Guia Completo: Códigos de Erro HTTP no Spring Boot

Introdução

Este guia apresenta as melhores práticas para escolher e implementar códigos de status HTTP adequados em aplicações Spring Boot, incluindo exemplos práticos e cenários comuns.

---

1. Códigos de Status Mais Utilizados

2xx - Sucesso

200 OK

- **Quando usar**: Requisições GET bem-sucedidas, atualizações que retornam dados
- **Spring**: Retorno padrão de `@GetMapping`, `@PutMapping` com retorno

```java
@GetMapping("/usuarios/{id}")
public ResponseEntity<Usuario> buscarUsuario(@PathVariable Long id) {
    Usuario usuario = usuarioService.buscarPorId(id);
    return ResponseEntity.ok(usuario);
}
```

201 CREATED

- **Quando usar**: Recurso criado com sucesso
- **Spring**: `ResponseEntity.created()` ou `@ResponseStatus(CREATED)`

```java
@PostMapping("/usuarios")
public ResponseEntity<Usuario> criarUsuario(@RequestBody Usuario usuario) {
    Usuario usuarioCriado = usuarioService.salvar(usuario);
    URI location = ServletUriComponentsBuilder
        .fromCurrentRequest()
        .path("/{id}")
        .buildAndExpand(usuarioCriado.getId())
        .toUri();
    return ResponseEntity.created(location).body(usuarioCriado);
}
```

204 NO CONTENT

- **Quando usar**: Operação bem-sucedida sem retorno de dados (DELETE, PUT sem retorno)

- **Spring**: `ResponseEntity.noContent()`

```java
@DeleteMapping("/usuarios/{id}")
public ResponseEntity<Void> deletarUsuario(@PathVariable Long id) {
    usuarioService.deletar(id);
    return ResponseEntity.noContent().build();
}
```

4xx - Erros do Cliente

400 BAD REQUEST

- **Quando usar**: Dados inválidos na requisição, validação falhou
- **Spring**: `@ResponseStatus(BAD_REQUEST)` ou validação com `@Valid`

```java
@PostMapping("/usuarios")
public ResponseEntity<Usuario> criarUsuario(@Valid @RequestBody Usuario usuario) {
    // Se @Valid falhar, Spring retorna 400 automaticamente
    return ResponseEntity.ok(usuarioService.salvar(usuario));
}

// Tratamento personalizado
@ExceptionHandler(MethodArgumentNotValidException.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
public Map<String, String> handleValidationExceptions(
    MethodArgumentNotValidException ex) {
    Map<String, String> errors = new HashMap<>();
    ex.getBindingResult().getAllErrors().forEach((error) -> {
        String fieldName = ((FieldError) error).getField();
        String errorMessage = error.getDefaultMessage();
        errors.put(fieldName, errorMessage);
    });
    return errors;
}
```

401 UNAUTHORIZED

- **Quando usar**: Usuário não autenticado
- **Spring**: Spring Security ou implementação manual

```java
```

```java
@ExceptionHandler(AuthenticationException.class)
@ResponseStatus(HttpStatus.UNAUTHORIZED)
public ErrorResponse handleAuthenticationException(AuthenticationException ex) {
    return new ErrorResponse("Credenciais inválidas", 401);
}
```

403 FORBIDDEN

- **Quando usar**: Usuário autenticado mas sem permissão
- **Spring**: Spring Security com autorização

```java
@PreAuthorize("hasRole('ADMIN')")
@DeleteMapping("/usuarios/{id}")
public ResponseEntity<Void> deletarUsuario(@PathVariable Long id) {
    usuarioService.deletar(id);
    return ResponseEntity.noContent().build();
}
```

404 NOT FOUND

- **Quando usar**: Recurso não encontrado
- **Spring**: Lançar exceção personalizada

```java
@GetMapping("/usuarios/{id}")
public ResponseEntity<Usuario> buscarUsuario(@PathVariable Long id) {
    Usuario usuario = usuarioService.buscarPorId(id)
        .orElseThrow(() -> new UsuarioNotFoundException("Usuário não encontrado"));
    return ResponseEntity.ok(usuario);
}

@ExceptionHandler(UsuarioNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
public ErrorResponse handleUsuarioNotFound(UsuarioNotFoundException ex) {
    return new ErrorResponse(ex.getMessage(), 404);
}
```

409 CONFLICT

- **Quando usar**: Conflito de estado (email duplicado, recurso já existe)
- **Spring**: Exceção personalizada para conflitos

```java
@ExceptionHandler(DataIntegrityViolationException.class)
@ResponseStatus(HttpStatus.CONFLICT)
public ErrorResponse handleDataIntegrityViolation(DataIntegrityViolationException ex) {
    return new ErrorResponse("Recurso já existe ou viola restrição", 409);
}
```

422 UNPROCESSABLE ENTITY

- **Quando usar**: Dados bem formados mas semanticamente incorretos
- **Spring**: Validações de regra de negócio

```java
@ExceptionHandler(BusinessException.class)
@ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
public ErrorResponse handleBusinessException(BusinessException ex) {
    return new ErrorResponse(ex.getMessage(), 422);
}
```

5xx - Erros do Servidor

500 INTERNAL SERVER ERROR

- **Quando usar**: Erros não tratados, falhas inesperadas
- **Spring**: Tratamento global de exceções

```java
@ExceptionHandler(Exception.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
public ErrorResponse handleGenericException(Exception ex) {
    log.error("Erro interno do servidor", ex);
    return new ErrorResponse("Erro interno do servidor", 500);
}
```

---

2. Implementações Práticas

Tratamento Global de Exceções

```java
```

```java
@RestControllerAdvice
public class GlobalExceptionHandler {

    private static final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);

    @ExceptionHandler(NotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ErrorResponse handleNotFound(NotFoundException ex) {
        log.warn("Recurso não encontrado: {}", ex.getMessage());
        return new ErrorResponse(ex.getMessage(), 404);
    }

    @ExceptionHandler(ValidationException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ErrorResponse handleValidation(ValidationException ex) {
        log.warn("Erro de validação: {}", ex.getMessage());
        return new ErrorResponse(ex.getMessage(), 400);
    }

    @ExceptionHandler(BusinessException.class)
    @ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
    public ErrorResponse handleBusiness(BusinessException ex) {
        log.warn("Erro de negócio: {}", ex.getMessage());
        return new ErrorResponse(ex.getMessage(), 422);
    }

    @ExceptionHandler(AccessDeniedException.class)
    @ResponseStatus(HttpStatus.FORBIDDEN)
    public ErrorResponse handleAccessDenied(AccessDeniedException ex) {
        log.warn("Acesso negado: {}", ex.getMessage());
        return new ErrorResponse("Acesso negado", 403);
    }

    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public ErrorResponse handleGeneric(Exception ex) {
        log.error("Erro interno", ex);
        return new ErrorResponse("Erro interno do servidor", 500);
    }
}
```

Classe de Response de Erro

```java
```

```java
public class ErrorResponse {
    private String message;
    private int status;
    private LocalDateTime timestamp;
    private String path;

    public ErrorResponse(String message, int status) {
        this.message = message;
        this.status = status;
        this.timestamp = LocalDateTime.now();
    }

    // Getters e setters
}
```

Exceções Personalizadas

java

```java
@ResponseStatus(HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException {
    public NotFoundException(String message) {
        super(message);
    }
}

@ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
public class BusinessException extends RuntimeException {
    public BusinessException(String message) {
        super(message);
    }
}

@ResponseStatus(HttpStatus.BAD_REQUEST)
public class ValidationException extends RuntimeException {
    public ValidationException(String message) {
        super(message);
    }
}
```

---

3. Validação com Bean Validation

Configuração de Validação

java

```java
@Entity
public class Usuario {

    @NotNull(message = "Nome é obrigatório")
    @Size(min = 2, max = 100, message = "Nome deve ter entre 2 e 100 caracteres")
    private String nome;

    @Email(message = "Email deve ser válido")
    @NotNull(message = "Email é obrigatório")
    private String email;

    @Past(message = "Data de nascimento deve ser no passado")
    private LocalDate dataNascimento;

    // Getters e setters
}
```

Controller com Validação

java

```java
@RestController
@RequestMapping("/api/usuarios")
@Validated
public class UsuarioController {

    @PostMapping
    public ResponseEntity<Usuario> criar(@Valid @RequestBody Usuario usuario) {
        Usuario usuarioCriado = usuarioService.salvar(usuario);
        return ResponseEntity.status(HttpStatus.CREATED).body(usuarioCriado);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Usuario> atualizar(
            @PathVariable Long id,
            @Valid @RequestBody Usuario usuario) {
        Usuario usuarioAtualizado = usuarioService.atualizar(id, usuario);
        return ResponseEntity.ok(usuarioAtualizado);
    }
}
```

---

4. Cenários Específicos e Códigos Recomendados

API REST CRUD

Operação	Método	Sucesso	Erro Comum	Código Erro
Criar	POST	201 CREATED	Dados inválidos	400 BAD REQUEST
Listar	GET	200 OK	-	-
Buscar	GET	200 OK	Não encontrado	404 NOT FOUND
Atualizar	PUT	200 OK	Não encontrado	404 NOT FOUND
Deletar	DELETE	204 NO CONTENT	Não encontrado	404 NOT FOUND

Autenticação e Autorização

java

```java
// Login
@PostMapping("/auth/login")
public ResponseEntity<TokenResponse> login(@RequestBody LoginRequest request) {
    try {
        TokenResponse token = authService.authenticate(request);
        return ResponseEntity.ok(token); // 200 OK
    } catch (BadCredentialsException ex) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build(); // 401
    }
}

// Recurso protegido
@GetMapping("/admin/relatorios")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<List<Relatorio>> listarRelatorios() {
    // Se não autenticado: 401 UNAUTHORIZED
    // Se não autorizado: 403 FORBIDDEN
    return ResponseEntity.ok(relatorioService.listar());
}
```

Upload de Arquivos

java

```java
@PostMapping("/upload")
public ResponseEntity<String> upload(@RequestParam("file") MultipartFile file) {
    if (file.isEmpty()) {
        return ResponseEntity.badRequest().body("Arquivo vazio"); // 400
    }

    if (file.getSize() > MAX_FILE_SIZE) {
        return ResponseEntity.status(HttpStatus.PAYLOAD_TOO_LARGE)
                .body("Arquivo muito grande"); // 413
    }

    try {
        uploadService.salvar(file);
        return ResponseEntity.status(HttpStatus.CREATED)
                .body("Arquivo enviado com sucesso"); // 201
    } catch (IOException ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Erro ao salvar arquivo"); // 500
    }
}
```

---

5. Configurações Adicionais

application.yml

yaml

```yaml
server:
  error:
    include-message: always
    include-binding-errors: always
    include-stacktrace: on_param
    include-exception: false

spring:
  mvc:
    throw-exception-if-no-handler-found: true
  web:
    resources:
      add-mappings: false
```

Interceptor para Logs

java

```java
@Component
public class LoggingInterceptor implements HandlerInterceptor {

    private static final Logger log = LoggerFactory.getLogger(LoggingInterceptor.class);

    @Override
    public void afterCompletion(HttpServletRequest request,
                                HttpServletResponse response,
                                Object handler, Exception ex) {
        int status = response.getStatus();
        String method = request.getMethod();
        String uri = request.getRequestURI();

        if (status >= 400) {
            log.warn("HTTP {} {} - Status: {}", method, uri, status);
        } else {
            log.info("HTTP {} {} - Status: {}", method, uri, status);
        }
    }
}
```

---

6. Checklist de Boas Práticas

✅ Códigos de Status

- [ ] Use 201 para criação de recursos
- [ ] Use 204 para operações sem retorno
- [ ] Use 400 para dados inválidos
- [ ] Use 401 para não autenticado
- [ ] Use 403 para não autorizado
- [ ] Use 404 para recurso não encontrado
- [ ] Use 409 para conflitos de estado
- [ ] Use 422 para erros de regra de negócio

✅ Tratamento de Exceções

- [ ] Implemente @RestControllerAdvice global
- [ ] Crie exceções personalizadas
- [ ] Use @ResponseStatus nas exceções
- [ ] Padronize formato de erro
- [ ] Configure logs adequadamente

✅ Validação

- [ ] Use Bean Validation (@Valid)
- [ ] Customize mensagens de erro
- [ ] Valide entrada e regras de negócio
- [ ] Retorne erros específicos

✅ Documentação

- [ ] Documente códigos de status na API
- [ ] Inclua exemplos de erro
- [ ] Especifique formatos de resposta

Conclusão

A escolha adequada de códigos HTTP no Spring Boot melhora significativamente a experiência do desenvolvedor que consome sua API. Use este guia como referência para implementar códigos de status consistentes e semânticamente corretos em suas aplicações.

Lembre-se: A consistência é fundamental. Defina padrões para sua equipe e siga-os em toda a aplicação.
