

Do Mágico ao Metódico



Dominando o `cds.ApplicationService` no SAP CAP

Um guia para desenvolvedores Node.js que desejam ir além do padrão e controlar o coração de seus serviços.

A "Mágica" que Todo Desenvolvedor CAP Conhece

Você define um serviço simples em CDS. Sem uma única linha de código de implementação, você já tem endpoints CRUD, suporte a Fiori, paginação e muito mais.

```
1 // srv/admin-service.cds
2 service AdminService {
3     entity Authors as projection on my.Authors;
4     entity Books as projection on my.Books;
5     entity Genre as projection on my.Genre;
6 }
```



Como isso é possível? De onde vem todo esse comportamento padrão?

A Resposta: O Motor Padrão `cds.ApplicationService`

Quando o CAP não encontra uma implementação personalizada para um serviço (um arquivo ` `.js`), ele não falha. Em vez disso, ele instancia automaticamente a classe `cds.ApplicationService` .



Esta classe é a implementação padrão de provedor de serviços, projetada para ser robusta e extensível.

```
// O que o `cds.serve` faz por baixo dos panos:  
let name = 'AdminService', options = {...}  
  
// Instancia a classe padrão para o nosso serviço  
let srv = new cds.ApplicationService(name, cds.model, options)  
  
await srv.init() // ← A inicialização que ativa toda a mágica
```

A Arquitetura do Motor: `init()` e os Handlers Genéricos

Dentro de `cds.ApplicationService`, o método `init()` executa uma rotina crucial: ele invoca uma série de métodos estáticos da própria classe, cada um responsável por registrar um conjunto de "handlers genéricos".

```
class cds.ApplicationService extends cds.Service {  
    init() {  
        // 1. Identifica todos os métodos estáticos que começam com 'handle_'  
        const generics = [  
            'handle_authorization', 'handle_etags',  
            'handle_validations', /* ...e todos os outros... */  
        ]  
  
        // 2. Executa cada um deles para registrar os handlers  
        for (let each of generics) {  
            this[each].call(this)  
        }  
  
        return super.init()  
    }  
    // ... definição dos métodos estáticos handle_*  
}
```

Ciclo de Registro



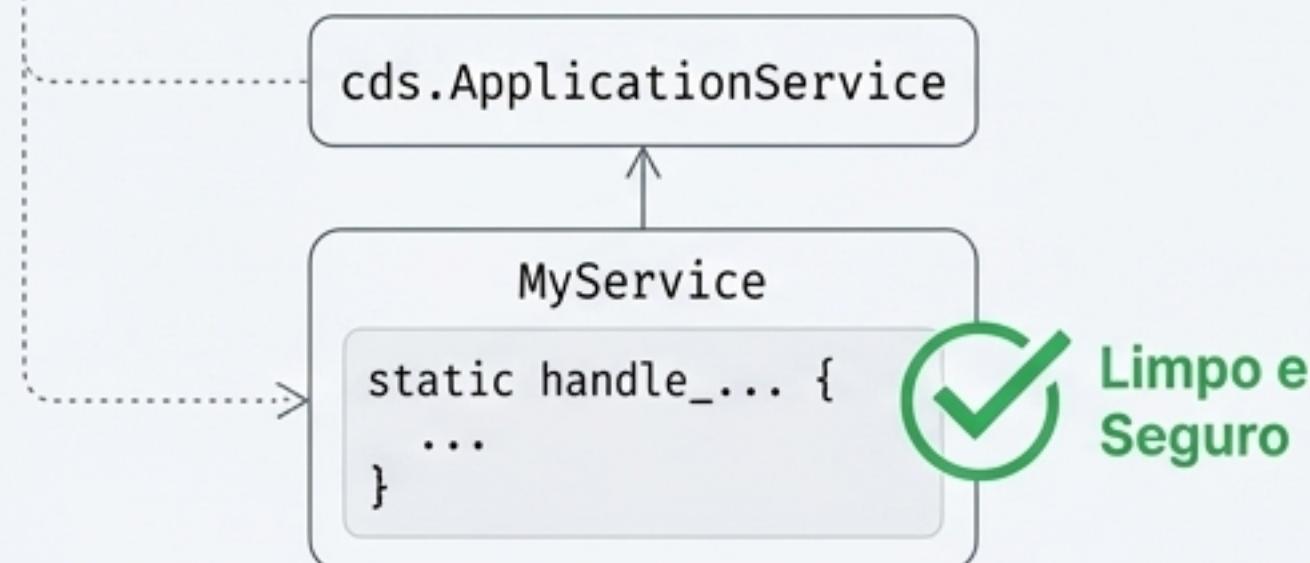
Uma Decisão de Design Inteligente: Por que Métodos `static`?

O uso de métodos estáticos (`static handle_*`) é uma escolha de design deliberada com dois objetivos principais:



Facilitar a Sobrescrita e Adição

Permite que você, em sua subclasse, sobrescreva ou adicione novos handlers genéricos de forma limpa e direta, sem alterar a lógica de `init()`.



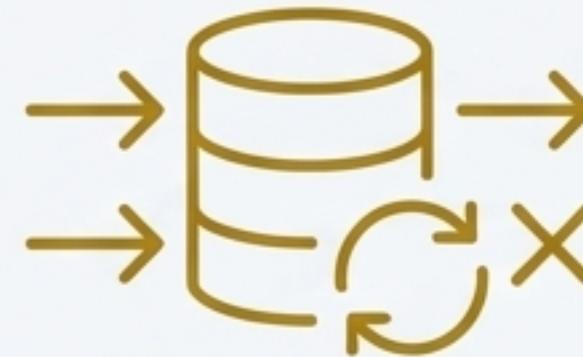
Evitar Conflitos

Garante que os nomes dos handlers genéricos (`handle_validations`, etc.) não entrem em conflito com métodos de *instância* que você possa criar em suas próprias classes de serviço.



O Kit de Ferramentas: O Núcleo de Operações e Segurança

Estes **handlers** formam a base de qualquer serviço, garantindo que os dados possam ser lidos/escritos e que as regras de acesso e concorrência sejam respeitadas.



``handle_crud()``

Adiciona handlers para todas as operações CRUD, incluindo 'deep' create e update. É o alicerce do serviço.



``handle_authorization()``

Implementa as checagens iniciais de autorização, conforme as anotações `@requires` e `@restrict`.



``handle_etags()``

Adiciona controle de concorrência otimista através de ETags, prevenindo 'lost updates'.

O Kit de Ferramentas: Integridade e Ciclo de Vida dos Dados

Garantem que os dados que entram no sistema são válidos e que campos de auditoria e versionamento são gerenciados automaticamente.



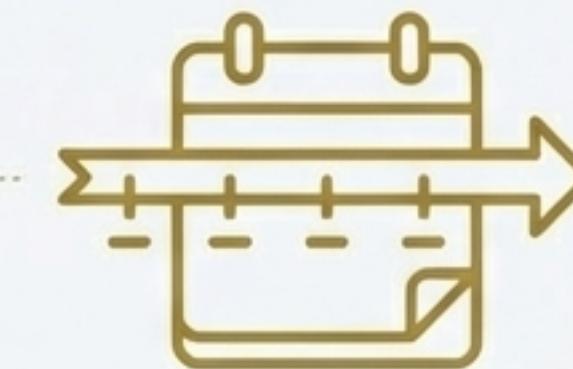
``handle_validations()``

Registra handlers para validação de entrada com base em anotações como `@assert.notNull` e tipos de dados.



``handle_managed_data()``

Gerencia automaticamente campos de auditoria como `createdAt`, `createdBy`, `modifiedAt` e `modifiedBy`.



``handle_temporal_data()``

Adiciona suporte para dados temporais, permitindo consultas 'as of' em um ponto específico do tempo.

O Kit de Ferramentas: Enriquecimento e Experiência do Usuário

Camadas de funcionalidades que traduzem dados brutos em informações prontas para o consumo e habilitam padrões de UI modernos.



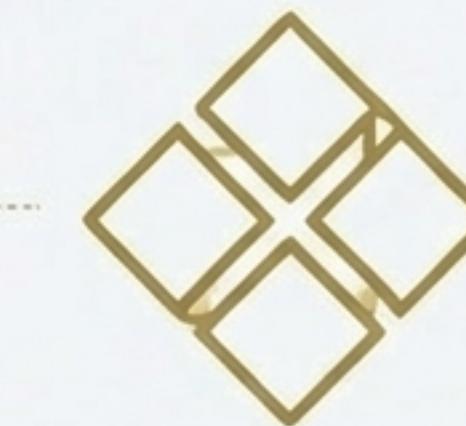
``handle_localized_data()``

Habilita o tratamento automático de dados localizados, servindo textos no idioma da requisição.



``handle_paging()``

Implementa a paginação (`\$top`, `\$skip`) e ordenação implícita para garantir respostas consistentes e performáticas.



``handle_fiori()``

Adiciona suporte a funcionalidades específicas do Fiori, como o rascunho (Draft) e a renderização de 'value helps'.

Referência Rápida: Os 9 Handlers Genéricos Padrão

`handle_crud()` Handlers para todas as operações CRUD.	`handle_fiori()` Suporte a Fiori Draft e outros.
`handle_authorization()` Checagens de autorização.	`handle_paging()` Paginação e ordenação implícita.
`handle_etags()` Controle de concorrência com ETags.	`handle_localized_data()` Tratamento de dados localizados.
`handle_validations()` Validação de entradas (@assert).	`handle_temporal_data()` Suporte a dados temporais.
`handle_managed_data()` Campos de auditoria (`createdAt`, `createdBy`...).	

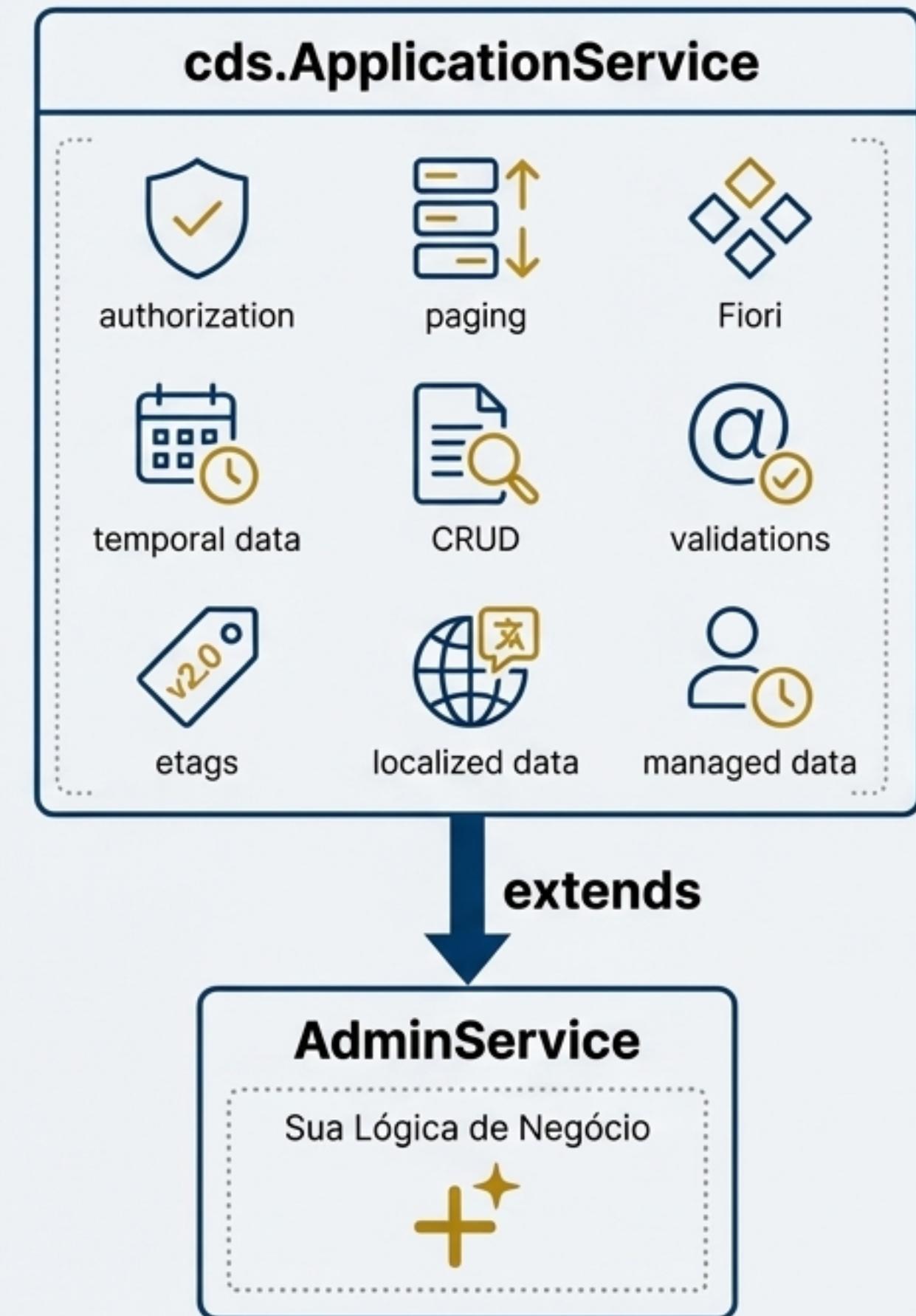
O Ponto de Partida: Criando Seu Serviço Personalizado

A forma mais comum de implementar um serviço é estender `cds.ApplicationService`. Isso garante que você herde todos os handlers genéricos, podendo então adicionar sua lógica de negócio específica.

```
// srv/admin-service.js
const cds = require('@sap/cds')

// Sua classe herda todo o poder de cds.ApplicationService
module.exports = class AdminService extends cds.ApplicationService {
  init() {
    // 'this.on', 'this.before', 'this.after' para sua lógica...
    // Ex: this.after('READ', 'Books', each => { ... });

    // É crucial chamar super.init() para registrar os handlers genéricos
    return super.init() ← Crucial!
  }
}
```



Assumindo o Controle: Sobrescrevendo um Handler Genérico

Para alterar ou substituir um comportamento padrão, basta reimplementar o método `static handle_*` correspondente em sua classe.

Exemplo Prático: Vamos substituir a validação padrão por uma lógica customizada.

Sobrescrita
Direta

```
class YourService extends cds.ApplicationService {  
    static handle_validations() {  
        // NOTA: A lógica padrão de @assert NÃO será registrada  
        // se você não chamar super.handle_validations()  
  
        // 'this' aqui é a instância do serviço  
        this.on('CREATE', '*', req => {  
            console.log('Executando minha validação customizada!')  
            // ... sua lógica de validação aqui  
        })  
  
        // Você pode opcionalmente chamar o handler original  
        // return super.handle_validations()  
    }  
    init() {  
        // ...  
        return super.init()  
    }  
}
```

Sua Lógica
Nova

Ajuste Fino: Desabilitando Funcionalidades Padrão

E se você não quiser uma funcionalidade genérica? Por exemplo, desabilitar o gerenciamento automático de ETags? Simplesmente sobrescreva o método e não faça nada.

```
class MyServiceWithoutETags extends cds.ApplicationService {  
    // Sobrescrevemos o método para que ele não faça nada.  
    // O handler de ETags nunca será registrado para este serviço.  
    static handle_etags() {  
        // Não chame super.handle_etags()  
        // Não registre nenhum handler aqui  
    }  
  
    init() {  
        // ...  
        return super.init()  
    }  
}
```

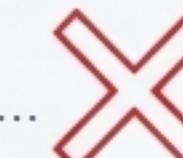
Antes

HTTP/1.1 200 OK
Content-Type: application/json
ETag: W/"..."



Depois

HTTP/1.1 200 OK
Content-Type: application/json



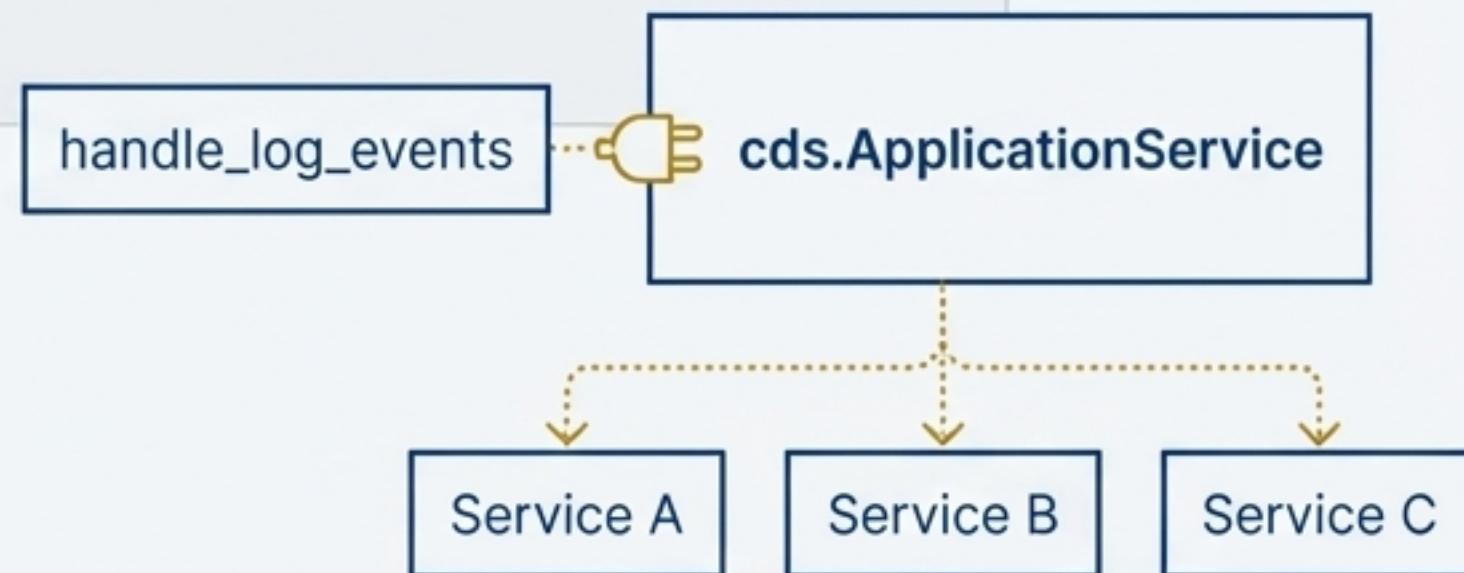
Indo Além: Adicionando Seus Próprios Handlers Genéricos

Você pode estender o próprio `cds.ApplicationService` para adicionar novas funcionalidades genéricas a ***todos*** os serviços que herdam dele. Basta adicionar um novo método de classe com o prefixo `handle_`.

```
const cds = require('@sap/cds')

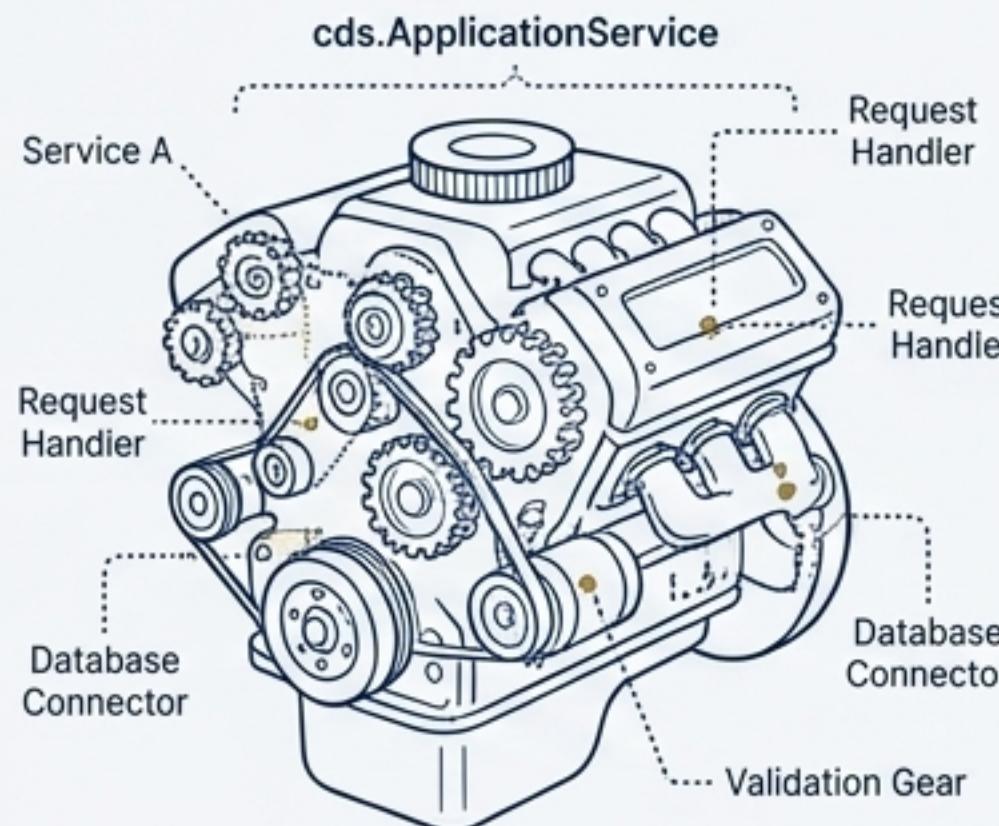
// Adicionamos um novo handler diretamente na classe base
cds.ApplicationService.handle_log_events = cds.service.impl(function() { ..... Extensão Global
    // Este handler será executado para TODOS os serviços
    this.on('*', req => {
        console.log(`Evento recebido: ${req.event} na entidade ${req.target.name}`)
    })
})
```

Coloque este código em um arquivo central (ex: `srv/server.js`)
antes da inicialização dos serviços.



Sua Estratégia de Controle: Um Resumo Visual

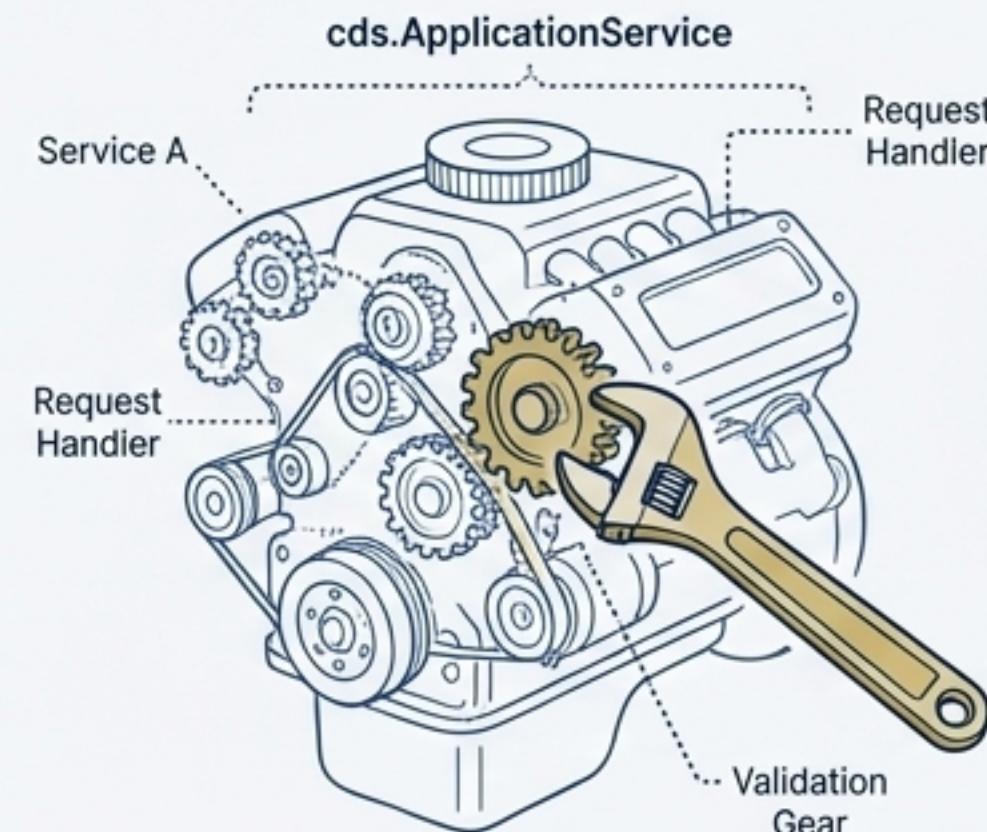
Usar o Padrão



```
class MyService extends cds.ApplicationService
```

Herda todo o kit de ferramentas padrão.
Ideal para a maioria dos casos.

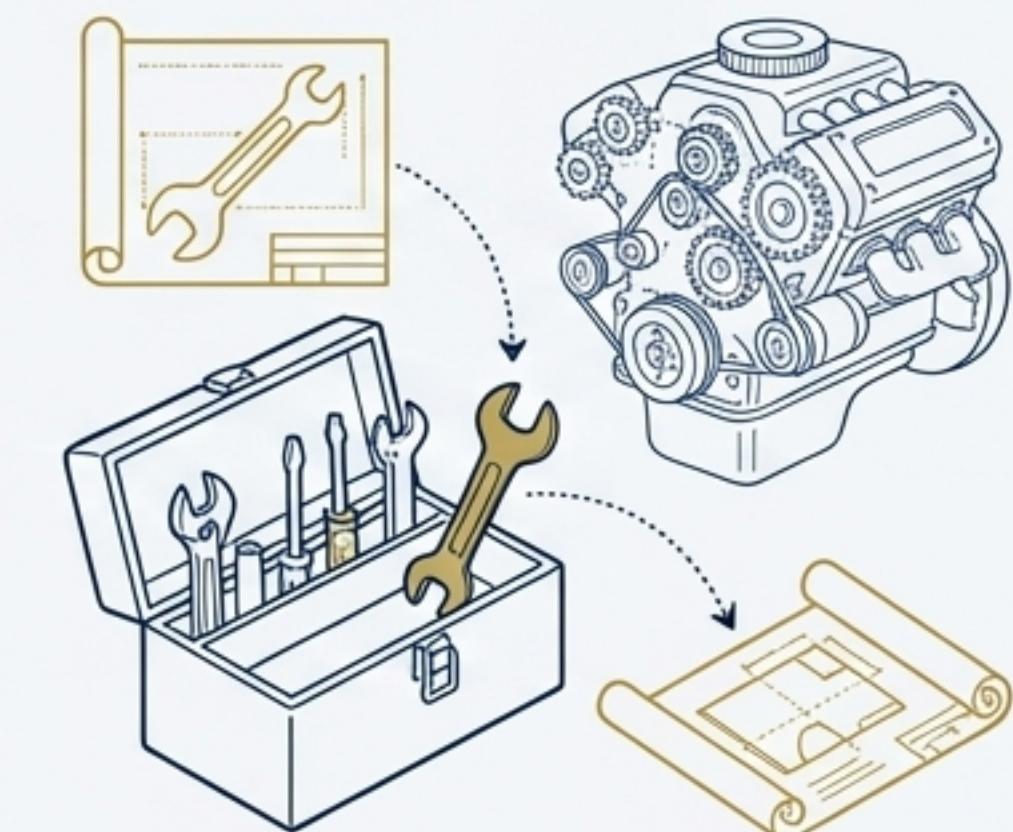
Ajustar uma Ferramenta



```
static handle_validations() { ... }
```

Modifica ou substitui uma funcionalidade
específica para um serviço.

Adicionar uma Nova Ferramenta

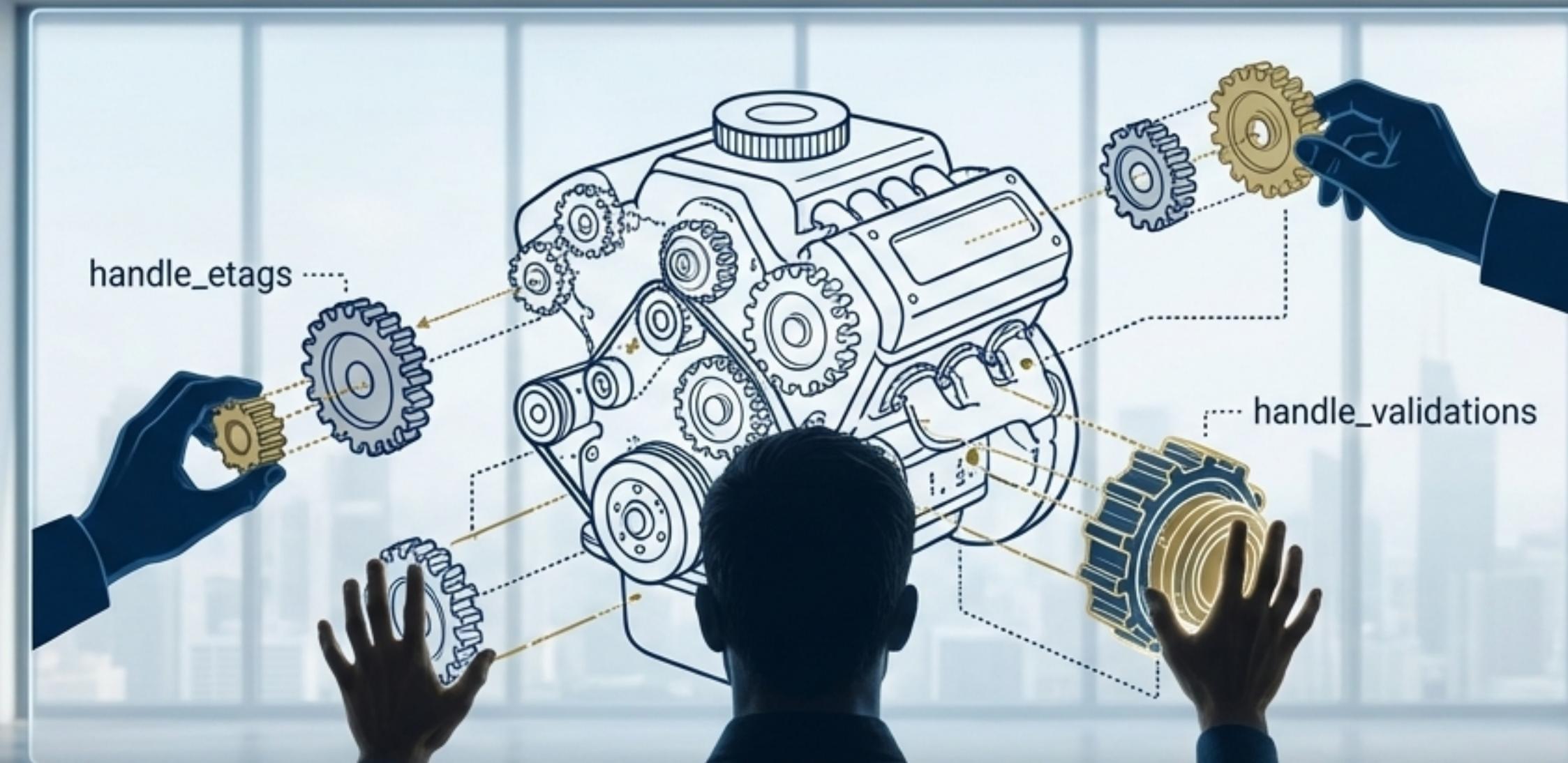


```
cds.ApplicationService.handle_custom = ...
```

Adiciona uma nova funcionalidade genérica
para todos os serviços da aplicação.

De Mágico a Metódico. O Poder em Suas Mão.

A 'mágica' do SAP CAP não é uma caixa preta. É um sistema de engenharia elegante, composto por ferramentas modulares e extensíveis.



Ao entender o `cds.ApplicationService` e seus handlers genéricos, você deixa de ser apenas um usuário do framework para se tornar seu arquiteto. Use esse conhecimento para construir serviços mais robustos, eficientes e perfeitamente adaptados às suas necessidades.