RestClient`：

```
import org.springframework.web.client.RestClient;

RestClient restClient = RestClient.create();
```

   為了自定義配置（例如超時），使用 `RestClient.builder()`。

3. **構建和執行請求**：使用流暢的 API 指定 HTTP 方法、URI、標頭和主體，然後檢索響應：

```
import org.springframework.http.MediaType;
import org.springframework.web.client.RestClient;
import java.util.List;

public class UserService {
    private final RestClient restClient;

    public UserService() {
        this.restClient = RestClient.create();
    }

    public List<User> fetchUsers() {
        return restClient.get()
            .uri("http://localhost:8080/users")
            .accept(MediaType.APPLICATION_JSON)
            .retrieve()
            .body(new ParameterizedTypeReference<List<User>>() {});
    }
}
```

   - `get()` 指定 HTTP 方法。
   - `uri()` 設置端點。
   - `accept()` 設置預期的內容類型。

- `retrieve()` 執行請求，`body()` 提取響應，使用 `ParameterizedTypeReference` 來處理泛型類型，例如列表。

4. **處理響應**：由於 RestClient 是同步的，響應會直接返回。為了更多控制（例如狀態碼），使用 `toEntity()`：

```
import org.springframework.http.ResponseEntity;

ResponseEntity<List<User>> response = restClient.get()
    .uri("http://localhost:8080/users")
    .accept(MediaType.APPLICATION_JSON)
    .retrieve()
    .toEntity(new ParameterizedTypeReference<List<User>>() {});
List<User> users = response.getBody();
```

關鍵點

- RestClient 是同步的，適合傳統的、阻塞應用程序。
- 它在 HTTP 錯誤時會拋出異常（例如 `RestClientException`），你可以捕捉並處理它們。
- 它是 `RestTemplate` 的替代品，具有更直觀的 API。

---

3. 使用 Spring WebClient

Spring WebClient 是 Spring WebFlux 中引入的反應式、非阻塞 HTTP 客戶端。它設計用於非同步操作，並與反應式流（Mono 和 Flux）集成。

使用 WebClient 的步驟

1. **添加依賴**：在項目中包含 WebFlux 依賴：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

2. **創建 WebClient 實例**：使用基礎 URL 或默認設置來實例化 `WebClient`：

```
import org.springframework.web.reactive.function.client.WebClient;

WebClient webClient = WebClient.create("http://localhost:8080");
```

使用 `WebClient.builder()` 進行自定義配置（例如編碼器、過濾器）。

3. **構建和執行請求**：使用流暢的 API 構建請求並檢索反應式響應：

```
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;
import java.util.List;

public class UserService {
    private final WebClient webClient;

    public UserService(WebClient webClient) {
        this.webClient = webClient;
    }

    public Mono<List<User>> fetchUsers() {
        return webClient.get()
            .uri("/users")
            .accept(MediaType.APPLICATION_JSON)
            .retrieve()
            .bodyToFlux(User.class)
            .collectList();
    }
}
```

   - `bodyToFlux(User.class)` 處理 `User` 對象的流。
   - `collectList()` 將 `Flux<User>` 轉換為 `Mono<List<User>>`。

4. **訂閱響應**：由於 WebClient 是反應式的，你必須訂閱 `Mono` 或 `Flux` 來觸發請求：

```
Mono<List<User>> usersMono = fetchUsers();
usersMono.subscribe(users -> System.out.println(users));
```

   或者在反應式管道中鏈接它，或者在反應式上下文中阻塞（不推薦）：

```
List<User> users = fetchUsers().block();
```

關鍵點

- WebClient 是非阻塞的，適合於使用 Spring WebFlux 構建的反應式應用程序或處理高並發。
- 使用 `Mono` 來處理單值響應，使用 `Flux` 來處理多值響應。
- 可以使用操作符（例如 `onErrorResume()` 或 `retry()`）來處理錯誤。

比較和何時使用每個

- **FeignClient**：用於微服務設置中的宣告式、接口型客戶端。它是同步的，簡化了服務之間的調用。
- **RestClient**：用於傳統、阻塞 Spring 應用程序中的同步 HTTP 請求。它是 `RestTemplate` 的現代替代品。
- **WebClient**：用於 WebFlux 基於的應用程序或處理高並發的反應式、非阻塞 HTTP 請求。

示例：調用 POST 端點

假設你需要通過調用 `POST /users` 並帶有一個 `User` 對象來創建一個用戶。

FeignClient

```
@FeignClient(name = "user-service", url = "http://localhost:8080")
public interface UserClient {
    @PostMapping("/users")
    User createUser(@RequestBody User user);
}

@Service
public class UserService {
    @Autowired
    private UserClient userClient;

    public User createUser(User user) {
        return userClient.createUser(user);
    }
}
```

RestClient

```
@Service
public class UserService {
    private final RestClient restClient = RestClient.create();

    public User createUser(User user) {
        return restClient.post()
            .uri("http://localhost:8080/users")
            .contentType(MediaType.APPLICATION_JSON)
```

```
            .body(user)
            .retrieve()
            .body(User.class);
    }
}
```

WebClient

```
@Service
public class UserService {
    private final WebClient webClient;

    public UserService(WebClient webClient) {
        this.webClient = WebClient.create("http://localhost:8080");
    }

    public Mono<User> createUser(User user) {
        return webClient.post()
            .uri("/users")
            .contentType(MediaType.APPLICATION_JSON)
            .bodyValue(user)
            .retrieve()
            .bodyToMono(User.class);
    }
}
```

結論

- **FeignClient**：定義一個接口，啟用它，然後像本地豆一樣使用它。
- **RestClient**：流暢地構建請求並同步獲取響應。
- **WebClient**：流暢地構建請求，檢索反應式流，並非同步處理它們。

根據你的應用程序架構選擇：FeignClient 用於微服務，RestClient 用於阻塞應用程序，WebClient 用於反應式應用程序。每個都遵循上述簡單的設置和使用模式。