Dunnington Software Development Incorporated

Moving At The Pace Of Integrity!

Briefing Pega Technology: Análise de Regras, Classes e Estrutura de Aplicação

Sumário Executivo

Este documento sintetiza os conceitos centrais da tecnologia Pega, conforme detalhado na sessão de treinamento conduzida por Darryl Dunnington. O núcleo da plataforma Pega é a regra, o bloco de construção fundamental que é traduzido em código Java executável. Tudo em uma aplicação Pega emana de uma regra, existindo mais de 200 tipos de regras para construir desde interfaces de usuário até integrações de banco de dados.

As regras são organizadas dentro de **classes**, que funcionam como agrupadores de instâncias de regras e definem seu escopo dentro da hierarquia da aplicação. Essa hierarquia é estruturada em camadas (Organização, Divisão, Aplicação), e a reutilização e especialização são governadas por mecanismos de herança de padrão e direcionada.

Para implantação e gerenciamento de versões, as classes e suas regras são contidas em **rulesets**. Esses contêineres para implantação permitem o movimento de funcionalidades entre ambientes (desenvolvimento, teste, produção) e suportam práticas de desenvolvimento em equipe, como check-out/check-in, edição privada e branching.

O motor da Pega utiliza um sofisticado algoritmo de **resolução de regras** de 10 passos para determinar qual versão de uma regra executar em um dado contexto, considerando fatores como a pilha de rulesets do usuário, herança de classe, disponibilidade da regra e *circumstancing* (criação de variantes de regras para casos específicos). A execução de uma aplicação envolve o carregamento desses componentes na memória, onde o objeto do caso é mantido na página pyWorkPage do clipboard e seus dados são persistidos no banco de dados em um BLOB conhecido como pzPvStream.

1. O Conceito Fundamental: Regras em Pega

As regras são a essência da tecnologia Pega. Elas são descritas como o "núcleo granular" da plataforma, representando as instruções e os passos individuais que são executados. A plataforma Pega de baixo código traduz o design da aplicação, encapsulado em regras, em código Java para execução.

1.1. Definição e Propósito

- **Unidade Executável:** Uma regra é o componente que é traduzido e se torna código executável. A Pega não conhece outra linguagem senão a de uma regra.
- **Abrangência:** Existem aproximadamente 209 tipos de regras, organizadas em 16 categorias, que cobrem todas as funcionalidades necessárias para construir uma aplicação, incluindo:
    - Interfaces de usuário (UIs)
    - Cálculos
    - Conexões com bancos de dados (ex: regra do tipo Database)
    - Envio de e-mails
- **Fundamento:** "Tudo é uma regra. Tudo emana de uma regra."

1.2. O Processo de Criação de Regras em Quatro Etapas

A utilização de uma regra segue um processo estruturado de quatro passos:

1. **Identificar:** Selecionar o tipo de regra apropriado dentre os mais de 200 disponíveis para realizar a tarefa desejada.
2. **Criar:** Gerar uma nova instância a partir do tipo de regra (o "blueprint"). Isso é feito clicando com o botão direito e selecionando "Create", o que abre um formulário de regra.
3. **Configurar:** Preencher o formulário da regra com informações essenciais:
    a. **Label/Name:** Um nome ou rótulo para a instância da regra.
    b. **Applied to Class:** A classe à qual a regra se aplica, definindo seu escopo.
    c. **Ruleset:** O ruleset onde a regra será armazenada para implantação, incluindo sua versão (major, minor).
4. **Salvar e Implementar:** Salvar a regra, que agora está associada a uma classe dentro da hierarquia da aplicação e pronta para ser implementada no fluxo de trabalho.

1.3. Métodos de Criação: Studio vs. Manual

- **Criação no Studio (Contextual):** Ao criar uma regra a partir do Case Designer, o studio já conhece o contexto (o tipo de caso/classe). Ele seleciona automaticamente o tipo de regra correto, abre o formulário apropriado e implementa a regra na localização específica do fluxo de trabalho.
- **Criação Manual:** Uma regra criada manualmente pode ser implementada em qualquer lugar dentro do fluxo de trabalho ou da aplicação, oferecendo maior flexibilidade.

2. A Estrutura Organizacional: Classes e Hierarquia

As classes são um conceito fundamental para agrupar e dar escopo às regras. Elas formam a espinha dorsal da estrutura de uma aplicação Pega.

2.1. Definição de Classe: Agrupamento de Regras

- **Definição Chave:** Uma classe é um "agrupamento de instâncias de regras". As regras são "aplicadas a" uma classe, um termo que em Pega é análogo a uma relação "tem um" (has-a) em programação orientada a objetos tradicional.
- **Classe como Regra:** A própria classe é um tipo de regra, encontrada na categoria SysAdmin, o que permite a criação de diferentes tipos de classes.

2.2. Tipos de Classes e Seu Escopo

Existem três tipos principais de classes, cada uma com regras de armazenamento específicas dentro da estrutura de camadas da aplicação (Enterprise Class Structure - ECS):

Tipo de Classe	Descrição	Camadas Permitidas para Armazenamento
Work-	Representa tipos de caso (case types).	Apenas na camada de Aplicação (Implementation e Framework).
Data-	Representa objetos de dados.	Pode ser armazenada em todas as camadas (Organization, Division, Application).

Integration-	Usada para regras relacionadas a integrações.	Pode ser armazenada em todas as camadas (Organization, Division, Application).

A hierarquia e a localização de uma classe podem ser visualizadas no **App Explorer** do Dev Studio ou através da tabela de herança ("Inheritance table").

2.3. Herança de Padrão e Direcionada

A Pega utiliza dois mecanismos de herança para encontrar e reutilizar regras, formando a base para o processo de resolução de regras.

- **Herança de Padrão (Pattern Inheritance):**
  - Baseada na estrutura de nomes das classes.
  - Para encontrar a classe pai, o sistema remove o sufixo final do nome da classe (a parte após o último hífen). O prefixo restante é o nome da classe pai.
  - A busca por uma regra começa na classe atual e sobe hierarquicamente através dos pais de padrão.
- **Herança Direcionada (Directed Inheritance):**
  - Após a busca por herança de padrão, o sistema continua a busca em classes explicitamente definidas, como Work-Cover-, Work- e, finalmente, as classes base da Pega (@baseclass).
  - O processo de busca sempre executa a herança de padrão primeiro, seguida pela herança direcionada.

3. Contêineres para Implantação: Rulesets

Rulesets são o mecanismo pelo qual as regras e classes são gerenciadas, versionadas e movidas entre diferentes ambientes.

3.1. Definição e Função

- **Definição Chave:** Um ruleset é um "contêiner para implantação".
- **Propósito:** Agrupa um conjunto de regras (contidas em classes) para facilitar a implantação de uma versão de um aplicativo de um ambiente para outro.

- **Ruleset Stack:** Uma lista ordenada de rulesets que compõem a aplicação de um usuário. É carregada dinamicamente no login e determina a ordem de busca de regras. O ruleset pessoal do desenvolvedor (ex: Author@MyTown) fica no topo da pilha.

3.2. Gerenciamento de Versões e Ciclo de Vida

- **Check-out/Check-in:** Para evitar conflitos em um ambiente de equipe, um desenvolvedor faz o "check-out" de uma regra. Uma cópia da regra é movida para seu ruleset pessoal. Como este está no topo da pilha, suas alterações são isoladas do resto da equipe até que ele faça o "check-in" da regra de volta ao ruleset principal.
- **Locking:** No final de um ciclo de desenvolvimento (sprint), um ruleset é tipicamente "travado" (locked) para evitar alterações. Um novo ruleset, com uma versão incrementada (ex: 01-01-02 após 01-01-01), é criado e desbloqueado para o próximo ciclo de trabalho.
- **Private Edit:** Permite que um desenvolvedor faça o "check-out" de uma regra que está em um ruleset travado. A alteração vai para o ruleset pessoal, mas não pode ser devolvida (check-in) ao ruleset travado. No entanto, ela pode ser salva no novo ruleset desbloqueado, efetivamente substituindo a versão mais antiga.
- **Skimming:** Um processo que cria uma nova versão de um ruleset contendo apenas a versão mais alta de cada regra de rulesets anteriores. É usado para otimizar e consolidar o número de rulesets em produção.

3.3. Tipos Especializados de Rulesets

- **Production Ruleset:** O único tipo de ruleset que pode permanecer desbloqueado em um ambiente de produção. É usado para delegar a manutenção de certas regras (ex: uma tabela de decisão com taxas de juros) a usuários de negócio.
- **Branching:** Utilizado para desenvolvimento paralelo. Equipes separadas (ex: equipe de UI, equipe de backend) trabalham em "branches" (ramos) isolados de regras. Quando o trabalho é concluído, o branch é mesclado de volta ao "tronco" principal de desenvolvimento.

4. O Algoritmo de Decisão: Resolução de Regras

A Pega emprega um algoritmo robusto para determinar qual regra executar quando múltiplas versões podem existir.

4.1. O Processo de 10 Etapas

O algoritmo de resolução de regras segue uma sequência de passos para filtrar e selecionar a instância de regra mais apropriada:

1. Verificar se a regra já está no cache.
2. Se não, encontrar todas as instâncias que correspondem ao nome e tipo da regra.
3. Descartar regras marcadas como Not Available.
4. Descartar regras de rulesets que não estão na pilha de rulesets do usuário.
5. Descartar regras em classes que não fazem parte da hierarquia da classe atual.
6. Classificar as regras restantes com base em circumstancing, classe, ruleset e versão.
7. Ordenar as regras com base na classificação.
8. Encontrar a instância mais apropriada (geralmente a primeira da lista ordenada).
9. Verificar se a regra finalista está marcada como Blocked.
10. Verificar as autorizações do usuário e carregar a regra.

4.2. Status de Disponibilidade de Regras

Cada regra possui um status de disponibilidade que impacta diretamente sua participação na resolução de regras.

Status	Descrição
Available	O estado padrão. A regra participa do processo de resolução. A versão mais recente no ruleset mais novo substitui as versões mais antigas.
Not Available	A regra é ignorada durante o processo de resolução.

Final	A regra pode ser executada, mas não pode ser editada ou copiada para outro ruleset. Usado para proteger regras centrais da Pega.
Blocked	Se esta regra for selecionada, o sistema gera um erro. **Caso de uso:** Desativar rapidamente uma funcionalidade com defeito em produção, criando uma nova versão bloqueada que exibe uma mensagem de "indisponível".
Withdrawn	Faz com que o algoritmo ignore esta instância e todas as instâncias em classes inferiores ou versões de ruleset mais antigas. **Caso de uso:** Desativar políticas ou funcionalidades governamentais antigas.

5. Conceitos Avançados de Regras

5.1. Regras Relevantes (Relevant Rules)

Uma regra pode ser marcada como "relevante", o que significa que ela aparecerá em listas suspensas contextuais dentro do App Studio. Isso facilita a seleção pelo desenvolvedor. Regras não marcadas como relevantes não aparecem nessas listas, embora ainda possam ser usadas. É essencialmente uma configuração de visibilidade para o tempo de design.

5.2. Circumstancing: Criando Variações de Regras

Circumstancing é a versão da Pega para os conceitos de sobrecarga e sobrescrita (overloading/overriding). Permite a criação de uma **regra variante** que é executada em vez da **regra base** sob circunstâncias específicas. Isso permite especializar o comportamento de uma regra com base em:

- **Propriedade:** O valor de uma propriedade (ex: se CustomerTier = "Gold").
- **Data:** Um intervalo de datas.
- **Multivariada:** Uma combinação de múltiplas propriedades.

6. A Arquitetura Pega em Execução

A execução de uma aplicação Pega segue um fluxo lógico desde o login do usuário até a persistência dos dados.

6.1. Do Login à Persistência de Dados

1. **Login:** Um usuário entra com um Operator ID, que está associado a um Access Group.
2. **Carregamento:** O Access Group aponta para uma Application Rule, que carrega a pilha de rulesets necessária para a aplicação.
3. **Execução:** O usuário executa um tipo de caso (Work- class). Um objeto de caso é criado na memória.
4. **Clipboard:** O objeto de caso em memória pode ser inspecionado na ferramenta **Clipboard** dentro de uma página de alto nível chamada pyWorkPage.
5. **Persistência:** À medida que o caso avança, seus dados são salvos automaticamente no banco de dados em uma única coluna do tipo BLOB (Binary Large Object) chamada **pzPvStream**. Cada instância de caso recebe uma chave primária única, a **pzInsKey**.

6.2. Estrutura de Classes da Empresa (Enterprise Class Structure - ECS)

A ECS organiza as classes e rulesets em camadas para promover a reutilização e a especialização em toda a empresa.

- **Camadas:** Organization, Division, Application (Implementation/Framework), e a camada Base Pega.
- **Estrutura de Ruleset:** Por padrão, cada camada na ECS possui dois rulesets. Um (ex: R1) armazena a maioria das regras, enquanto o outro (R2) é dedicado a regras de classes de integração (Integration-).

7. Recursos do Curso e Tarefas Administrativas

7.1. Materiais de Suporte Disponíveis

Os participantes têm acesso a uma variedade de recursos para apoiar seu aprendizado:

- Gravações das sessões (manhã e noite).
- Notas de reunião (em inglês e espanhol).
- Documento de resumo consolidado.
- Podcast (em inglês e espanhol).
- Mapas mentais (Mind Maps).
- Canal no YouTube com 26 vídeos das sessões.

7.2. Ações Requeridas dos Participantes

- **Certificado BAE:** Verificar o e-mail para o recebimento do "Certificate of Achievement" da certificação BAE.
- **Upload de Pontuação:** Fazer o upload dos resultados da certificação BAE na pasta 1113.
- **Ensaios SAPE:** Concluir todos os ensaios para estar preparado e aprovado para a certificação SAPE.
- **Distintivos Digitais (Badges):** Trabalhar para obter os distintivos digitais, incluindo o "System Architect", para um total de cinco badges. O treinamento futuro da AWS resultará em quatro certificados adicionais.