

Dominando os Dados no CAP Java

Uma Jornada do Básico ao Avançado para Desenvolvedores

Aprenda a representar, manipular e processar dados de forma eficiente e robusta no SAP Cloud Application Programming Model. Esta apresentação é o seu guia definitivo para trabalhar com dados CDS.

A Base de Tudo: `CdsData` é um `Map` com Superpoderes

No CAP Java, toda representação de dados estruturados (entidades, tipos) começa com a interface `java.util.Map<String, Object>`.

CAP estende esta interface com `com.sap.cds.CdsData`, que adiciona métodos de conveniência essenciais para manipulação de dados.



```
// Dados de entidades, tipos estruturados e elementos cds.Map  
// são representados como:  
Map<String, Object> book = new HashMap<>();  
book.put("ID", 97);  
book.put("title", "Dracula");  
  
// CdsData oferece uma interface aprimorada para este Map.  
CdsData data = Struct.access(book).as(CdsData.class);
```

Isto significa que você obtém a flexibilidade de um Map com a robustez das ferramentas do CAP.

A Tradução Essencial: Mapeamento de Tipos CDS para Java

Para garantir consistência e previsibilidade, o CAP Java define um mapeamento claro entre os tipos predefinidos do CDS e os tipos Java.

Inter Regular (#212529)

Tabela 1: Tipos Predefinidos do CDS

Inter Medium (#212529)

Tipo CDS	Tipo Java	Observação
cds.UUID	java.lang.String	
cds.Boolean	java.lang.Boolean	
cds.Int32, cds.Integer	java.lang.Integer	
cds.Int64, cds.Integer64	java.lang.Long	
cds.Decimal	java.math.BigDecimal	
cds.Double	java.lang.Double	
cds.Date	java.time.LocalDate	data sem fuso horário
cds.Time	java.time.LocalTime	hora sem fuso horário
cds.Timestamp	java.time.Instant	precisão de µs
cds.String	java.lang.String	
cds.LargeString	java.io.Reader	se anotado com @Core.MediaType
cds.LargeBinary	java.io.InputStream	se anotado com @Core.MediaType
cds.Vector	com.sap.cds.CdsVector	
cds.Map	java.util.Map	

Tabela 2: Tipos Específicos do SAP HANA (para compatibilidade)

Inter Medium (#212529)

Tipo CDS	Tipo Java
hana.TINYINT, hana.SMALLINT	java.lang.Short
hana.REAL	java.lang.Float
hana.CLOB	java.io.Reader (se anotado com @Core.MediaType)

Representando Estruturas: Entidades e Associações

Entidades e tipos estruturados no CDS são representados como mapas em Java, onde as chaves são os nomes dos elementos e os valores são seus respectivos dados.

Exemplo Lado a Lado: Entidade Simples

JSON Payload

```
{  
  "ID": 97,  
  "title": "Dracula"  
}
```

Construção em Java

```
Map<String, Object> book = new HashMap<>();  
book.put("ID", 97);  
book.put("title", "Dracula");
```

Exemplo Lado a Lado: Associação "To-One" Aninhada

JSON Payload

```
{  
  "ID": 97,  
  "author": { "ID": 23, "name": "Bram Stoker" }  
}
```

Construção em Java

```
Map<String, Object> author = new HashMap<>();  
author.put("ID", 23);  
author.put("name", "Bram Stoker");
```

```
Map<String, Object> book = new HashMap<>();  
book.put("ID", 97);  
book.put("author", author); // O valor é outro Map
```

Lidando com Relações: Associações "To-Many"

Uma associação "to-many" ou uma composição de muitos é representada em Java como uma `List<Map<String, Object>>`.

JSON Payload

```
{  
  "ID": 23,  
  "name": "Bram Stoker",  
  "books": [  
    { "ID": 97, "title": "Dracula" },  
    { "ID": 98, "title": "Miss Betty" }  
]
```

Construção em Java

```
Map<String, Object> book1 = new HashMap<>();  
book1.put("ID", 97); book1.put("title", "Dracula");  
  
Map<String, Object> book2 = new HashMap<>();  
book2.put("ID", 98); book2.put("title", "Miss Betty");  
  
Map<String, Object> author = new HashMap<>();  
author.put("ID", 23);  
author.put("name", "Bram Stoker");  
author.put("books", Arrays.asList(book1, book2));  
// O valor é uma Lista de Mapas
```

A Magia do Path Access: Manipulando Dados Aninhados sem Esforço

O Problema

Acessar dados em mapas aninhados requer múltiplas verificações para evitar `NullPointerException` e código verboso.

Antes e Depois

```
if (map != null) {  
    if (map != null) {  
        if (map != null) (author.) {  
            if (map != null) {  
                if (map != null) (factors) {  
                    if (map != null) {  
                        data.putPath("author.name"); ...  
                }  
            }  
        }  
    }  
}
```

Faded: Complex null checks & boilerplate

A Solução CAP

A interface `CdsData` fornece métodos de acesso por "caminho" (`path`) que são nulos-seguros.

Demonstração: Escrevendo Dados (`putPath`)

```
// Mapas aninhados são criados sob demanda, de forma segura.  
CdsData book = Struct.create(CdsData.class);  
book.put("ID", 97);  
book.putPath("author.ID", 23);  
book.putPath("author.name", "Bram Stoker");
```

Demonstração: Lendo e Removendo Dados

getPath

```
String authorName =  
data.getPath("author.name"); // Retorna  
null se não existir
```

containsPath

```
boolean exists =  
data.containsPath("author.name");
```

removePath

```
data.removePath("author.name"); //  
Remove e limpa mapas vazios
```

Use os métodos de `path access` para manipular estruturas de dados complexas com conveniência e segurança.

Dados em Ação: Estruturas de Dados na CDS Query Language (CQL)

As estruturas de `Map` que representam entidades são o payload para operações de escrita (Insert, Update) no banco de dados.



Deep Inserts (Criação Aninhada)

Criar uma entidade e suas entidades filhas (via composição) em uma única operação.

```
// Criando uma Order com seu Header em uma única chamada
OrderHeaders header = OrderHeaders.create();
header.setStatus("open");

Orders order = Orders.create();
order.setHeader(header); // Aninhando a composição

Insert insert = Insert.into(ORDERS).entry(order);
```



Associações Gerenciadas

Associar uma entidade a outra já existente. O framework cuida das chaves estrangeiras.

```
// Associando um novo Book a um Author existente
Authors author = Authors.create();
author.setId(100); // Apenas a chave do autor existente é necessária

Books book = Books.create();
book.setAuthor(author);

Insert insert = Insert.into(BOOKS).entry(book);
```

Sempre defina associações gerenciadas usando o elemento da associação, evitando o uso direto de elementos de chave estrangeira.

Elevando seu Código com Acesso Tipado

A Dor do `Map<String, Object>`

- **X Sem Type Safety:** Risco de `ClassCastException` em tempo de execução.
- **X Sem Code Completion na IDE:** Nomes de elementos são strings, propensos a erros de digitação.
- **X Verificação em Runtime:** Nomes de elementos só são validados durante a execução.

A Solução: Interfaces de Acesso

Interfaces Java que estendem `CdsData` e fornecem métodos `get/set` para os elementos do CDS.

```
// Uma interface para a entidade Books
interface Books extends Map<String, Object> { // Estender Map

    @CdsName("ID")      // Anotação para mapear o elemento CDS
    IntegergetID();

    StringgetTitle();
    voidsetTitle(String title);
}
```

Como Usar

```
import static com.sap.cds.Struct.access;
...
// Cria um proxy que dá acesso tipado aos dados do Map
Books book = access(data).as(Books.class);

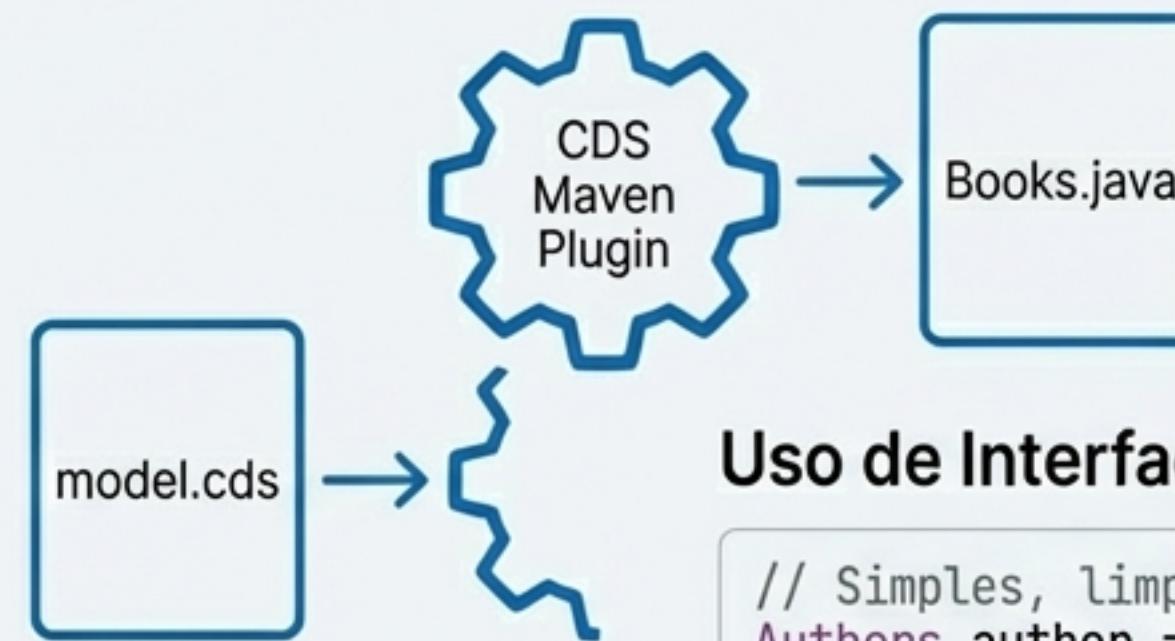
String title = book.getTitle();    // Leitura tipada e segura
book.setTitle("Miss Betty");     // Escrita tipada
```

O Poder da Automação: Interfaces Geradas pelo CDS Maven Plugin

Você não precisa escrever essas interfaces manualmente! O **CDS Maven Plugin** as gera automaticamente a partir do seu modelo CDS durante o build.

Benefícios

- **🚀 Produtividade Máxima:** Código boilerplate gerado para você.
- **⌚ Consistência:** Garante que o código Java esteja sempre sincronizado com o modelo CDS.
- **✅ Conveniência:** As interfaces geradas já incluem métodos estáticos `create()`.



Uso de Interface Gerada

```
// Simples, limpo e com type-safety
Authors author = Authors.create().name("Emily Brontë");

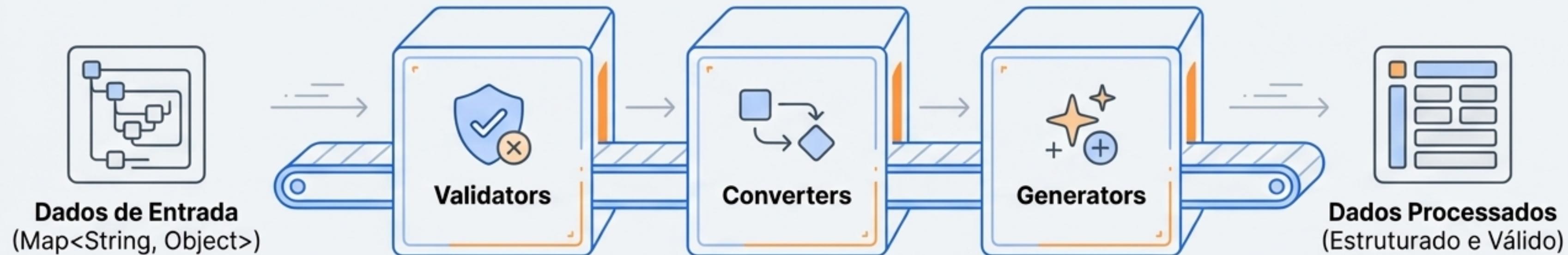
Books book = Books.create()
    .title("Wuthering Heights")
    .author(author);
```

Customização Avançada

- **Renomeando Elementos:** Use a anotação `@cds.java.name` no modelo CDS para evitar conflitos com palavras-chave do Java.
- **Herança:** Use `@cds.java.extends` para criar uma hierarquia de herança entre as interfaces geradas (por exemplo, para aspects).

O Guardião dos Dados: Validando e Transformando com `CdsDataProcessor`

Pense no `CdsDataProcessor` como uma linha de montagem para seus dados. Ele processa estruturas de dados profundamente aninhadas, aplicando uma sequência de ações.



Como Funciona

1. Crie uma instância: `CdsDataProcessor processor = CdsDataProcessor.create();`
2. Adicione ações com filtros:

```
// Adiciona um validador para o elemento 'quantity'
processor.addValidator(
    (path, element, type) -> element.getName().equals("quantity"), // Filtro
    (path, element, value) -> {
        if (int) value < 0) { // Ação
            log.warn("Quantidade negativa: " + path.toRef());
        }
    });
});
```

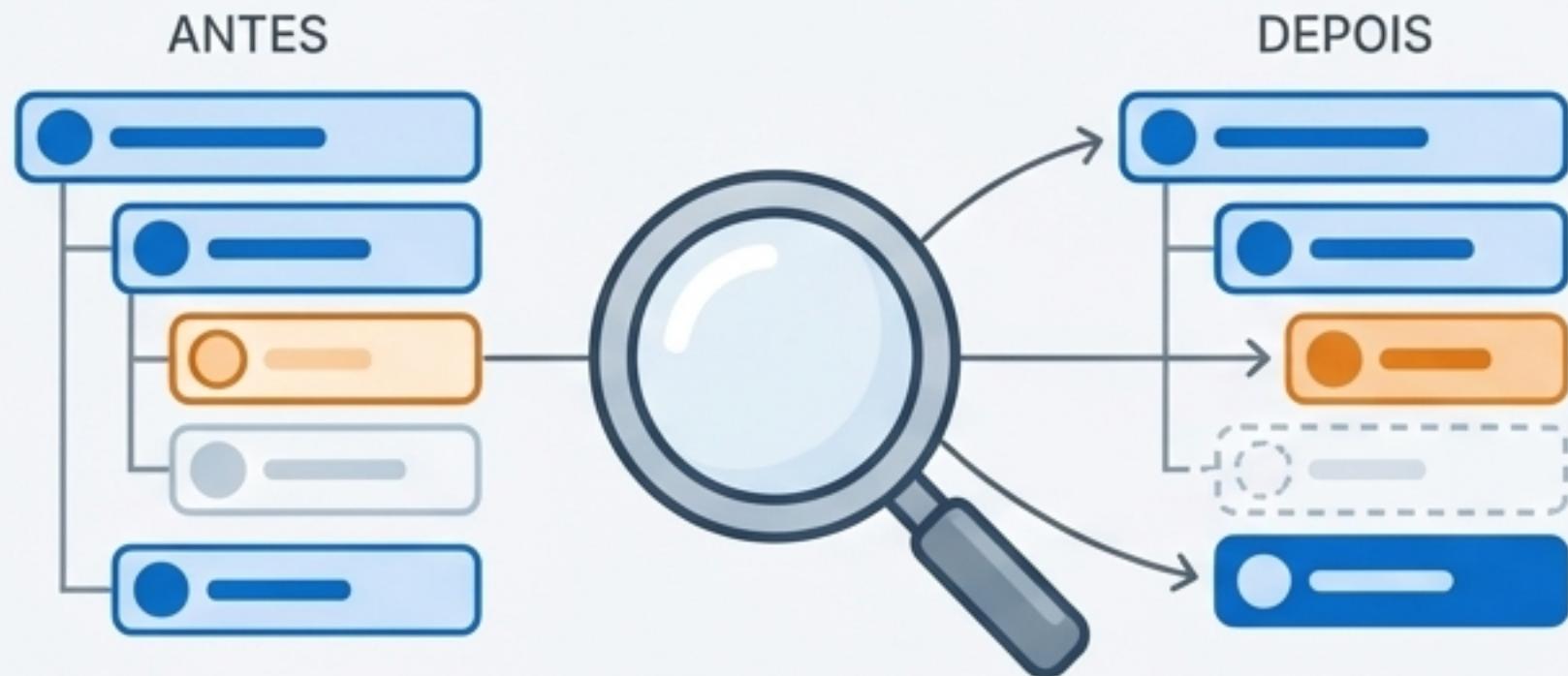
3. Execute o processamento: `processor.process(data, rowType);`

Modos de Validação

Controle a execução com modos de validação: `CONTAINS` (padrão), `NULL` e `DECLARED`.

O Detetive de Mudanças: Comparando Snapshots com `CdsDiffProcessor`

Use Case: Reagir a mudanças nos dados, comparando a imagem de uma entidade *antes* e *depois* de uma operação.



Como Funciona

1. Crie o processador: `CdsDiffProcessor diff = CdsDiffProcessor.create();`
2. Implemente um `DiffVisitor` para tratar as mudanças.
3. Adicione o visitor e processe as duas imagens de dados (antiga e nova).

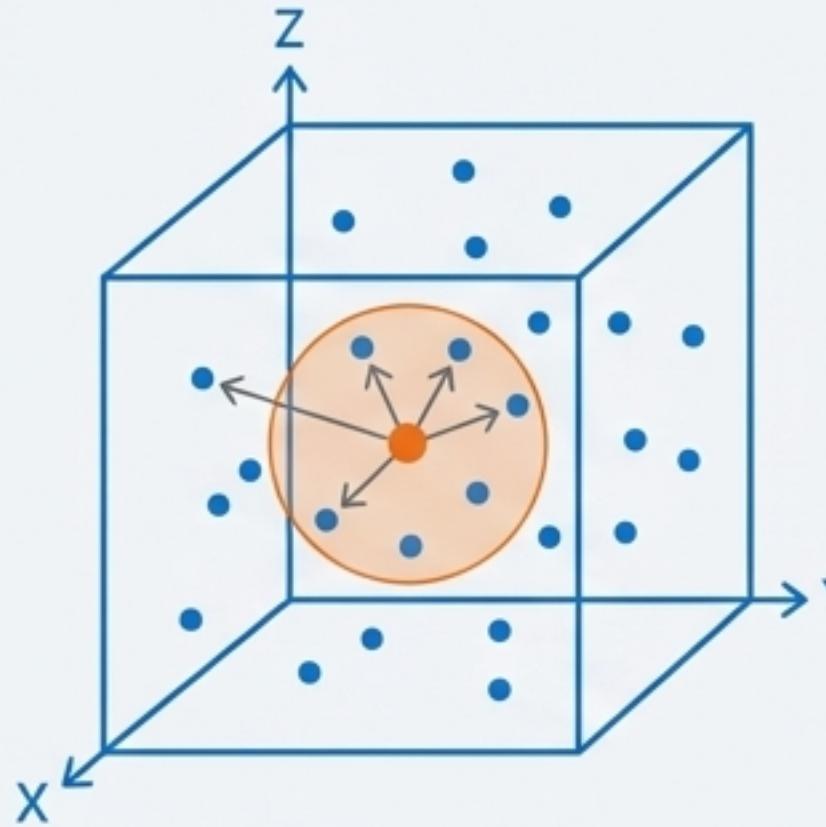
Exemplo de `DiffVisitor`

```
diff.add(new DiffVisitor() {  
    @Override  
    public void changed(Path newPath, Path oldPath, CdsElement  
        element, Object newValue, Object oldValue) {  
        // Lógica para valores alterados  
    }  
    @Override  
    public void added(Path newPath, Path oldPath, CdsElement  
        association, Map<String, Object> newValue) {  
        // Lógica para dados/entidades adicionados  
    }  
    @Override  
    public void removed(Path newPath, Path oldPath, CdsElement  
        association, Map<String, Object> oldValue) {  
        // Lógica para dados/entidades removidos  
    }  
});  
  
diff.process(newImage, oldImage, type);
```

O processador percorre as duas estruturas, identifica as diferenças (adições, remoções, alterações) e notifica seu visitor, simplificando a lógica de detecção de mudanças.

A Fronteira da IA: Trabalhando com Vector Embeddings

Para aplicações de IA, como busca semântica e sistemas de recomendação, é crucial trabalhar com embeddings de vetores.



Poder em Ação: Funções de Similaridade em CQL

Use `CQL.cosineSimilarity` ou `CQL.l2Distance` diretamente em suas queries para encontrar os dados mais relevantes.

Encontrando livros com descrições similares.

```
// embedding é um float[] obtido de um modelo de IA  
CqnVector v = CQL.vector(embedding);  
  
// Encontra livros cuja similaridade de cosseno do embedding é > 0.9  
CdsResult<Books> similarBooks = service.run(Select.from(BOOKS)  
    .where(b -> CQL.cosineSimilarity(b.embedding(), v).gt(0.9))  
);
```

Suporte no CAP Java

- **Tipo CDS:** `cds.Vector(dimensions)`
(ex: `Vector(1536)`)
- **Tipo Java:** `com.sap.cds.CdsVector`
(pode ser criado a partir de `float[]` ou `String`)



Elementos `cds.Vector` não são incluídos em `SELECT *`. Eles devem ser selecionados explicitamente.

Lidando com Streams: Processamento de Media Types

Elementos de mídia (imagens, documentos) são anotados com `@Core.MediaType` e representados em Java como `InputStream` (para `LargeBinary`) ou `Reader` (para `LargeString`).



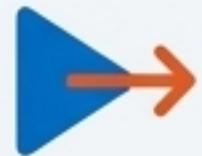
```
// cada example model
entity Books : cuid, managed {
    title      : String(111);
    coverImage : LargeBinary
        @Core.MediaType: 'image/png';
}
```

Streams não são resetáveis.

Os dados só podem ser lidos **UMA VEZ**.

- Uma vez que seu código consome o stream (lê até o final), ele fica vazio.
- Passar um stream já consumido para outra parte do código (como o handler padrão de persistência) resultará em nenhum dado sendo escrito.
- Isso requer um cuidado especial ao implementar lógica customizada.

Padrões de Processamento para Media Types



Cenário 1: Nenhuma Lógica Customizada (O Padrão)

Você só quer salvar o arquivo no banco de dados (upload) ou servi-lo ao cliente (download).

Solução: Não faça nada! Os handlers padrão do CAP cuidam de tudo para você.



Cenário 2: Lógica Customizada com Consumo Total

Você precisa processar o conteúdo completo do stream (ex: parsear um CSV, validar um arquivo).

Solução: Use um handler `@On``. Consuma o stream e **não** chame o handler padrão.

```
@On(event = CqnService.EVENT_UPDATE)
public void process(CdsUpdateEventContext ctx,
List<Books> books) {
    books.forEach(book -> {
        InputStream is = book.getCoverImage();
        // ... seu código que consome totalmente o
        stream ...
    });
    ctx.setResult(books); // Evita o handler padrão
}
```



Cenário 3: Pré/Pós-Processamento com Proxy (Inspeção)

Você precisa inspecionar/modificar os dados do stream ***antes*** que ele seja persistido (ex: scan de vírus).

Solução: Crie um wrapper (Proxy) que estende `FilterInputStream`. Substitua o stream original pelo seu proxy em um handler `@Before`` (upload) ou `@After`` (download).

```
@Before(event = CqnService.EVENT_UPDATE)
public void preProcess(List<Books> books) {
    books.forEach(book -> {
        book.setCoverImage(new
VirusScannerProxy(book.getCoverImage()));
    });
} // Handler padrão consome via proxy
```

Sua Jornada para a Maestria de Dados no CAP Java



Os Pilares do Domínio de Dados

- A Fundação (`CdsData`)**: Tudo é um `Map<String, Object>`, mas aprimorado. É a base flexível para toda a manipulação de dados.
- A Conveniência (`Path Access`)**: Manipule estruturas aninhadas com confiança e código limpo, sem `NullPointerException`.
- A Robustez (`Typed Access`)**: Eleve seu código com segurança de tipos, autocompletar da IDE e refatoração segura, graças às interfaces geradas automaticamente.
- A Automação (`Processors`)**: Use `CdsDataProcessor` e `CdsDiffProcessor` para automatizar tarefas complexas de validação, transformação e detecção de mudanças.
- A Modernidade (`Vector` & `Streams`)**: Esteja preparado para os desafios modernos, desde aplicações de IA até o manuseio eficiente de grandes arquivos de mídia.

Com estas ferramentas e conceitos, você está equipado para construir aplicações robustas, eficientes e escaláveis com o CAP Java.