Calling External APIs in Spring Boot and Handling Issues

Spring Boot Approach to External API Calls

In Spring Boot, you have several robust options for calling external APIs:

1. Using RestTemplate (Synchronous)

java

```java
@Service
public class ApiService {

    private final RestTemplate restTemplate;

    public ApiService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public ResponseEntity<String> callExternalApi(String url) {
        return restTemplate.getForEntity(url, String.class);
    }
}
```

2. Using WebClient (Asynchronous - Recommended for new projects)

java

```java
@Service
public class ApiService {

    private final WebClient webClient;

    public ApiService(WebClient.Builder webClientBuilder) {
        this.webClient = webClientBuilder.baseUrl("https://api.example.com"
).build();
    }

    public Mono<String> callExternalApi(String endpoint) {
        return webClient.get()
                .uri(endpoint)
                .retrieve()
                .bodyToMono(String.class);
    }
}
```

Handling Common Issues in Spring Boot

1. Retry Mechanism

java

```java
@Configuration
public class AppConfig {

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder
                .setConnectTimeout(Duration.ofSeconds(5))
                .setReadTimeout(Duration.ofSeconds(5))
                .build();
    }

    @Bean
    public RetryTemplate retryTemplate() {
        RetryTemplate retryTemplate = new RetryTemplate();

        FixedBackOffPolicy backOffPolicy = new FixedBackOffPolicy();
        backOffPolicy.setBackOffPeriod(1000); // 1 second delay
        retryTemplate.setBackOffPolicy(backOffPolicy);

        SimpleRetryPolicy retryPolicy = new SimpleRetryPolicy();
        retryPolicy.setMaxAttempts(3);
        retryTemplate.setRetryPolicy(retryPolicy);

        return retryTemplate;
    }
}

@Service
public class ApiService {

    private final RestTemplate restTemplate;
    private final RetryTemplate retryTemplate;
    public ApiService(RestTemplate restTemplate, RetryTemplate retryTemplat
e) {
        this.restTemplate = restTemplate;
        this.retryTemplate = retryTemplate;
    }

    public String callApiWithRetry(String url) {
        return retryTemplate.execute(context -> {
            return restTemplate.getForObject(url, String.class);
        });
    }
}
```

2. Circuit Breaker with Resilience4j

java

```
@Configuration
public class ResilienceConfig {

    @Bean
    public CircuitBreakerConfig circuitBreakerConfig() {
        return CircuitBreakerConfig.custom()
                .failureRateThreshold(50)
                .waitDurationInOpenState(Duration.ofMillis(1000))
                .slidingWindowSize(2)
                .build();
    }

    @Bean
    public CircuitBreakerRegistry circuitBreakerRegistry() {
        return CircuitBreakerRegistry.of(circuitBreakerConfig());
    }

    @Bean
    public CircuitBreaker circuitBreaker(CircuitBreakerRegistry registry) {
        return registry.circuitBreaker("externalApi");
    }
}

@Service
public class ApiService {

    private final WebClient webClient;
    private final CircuitBreaker circuitBreaker;
    public ApiService(WebClient.Builder webClientBuilder, CircuitBreaker ci
rcuitBreaker) {
        this.webClient = webClientBuilder.baseUrl("https://api.example.com"
).build();
        this.circuitBreaker = circuitBreaker;
    }

    public String callApiWithCircuitBreaker(String endpoint) {
        return circuitBreaker.executeSupplier(() ->
            webClient.get()
                .uri(endpoint)
                .retrieve()
                .bodyToMono(String.class)
                .block()
        );
    }
}
```

3. Proper Error Handling

java

```
@ControllerAdvice
public class ApiExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(WebClientResponseException.class)
    public ResponseEntity<ErrorResponse> handleWebClientResponseException(W
ebClientResponseException ex) {
        ErrorResponse error = new ErrorResponse(
            "API_ERROR",
            "Error calling external API: " + ex.getMessage()
        );
        return new ResponseEntity<>(error, ex.getStatusCode());
    }

    @ExceptionHandler(ResourceAccessException.class)
    public ResponseEntity<ErrorResponse> handleResourceAccessException(Reso
urceAccessException ex) {
        ErrorResponse error = new ErrorResponse(
            "API_CONNECTION_ERROR",
            "Could not connect to external API: " + ex.getMessage()
        );
        return new ResponseEntity<>(error, HttpStatus.SERVICE_UNAVAILABLE);
    }
}
```

```
public class ErrorResponse {
    private String code;
    private String message;
    // constructors, getters, setters
}
```

Best Practices for Spring Boot API Calls

1. **Configuration Management**:

   java

   ```
   @ConfigurationProperties(prefix = "external.api")
   public class ApiConfig {
       private String baseUrl;
       private int timeout;
       // getters and setters
   }
   ```

2. **Request/Response Logging**:
   java

   ```
   @Bean
   public WebClient webClient(WebClient.Builder builder) {
       return builder
           .filter(ExchangeFilterFunction.ofRequestProcessor(clientReque
   st -> {
               log.info("Request: {} {}", clientRequest.method(), client
   Request.url());
               return Mono.just(clientRequest);
           }))
           .filter(ExchangeFilterFunction.ofResponseProcessor(clientResp
   onse -> {
               log.info("Response status: {}", clientResponse.statusCode
   ());
               return Mono.just(clientResponse);
           }))
           .build();
   }
   ```

3. **Rate Limiting**:
   java

   ```
   @Bean
   public RateLimiter rateLimiter() {
   ```

```
    return RateLimiter.of("apiRateLimiter", RateLimiterConfig.custom(
)
        .limitForPeriod(100)
        .limitRefreshPeriod(Duration.ofMinutes(1))
        .timeoutDuration(Duration.ofSeconds(5))
        .build());
}
```

4. **Timeouts Configuration**:

   yaml

```yaml
# application.yml
external:
  api:
    connect-timeout: 5000
    read-timeout: 10000
```