



Audit Logging em CAP Java: Dominando o `AuditLogService`

Um guia para registrar eventos de auditoria de forma
transacional, resiliente e segura.

O que é o `AuditLogService` e por que ele é essencial?

Introdução de alto nível. O `AuditLogService` é um serviço dedicado no CAP Java (a partir da versão 1.18.0) para emitir eventos críticos de auditoria de forma estruturada. Ele eleva o logging de auditoria de uma tarefa manual para um recurso suportado e integrado ao framework.



Acesso a Dados Pessoais



Modificação de Dados Pessoais



Mudanças de Configuração



Eventos de Segurança

A ferramenta certa para os quatro pilares da auditoria.

Passo 1: Obtenha uma instância do serviço

Existem duas maneiras principais de acessar o `AuditLogService`. A injeção de dependência é a abordagem recomendada para projetos Spring Boot devido à sua simplicidade e clareza.

Com Injeção de Dependência (`@Autowired`)



```
import com.sap.cds.services.auditlog.  
AuditLogService;  
  
@Autowired  
private AuditLogService auditLogService;
```

Via `ServiceCatalog`

```
ServiceCatalog catalog =  
context.getServiceCatalog();  
auditLogService = (AuditLogService)  
catalog.getService(AuditLogService.DEFAULT  
_NAME);
```

Use a injeção de dependência com `@Autowired` para um código mais limpo e manutenível.

Registrando acesso e modificação de dados pessoais

Os métodos `logDataAccess` e `logDataModification` são projetados especificamente para rastrear interações com dados sensíveis, um requisito fundamental para conformidade com regulamentações de privacidade.

Rastreando Acesso a Dados (`logDataAccess`)

```
List<Access> accesses = new ArrayList<>();  
Access access = Access.create();  
// preencha o objeto 'access' com os dados  
accesses.add(access);  
auditLogService.logDataAccess(accesses);
```

Rastreando Modificação de Dados (`logDataModification`)

```
List<DataModification> dataModifications = new ArrayList<>();  
DataModification modification = DataModification.create();  
// preencha o objeto 'modification' com os dados  
dataModifications.add(modification);  
auditLogService.logDataModification(dataModifications);
```

Rastreando mudanças de configuração e eventos de segurança

Além dos dados pessoais, é crucial auditar alterações na configuração do sistema e eventos de segurança, como tentativas de login ou escalonamento de privilégios.

Registrando Mudanças de Configuração (`logConfigChange`)

```
List<ConfigChange> configChanges = new ArrayList<>();  
ConfigChange configChange = ConfigChange.create();  
// preencha o objeto 'configChange' com os dados  
configChanges.add(configChange);  
auditLogService.logConfigChange(Action.UPDATE, configChanges);
```

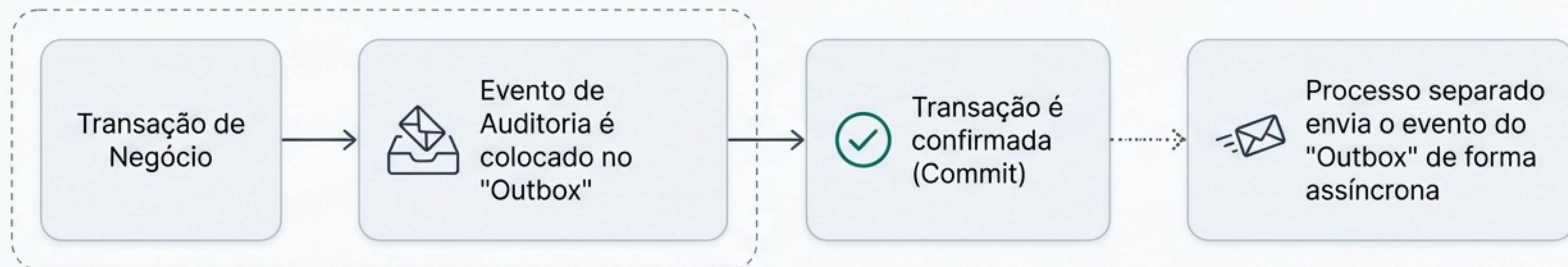
Registrando Eventos de Segurança (`logSecurityEvent`)

```
String action = "login";  
String data = "user-name";  
auditLogService.logSecurityEvent(action, data);
```

Cobertura completa dos quatro pilares da auditoria com uma API consistente e intuitiva.

O segredo da resiliência: Envio assíncrono com o serviço Outbox

O `AuditLogService` não envia eventos diretamente. Em vez disso, ele utiliza um serviço Outbox interno para desacoplar a transação de negócio do envio do log de auditoria. O evento é armazenado na mesma transação da requisição. Se a transação falhar, o evento de auditoria também é descartado.



Seus logs de auditoria estão vinculados atomicamente à sua transação de negócio.

Por que o envio assíncrono é uma virada de jogo

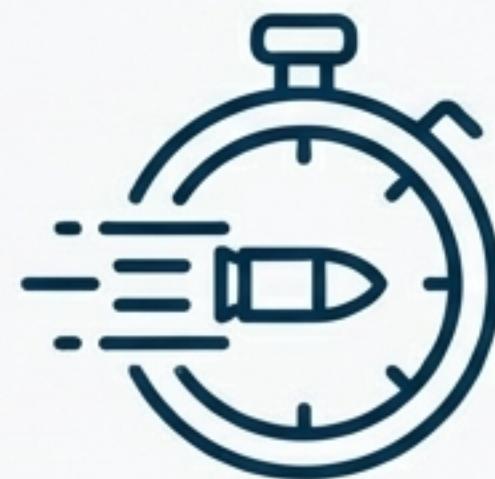
O desacoplamento proporcionado pelo padrão Outbox oferece benefícios críticos para aplicações de nível empresarial.



Integridade Transacional

Eventos de auditoria são cancelados se a transação principal falhar (rollback).

Nada é registrado incorretamente.



Performance

A requisição do usuário não é bloqueada esperando pela chamada de rede ao serviço de log central. A resposta é mais rápida.

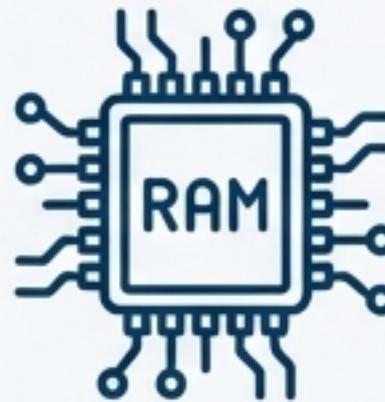


Resiliência

Falhas temporárias na comunicação com o serviço de log não interrompem a transação de negócio. O sistema continua operando.

Outbox em memória vs. persistente: Escolhendo a garantia de entrega

O serviço Outbox pode operar em dois modos, com diferentes garantias. A escolha correta é fundamental para a confiabilidade do seu sistema de auditoria.



Default (Em Memória)

Rápido e simples, ideal para desenvolvimento.



Persistente (Recomendado para Produção)

A solução robusta. Requer configuração adicional para persistir os eventos no banco de dados.

⚠ Não há garantia de entrega dos eventos em caso de reinicialização ou falha da aplicação após o commit da transação, mas antes do envio.

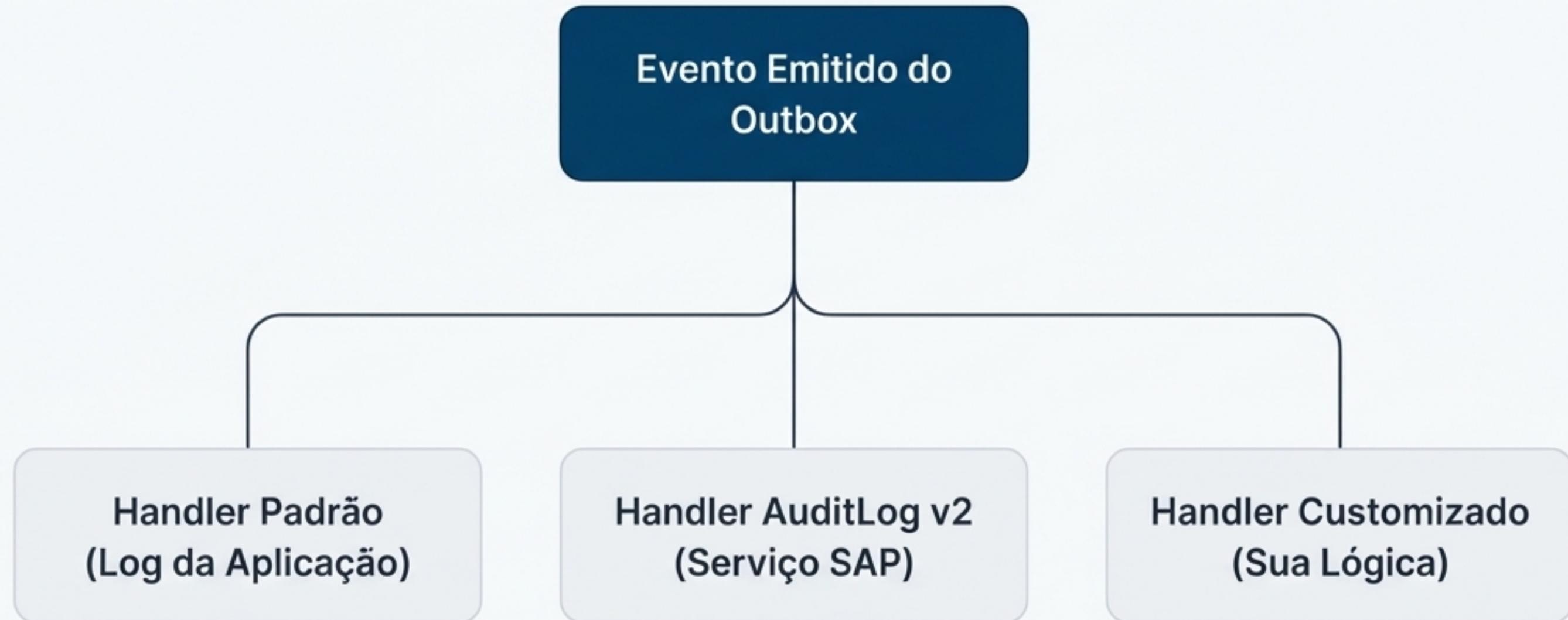
✓ Garante que todos os eventos de transações bem-sucedidas sejam processados e enviados, mesmo em caso de falhas.

Para ambientes de produção, SEMPRE configure um outbox persistente para máxima confiabilidade e conformidade.

*Eventos de segurança ('logSecurityEvent') são sempre enviados de forma síncrona, independentemente da configuração do outbox.

Para onde vão os eventos? Conhecendo os AuditLog Handlers

Uma vez que um evento é liberado do Outbox, ele é entregue a um ou mais 'handlers' registrados. O CAP Java oferece flexibilidade para processar esses eventos de diferentes maneiras, dependendo das necessidades da sua aplicação.



Opção 1: O Handler Padrão (Logs na Aplicação)

Por padrão, o CAP Java inclui um handler que escreve as mensagens de auditoria no log da aplicação. Este handler é registrado para todos os eventos, mas por padrão opera em nível `DEBUG`, o que significa que você precisa ativá-lo explicitamente.

```
`srv/src/main/resources/application.yaml`
```

```
logging:  
  level:  
    com.sap.cds.auditlog: DEBUG
```

Simples e extremamente útil para desenvolvimento e depuração local, mas não adequado para auditoria formal em produção.

Opção 2: Integração nativa com o Serviço AuditLog v2

Para uma solução de auditoria completa e centralizada, o CAP Java oferece um handler que se integra automaticamente ao serviço AuditLog v2 (plano premium). Se a dependência presente e um service binding existir, a integração é automática.

Dependência Maven

```
<dependency>
  <groupId>com.sap.cds</groupId>
  <artifactId>cds-feature-auditlog-v2</artifactId>
  <scope>runtime</scope>
</dependency>
```

Desativação (Opcional)

```
cds:
  auditlog.v2.enabled: false
```

A rota automática para integração de nível enterprise. O valor padrão de `enabled` é `true`.

Opção 3: Controle total com um Handler customizado

Quando você precisa de uma lógica de processamento específica (ex: enviar para um sistema SIEM diferente, aplicar filtros personalizados), você pode implementar seu próprio **handler**. A classe deve implementar a interface `EventHandler`.

- **@Component**: Para registrar o handler no contexto do Spring.
- **@ServiceName(...)**: Para escutar eventos de todas as fontes do `AuditLogService`.
- **@On**: Para marcar o método que processará o evento.

```
@Component  
@ServiceName(value = "*", type = AuditLogService.class)  
class CustomAuditLogHandler implements EventHandler {  
  
    @On  
    public void handleDataAccessEvent(DataAccessLog  
Context context) { ... }  
  
    @On  
    public void handleDataModificationEvent  
(DataModificationLogContext context) { ... }  
  
    @On  
    public void handleConfigChangeEvent  
(ConfigChangeLogContext context) { ... }  
  
    @On  
    public void handleSecurityEvent(SecurityLogContext  
context) { ... }  
}
```

Exemplo de implementação de um Handler customizado

Veja um exemplo completo de uma classe de handler que implementa métodos para os quatro tipos de eventos. Este código serve como um ponto de partida para sua própria lógica de negócios.

```
import com.sap.cds.services.auditlog.*;
import com.sap.cds.services.handler.*;
import com.sap.cds.services.handler.annotations.*;
import org.springframework.stereotype.Component;

@Component
@ServiceName(value = "*", type = AuditLogService.class)
class CustomAuditLogHandler implements EventHandler {

    @On
    public void handleDataAccessEvent(DataAccessLogContext context) {
        // Lógica customizada para eventos de acesso a dados
        // Ex: context.getAccesses().forEach(...)
    }

    @On
    public void handleDataModificationEvent(DataModificationLogContext context) {
        // Lógica customizada para eventos de modificação de dados
    }

    @On
    public void handleConfigChangeEvent(ConfigChangeLogContext context) {
        // Lógica customizada para eventos de mudança de configuração
    }

    @On
    public void handleSecurityEvent(SecurityLogContext context) {
        // Lógica customizada para eventos de segurança
    }
}
```

@Component → Registra o bean no Spring
@ServiceName(value = "*", type = AuditLogService.class) → Escuta todos os eventos do AuditLogService
@On → Define o método como um handler de evento

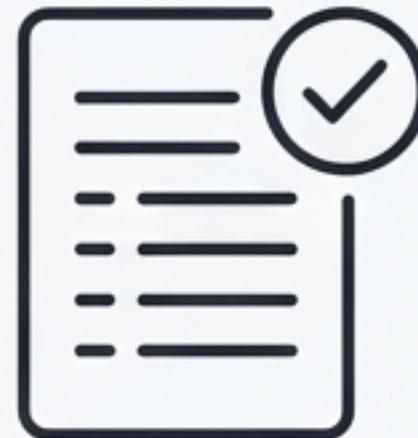


⚠ Atenção: Compliance e Privacidade de Dados são sua responsabilidade

A ferramenta é poderosa, mas a responsabilidade final pela conformidade é do desenvolvedor. Considere estes pontos críticos:

- **Outbox Persistente:** Ao usar um outbox persistente, eventos de auditoria contendo dados pessoais podem ser armazenados temporariamente no banco de dados da aplicação. Garanta que as políticas de acesso, criptografia e retenção de dados da sua aplicação estejam em conformidade com as regras de privacidade.
- **Privilégios de Operador:** Tenha cuidado ao persistir logs de auditoria de usuários com privilégios de operador de banco de dados. Um usuário com acesso direto ao DB poderia, teoricamente, modificar seus próprios registros de auditoria antes que fossem enviados.

Resumo da Jornada: Seus Pontos-Chave



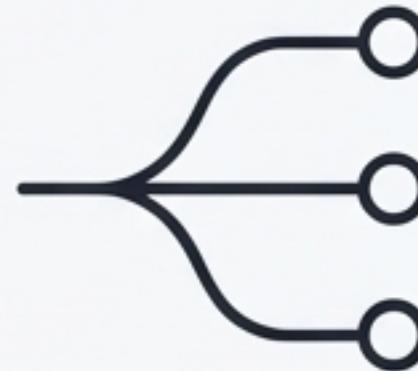
Use o `AuditLogService`:

Para um registro de auditoria estruturado, transacional e robusto.



Entenda o Outbox:

O design assíncrono é a chave para performance e resiliência. Use o outbox persistente em produção.



Escolha seu Handler: Use o padrão para depuração, o v2 para integração com o serviço SAP, ou crie um customizado para necessidades específicas.



Priorize a Conformidade:

Sempre considere as implicações de privacidade e segurança do armazenamento de logs de auditoria.

Para mais detalhes, consulte a [documentação oficial do CAP Java](#).