Gestão de Configuração e Mudanças de Software

Allan Lima

Arquitetura, Design e Implementação de Sistemas para Internet
Pós Graduação
Faculdade 7 de Setembro

arglbr@gmail.com

nov/2014

Introdução ao Maven

Parte 1

O que é Maven?

- Uma tentativa de definir Maven:
  - Ferramenta para simplificar o processo de build.
- `Build`?
  - Tarefas rotineiras que levam a construção ou montagem de um software a partir do código fonte.
- Tarefas rotineiras?
  - Compilar, executar testes, empacotar aplicação, etc.
- Outra tentativa de definir Maven:
  - Um conjunto de padrões usados para gerenciar e descrever projetos em java.

Quais os benefícios do Maven?

- Modelo que pode ser aplicado aos projetos Java.
- A idéia é que o modelo traga mais transparência, mais reuso, mais facilidade de manutenção e entendimento.
- Fornece uma abstração que a maioria dos desenvolvedores estão familiarizados:
  - Semelhante a abstração do automóvel: se você aprendeu a dirigir em um modelo A de carro, então poderá facilmente dirigir um modelo B.

Quais os benefícios do Maven?

- Abordagem declarativa:
  - POM — Project Object Model.
  - A tarefas são delegadas para o POM e para os plugins.
  - Os desenvolvedores podem usar as tarefas (encapsuladas pelos plugins) sem necessariamente entender como elas funcionam internamente

Princípios do Maven

- Convenções sobre a configuração
- Reuso de lógicas de builds
- Execução declarativa
- Organização coerente de dependências

Princípio 1: Convenção sobre a configuração

- Estratégia de "propriedades *defaults*" para a maioria das tarefas (podem ser alteradas quando conveniente) economiza tempo.
- Convenções primárias:
  - Estrutura de diretórios padrão para projetos
    - Código fonte, recursos (xml, properties), saída de arquivos gerados, documentação etc
  - Cada projeto gera um único resultado: `jar`, `war`, `ear`...
  - Padrões de nomes
    - Para diretórios: `my-app/src/main/java`
    - Para arquivos gerados (outputs): `commons-logging-1.2.jar`

Princípio 2: Reuso de lógicas de *builds*

- Toda a lógica de *build* é encapsulada pelos *plugins*
- Um *plugin* para
  - Para compilar o código fonte
  - Para executar os testes de unidade
  - Para empacotar a aplicação (`jar`, `war`, `ear`)
  - Para gerar javadocs
  - etc

Princípio 3: Execução declarativa

- Tudo no maven é orientado de forma declarativa no `POM` e nas configurações específicas dos *plugins*. Exemplo do `POM`:

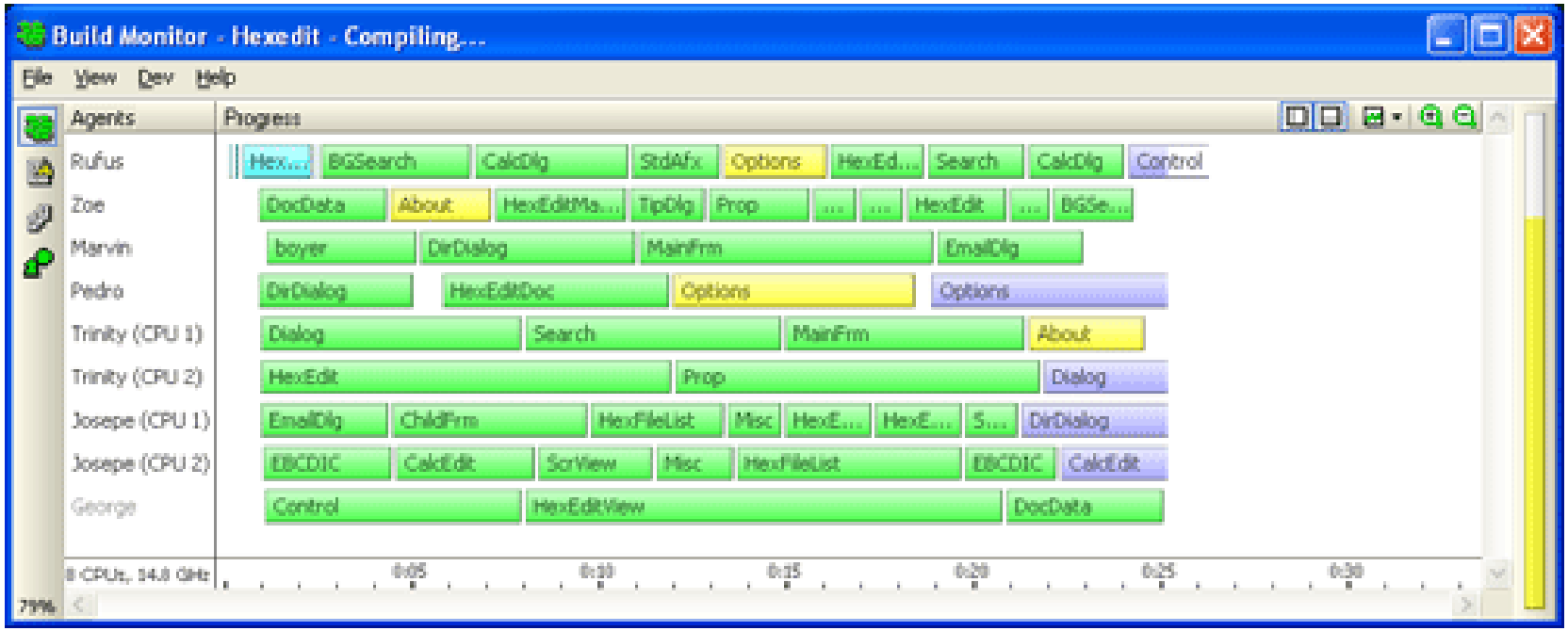
```xml
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.curso</groupId>
    <artifactId>dv-generator</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>DV Generador</name>
    <description>Gerador de digito verificador</description>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Automação Avançada

- Nem todas as ferramentas de automação podem executar build distribuída
  - A maioria apenas fornece suporte ao processamento distribuído
  - A maioria das soluções dão suporte apenas a C/C++.
- Um exemplo de solução de build distribuída é o IncrediBuild Xoreax para a plataforma Microsoft Visual Studio

Automação Avançada

- IncrediBuild Xoreax:



Vantagens

- Melhora a qualidade do produto
  - Acelerar o processo de compilação e link
  - Eliminar tarefas redundantes
  - Minimiza as *bad build*
  - Elimina a dependência de pessoa-chave
  - Possui histórico de versões e *releases* para investigar problemas
  - Poupa tempo e dinheiro (por causa das razões listadas acima)

Vantagens

- **On-Demand:** um usuário executa um script na linha de comando
- **Automação agendadas:** um servidor de integração contínua executa uma `nightly build`
- **Automação por evento:** um servidor de integração contínua executando uma build a cada commit de um sistema de controle de versão

Script de Build

- Uma forma específica de automação de build é a criação de scripts de build (Makefiles). Isto é conseguido através de ferramentas como:
  - GNU Automake
  - Cmake
  - Imake
  - Qmake
  - Apache Ant
  - Apache Maven
  - OpenMake Meister

Script de Build

- Requisitos **básicos** de um sistema de build
  - Processo de compilação incremental.
  - Build frequente durante a noite para detectar os problemas mais cedo.
  - Suporte à gerenciamento de dependência de código fonte.
  - Relatar que arquivos fonte deram origem à um determinado executável.
  - Construção rápida.
  - Relatórios sobre a construção, compilação e link.

Script de Build

- Requisitos **opcionais** de um sistema de build
  - Gerar notas de lançamento e outros documentos, como páginas de ajuda.
  - Construir relatórios de status.
  - Relatórios de aprovação ou reprovação de teste.
  - Resumo dos recursos adicionados / modificados / excluídos a cada nova compilação.