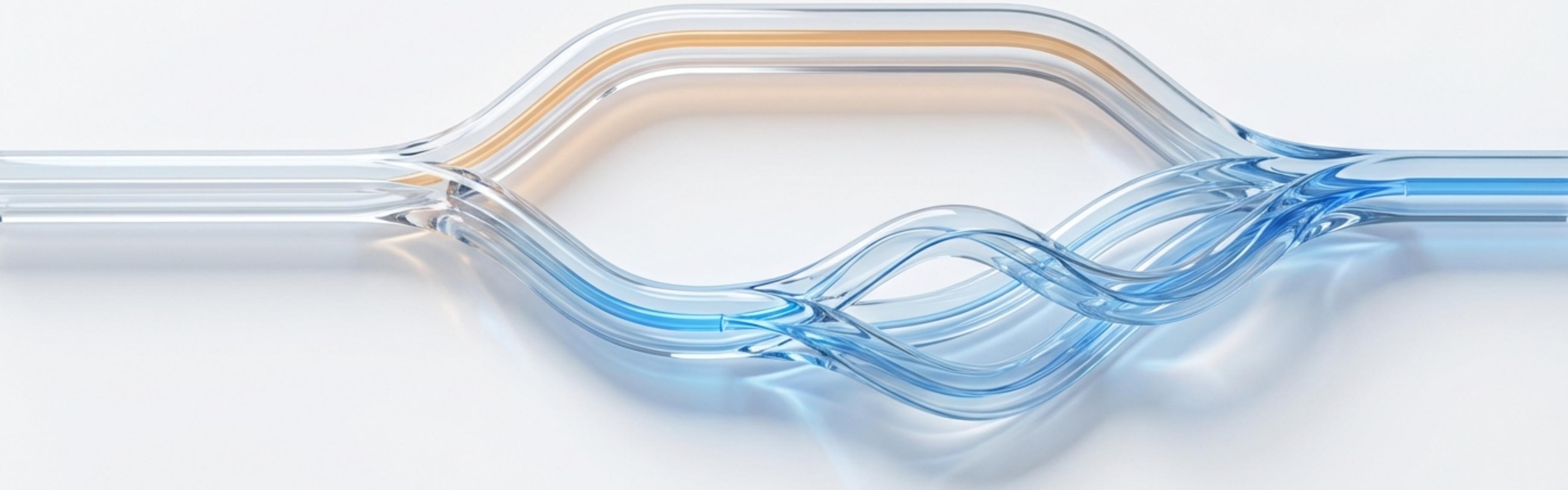


Dominando o Tratamento de Erros em CAP Java

De Exceções a Mensagens Ricas: Um Guia para Desenvolvedores



Sua Decisão Fundamental: Parar ou Coletar?

O CAP Java SDK oferece duas estratégias principais para indicar erros. A escolha correta depende do seu objetivo: interromper o processo imediatamente ou coletar informações e prosseguir.



Lançando Exceções

Aborta completamente o processamento do evento e reverte a transação. Ideal para erros que invalidam toda a operação.



Usando a Messages API

Adiciona mensagens (erro, aviso, sucesso) à requisição atual sem afetar o fluxo de processamento ou a transação. Perfeito para coletar múltiplos erros de validação.

Caminho A: Exceções — A Interrupção Imediata

Qualquer exceção lançada por um event handler **aborda o processamento** e aciona o **rollback da transação**. A classe `ServiceException` é a ferramenta recomendada.

- Permite especificar um `ErrorStatus` (mapeado para um código de status HTTP).
 - O padrão, se não especificado, é 500 Internal Server Error.
 - O OData Adapter converte automaticamente exceções em uma resposta de erro OData para o cliente.

```
// Especificando um status de erro claro
import com.sap.cds.services.ErrorStatuses;
import com.sap.cds.services.
ServiceException;

// ...

throw new ServiceException(ErrorStatuses.
CONFLICT, "Not enough stock available");

// Envolvendo a exceção original para
rastreabilidade
throw new ServiceException(ErrorStatuses
.BAD_REQUEST, "No book title specified",
originalException);
```

Caminho B: Messages API

— O Coletor Flexível

Permite que event handlers adicionem mensagens de erro, aviso, informação ou sucesso à requisição. Mensagens de aviso, info ou sucesso não afetam o processamento.

- Acesse via `context.getMessages()` ou injeção de dependência (`@Autowired Messages messages`).
- O OData V4 Adapter coleta essas mensagens e as insere no header HTTP `sap-messages`.
- O SAP Fiori utiliza essas mensagens para exibir informações detalhadas na UI, com estilo baseado na severidade.

```
// Acessando via EventContext
context.getMessages().success("The order was
successfully placed");
context.getMessages().warn("No book title
specified");

// Adicionando detalhes com a builder API
context.getMessages()
    .error("The book is no longer available")
    .code("BNA")
    .longTextUrl("/help/books/BNA");
    .longTextUrl("/help/books/BNA");
```

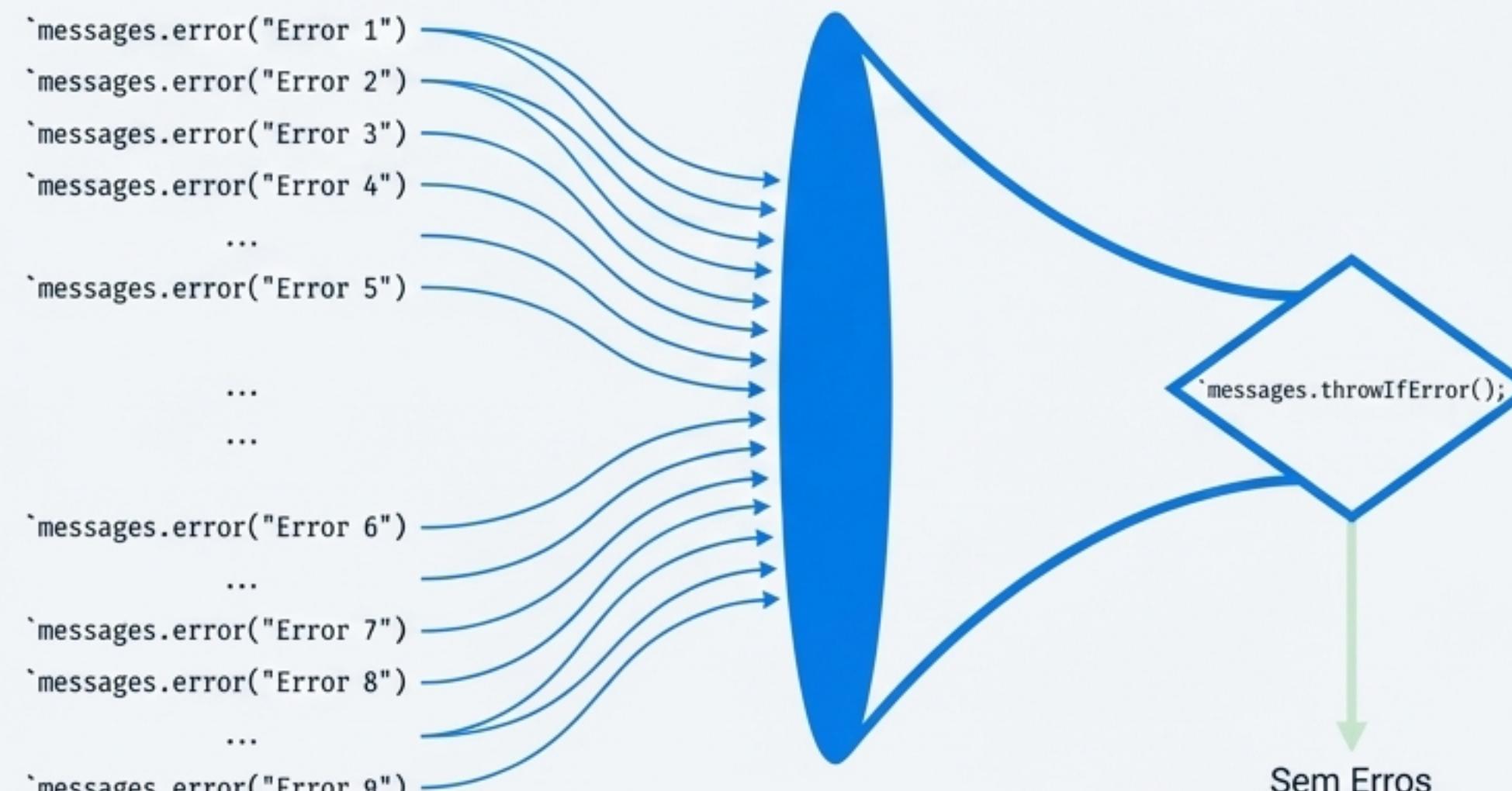
Quando Usar Cada Abordagem?

Característica	Lançando Exceções (`ServiceException`)	Usando a Messages API
Cenário Ideal	Falhas críticas que impedem a continuação (ex: estoque indisponível, falha em sistema externo).	Coletar múltiplos erros de validação em um formulário. Fornecer avisos ou informações não bloqueantes.
Efeito no Fluxo	IMEDIATO : Interrompe o processamento.	NENHUM : O processamento continua.
Efeito na Transação	Rollback da transação ativa.	Nenhum (por padrão).
Comunicação ao Cliente	Resposta de erro OData.	Header HTTP `sap-messages`.
UI (Fiori)	Mostra um erro bloqueante.	Mostra 'toast messages' ou pop-ups informativos.

Unindo os Dois Mundos: Coletando Erros para Lançar Lançar uma Única Exceção

Conceito

É possível usar a Messages API para coletar múltiplos erros e, ao final do processo de validação, lançar uma `ServiceException` se algum erro foi encontrado.



Como Funciona

1. Seus métodos de validação adicionam erros usando `messages.error(...)`.
2. Ao final, você chama `messages.throwIfError()`.
3. Se houver mensagens de erro, o método cria uma `ServiceException` e aborta a requisição, incluindo as demais mensagens na resposta de erro OData.

Ponto Chave

O CAP Java chama `messages.throwIfError()` automaticamente ao final da fase `'Before'` dos handlers. Para validações, essa é a abordagem recomendada.

Elevando o Nível: Da Mensagem Estática à Comunicação Inteligente

Formatação e Localização

Formatação de Mensagens

Tanto `ServiceException` quanto a `Messages` API suportam o estilo de formatação do SLF4J (`{}`).

```
messages.warn("Can't order {} books: Not enough  
on stock", orderQuantity);  
  
throw new ServiceException(  
ErrorStatuses.BAD_REQUEST, "Invalid number: {}",  
invalidValue);
```



Localização de Mensagens

Coloque seus textos em arquivos de propriedades (ex: `messages.properties`). Passe a chave da mensagem para a API em vez do texto. O CAP integra-se com o suporte do Spring para resource bundles
resource bundles (`src/main/resources/messages.properties`).

```
my.message.key = This is a localized message with  
{0} parameters
```

```
messages.warn("my.message.key", paramNumber);
```

O Framework a Seu Favor: Traduções Nativas e Como Personalizá-las



Traduções "Out-of-the-Box"

CAP Java já fornece traduções para mensagens de erro comuns de anotações de validação (`@mandatory`, `@assert...`) e segurança (`@requires`, `@restrict`). Os nomes de entidades e elementos são usados para tornar as mensagens mais claras.

Para desabilitar: `cds.errors.defaultTranslations.enabled: false`



Sobrescrevendo Mensagens Padrão

Se uma tradução padrão não for adequada, você pode sobrescrevê-la. Forneça a nova mensagem usando o código de erro correspondente no seu arquivo `'messages.properties'`.

```
# src/main/resources/messages.properties
409003 = Please enter a value
```



Pro Tip: Explore a enumeração `'com.sap.cds.services.utils.CdsErrorStatuses'` na sua IDE para descobrir todos os códigos de erro disponíveis.

Precisão Cirúrgica: Apontando Erros para a UI com `target`

O Problema

Mensagens de erro genéricas podem confundir o usuário.
Qual campo está incorreto?

A Solução

A propriedade `target` na mensagem informa à UI (especialmente SAP Fiori) a qual elemento da entidade a mensagem se refere, permitindo exibir o erro junto ao campo correspondente.



Caveat Importante

- O SAP Fiori interpreta corretamente o `target` em respostas de erro OData V4.
- No header `sap-messages`, o `target` é majoritariamente ignorado.
- **Conclusão:** Para um targeting eficaz na UI, use a propriedade `target` ao lançar uma `ServiceException`.

Before



After



`target` em Ação: Eventos CRUD

Em um handler `@Before` para a criação de uma entidade `Books`, queremos validar o campo `title`.

Em eventos CRUD, o `target` é relativo ao parâmetro `cqn`, que representa a entidade sendo processada.

API Genérica (Lambda)

```
// Referência implícita ao 'cqn'  
throw new ServiceException(ErrorStatuses.  
BAD_REQUEST, "No title specified")  
.messageTarget(b -> b.get("title"));
```

API Tipada (Type-Safe)

```
// Mais seguro e com autocomplete da IDE  
throw new ServiceException(ErrorStatuses.  
BAD_REQUEST, "No title specified")  
.messageTarget(Books_.class, b -> b.title());
```

Navegando em Associações

```
// Validando o nome do autor associado  
throw new ServiceException(ErrorStatuses.BAD_REQUEST, "No author name specified")  
.messageTarget(Books_.class, b -> b.author().name());
```

`target` em Ação: Ações e Funções 'Bound'

Para uma ação `addReview` vinculada à entidade `Books`, o `target` pode se referir a:

1. Um campo da entidade `Books` vinculada (via `cqn`).
2. Um dos parâmetros de entrada da própria ação (ex: `reviewer`, `rating`).

</> Referenciando um Parâmetro da Ação

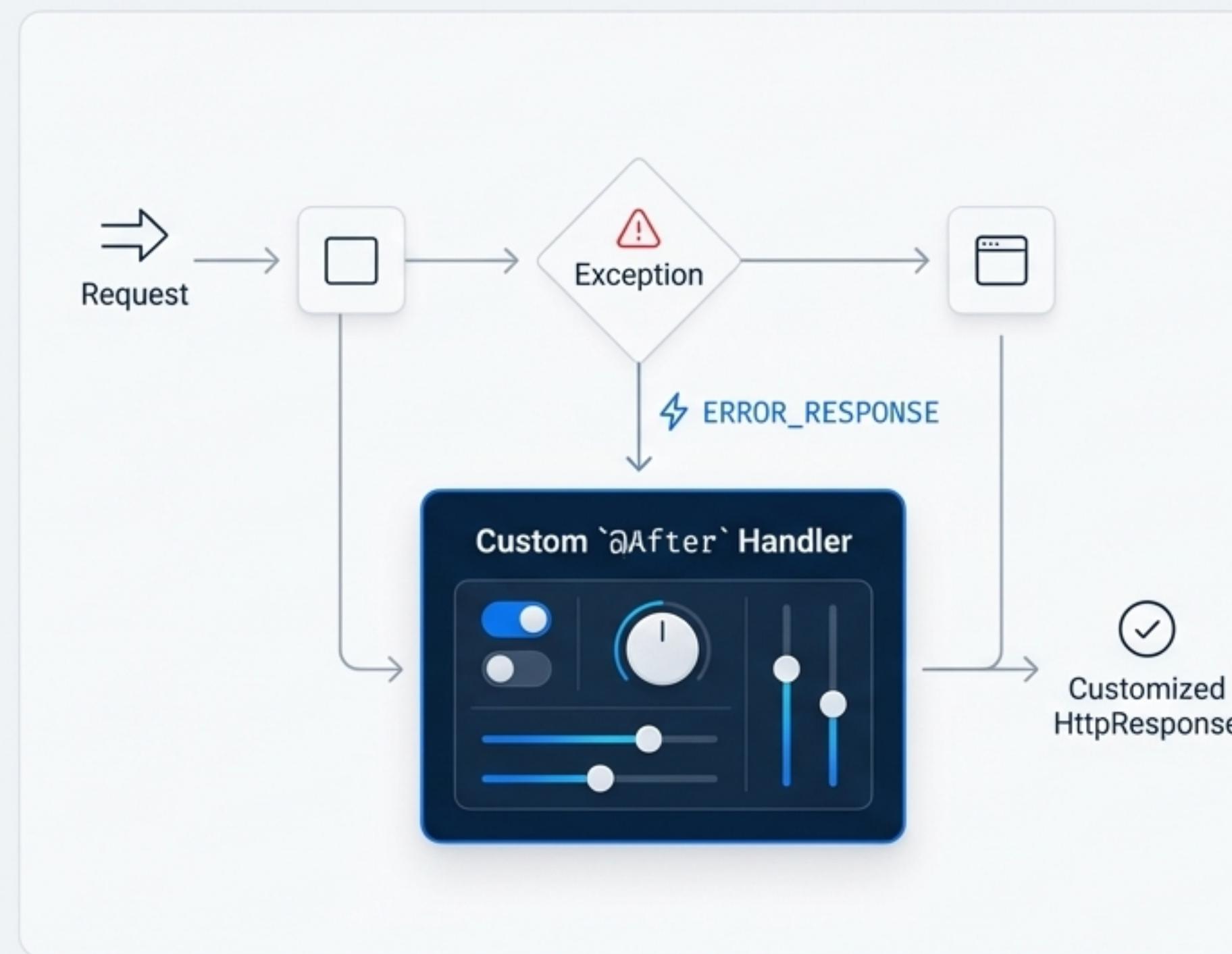
```
// Target no campo 'firstName' do
// parâmetro 'reviewer'
throw new ServiceException(
    ErrorStatuses.BAD_REQUEST,
    "Invalid reviewer first name")
    .messageTarget("reviewer", r
        -> r.get("firstName"));

// Target no parâmetro 'rating'
throw new ServiceException(
    ErrorStatuses.BAD_REQUEST,
    "Invalid review rating")
    .messageTarget("rating");
```

</> Referenciando a Entidade Vinculada

```
// Target no campo 'descr' da
// entidade 'Books'
throw new ServiceException(
    ErrorStatuses.BAD_REQUEST,
    "Invalid book description")
    .messageTarget(Books_.class,
        b -> b.descr());
```

Nível Mestre: Controle Total com Error Handlers Personalizados



⚡ O Gatilho

- Quando uma exceção é lançada, o CAP Java dispara o evento `ERROR_RESPONSE` do `ApplicationLifecycleService` antes de enviar a resposta ao cliente.

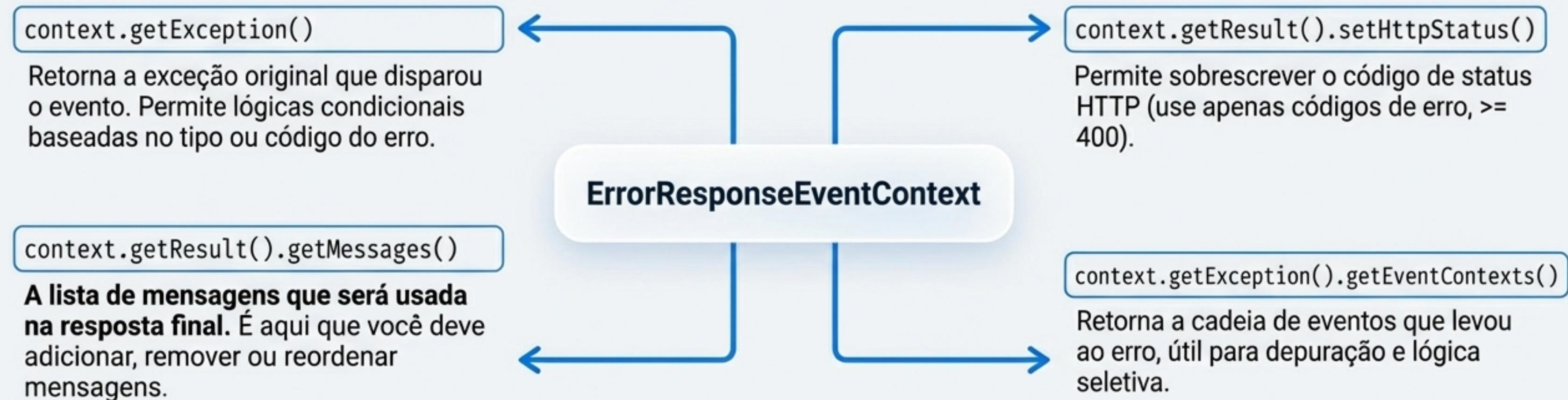
⌚ A Oportunidade

- Você pode implementar um handler com `@After` para este evento e personalizar a resposta de erro.

☰ O Que Você Pode Fazer

- **Augmentar:** Adicionar informações de contexto à resposta.
- **Modificar:** Alterar textos de erro, reordenar mensagens.
- **Substituir:** Criar uma resposta de erro completamente nova.
- **Alterar Status:** Mudar o código de status HTTP resultante.

Anatomia de um `ErrorResponseEventContext`



AVISO CRÍTICO



Não adicione mensagens ao `context.getMessages()` (o `Messages` do `RequestContext`). Elas **não** serão incluídas na resposta final. Manipule apenas a lista obtida via `context.getResult().getMessages()`.

Exemplo Prático: Sobrescrevendo a Mensagem de Autorização

Objetivo

Substituir a mensagem padrão para erros de autorização por um texto mais amigável.

Implementação

- Crie uma classe que implementa `EventHandler`.
- Anote com `@ServiceName(ApplicationLifecycleService.DEFAULT_NAME)`.
- Crie um método com `@After` que recebe `ErrorResponseEventContext`.
- Dentro do método, verifique o `ErrorStatus` da exceção e modifique a lista de mensagens.

```
@Component  
@ServiceName(ApplicationLifecycleService.DEFAULT_NAME)  
public class SimpleExceptionHandler implements EventHandler {  
  
    @After  
    public void overrideMissingAuthMessage(ErrorResponseEvent  
Context context) {  
  
        if (context.getException().getErrorStatus().equals(Cds  
ErrorStatuses.EVENT_FORBIDDEN)) {  
            // Substitui a primeira (principal) mensagem de erro  
            context.getResult().getMessages().set(0,  
                Message.create(Message.Severity.ERROR, "You cannot  
execute this action"));  
        }  
    }  
}
```

Resumo Estratégico: O Tratamento de Erros Ideal

Checklist para Desenvolvedores CAP Java



Para Validações: Use a `Messages API` para coletar todos os erros em um único passo e confie na chamada automática de `throwIfError()` ao final da fase `@Before`.



Para Falhas Críticas: Lance uma `ServiceException` diretamente com um `ErrorStatus` apropriado (ex: `CONFLICT`, `BAD_REQUEST`). É a forma mais clara de sinalizar uma falha irrecuperável.



Para UI Fiori: Sempre que possível, use `messageTarget()` com uma `ServiceException` para destacar os campos exatos na interface e melhorar a experiência do usuário.



Para Personalização Total: Quando as respostas padrão não são suficientes, implemente um handler para o evento `ERROR_RESPONSE` para ter controle absoluto sobre a saída.



Sempre: Prefira chaves de internacionalização (`messages.properties`) em vez de texto fixo no código. Sua aplicação estará pronta para o mundo.