SUMMA TECHNOLOGIES

Desenvolvendo aplicações desktop em Java: presente e futuro

Michael Nascimento Santos

Conexão Java 2006

Michael Nascimento Santos

- 7 anos de experiência com Java
- Expert nas JSRs 207, 250, 270 (Java 6), 296 (Swing Application Framework) e 303 (Bean Validation)
- Co-fundador do SouJava
- Fundador do genesis (https://genesis.dev.java.net) e do ThinNB (https://thinnb.dev.java.net)
- Palestrante no JavaOne, JustJava, Abaporu, FISL, COMDEX, BrasilOne e Conexão Java

Agenda

- Desktop em Java ?
- Tecnologias de UI (Swing x SWT x Thinlet)
- Produtividade no design
- Programando a lógica de interface
- Distribuindo a aplicação
- Integrando com o backend
- Demonstração prática
- O futuro

Desktop em Java?

- Muito se fala de Java na web, no servidor, no backend...

  E o desktop, cadê?

- Havia razões para isso:

  Problemas de performance

  Dificuldade de distribuição

  Dificuldade de desenho das telas

  Dificuldade de programação

Desktop em Java?

- Problemas de performance resolvidos nas versões mais novas
- Distribuição resolvida com Java WebStart
- Desenho e programação resolvidas mais recentemente
- Exemplos famosos:

  Azureus

  Imposto de Renda multi-plataforma

  Eclipse, IntelliJ IDEA, NetBeans

Tecnologias de UI

- Existem algumas opções de APIs para desenvolvimento desktop. Principais:

  Swing

  SWT

  Thinlet

- Cada uma tem as suas características. É preciso conhecer, comparar e analisar para escolher corretamente

Swing

- Toolkit gráfico presente no JDK
- Padrão, portado para todas as plataformas
- Culpado pela má fama da lentidão do Java no desktop por ser “emulado”

  Problemas de performance foram superados
- Criticado por não ter aparência “nativa”

  JGoodies Looks

  Melhoras significativas, especialmente no Java 6

Swing

- Exemplos de aplicações:
  - ***Imposto de renda multiplataforma***
  - ***NetBeans***
  - ***IntelliJ IDEA***
  - ***Ferramentas da Oracle***

SWT

- Criado com o objetivo de usar os componentes gráficos (widgets) de cada sistema operacional, bem como sua aparência
- Performance muito boa no Windows, varia muito em outras plataformas
- Modelo de programação “estranho”
- Desenvolvimento mantido pela Eclipse Foundation
- Exemplos:

  Eclipse & WSAD

  Azureus

Thinlet

- Criado com o objetivo de ser leve, abrangente e rodar em JVMs velhas e/ou limitadas (browsers e PDAs)
- Bastante pequeno (39 kb)
- Muito fácil de desenhar telas (xml)
- Muuuuuito difícil de estender
- Consome realmente poucos recursos
- Exemplo:

  ThinFeeder

Quando usar...

- Thinlet

  Cliente leve, limitado graficamente

- Swing x SWT

  Mais uma questão de gosto do que uma resposta direta...

  Swing possui performance homogênea e é suportado pela Sun

  SWT tende a ser mais rápido no Windows e é ligado aos componentes nativos

  Eu prefiro Swing :-D

Produtividade no design

- Antes, bem, não havia :-)
- Tudo feito na mão:
  - ***Muito tempo perdido em design***
  - ***Qualidade do código dependia do desenvolvedor***
  - ***Maior flexibilidade***
- Bem-vindo a era das ferramentas de design:
  - ***ThinNB***
  - ***Matisse (NetBeans)***
  - ***Visual Editor Project (Eclipse)***

Design com Thinlet

- Formato XML feito para ser simples de editar
- Existe editor standalone: ThinG (http://thing.sf.net/)
- Suporte dentro do NetBeans adicionado pelo ThinNB (https://thinnb.dev.java.net/)

  Reconhece os arquivos xml

  Permite preview dentro da IDE

  Integra o ThinG ao NetBeans

Matisse x VEP

- Matisse:
  - ***Mais intuitivo e completo por enquanto***
  - ***Não permite editar o código***
- Visual Editor Project:
  - ***Suporta SWT também***
  - ***"Interpreta" o código escrito, mas quando se perde, não há o que fazer...***
- "Mas eu não gosto do código que a ferramenta gera..."
  - ***E você olhava os forms do VB e do Delphi?***

Programando a lógica de interface

- Estudar a API gráfica e lidar com listeners, models, jeitos diferentes de fazer as mesmas coisas...

  NÃO!!!!

  Você programa pra web com servlets e JSPs nus e crus?

- Use uma solução de alto nível; use um framework de binding

  Ligam sua classe à interface gráfica de forma padronizada

  Só se preocupe com a lógica de interface

JGoodies Binding

- Permite fazer binding de beans a componentes gráficos
- Modelo “explícito” de programação: você define manualmente quais propriedades ligar aos componentes e como ligar
- Limitações

  Requer definição de JavaBean clássico com PropertyChangeListener e eventos disparados “na mão”

  Exige que o model dos componentes seja explicitamente declarado

genesis - https://genesis.dev.java.net/

- Permite fazer binding de qualquer JavaBean
- Assume muitas coisas com base nos componentes
- Permite trabalhar com componentes existentes (e seus models) e acrescentar outros
- Possui conceitos/abstrações de mais alto nível, como ações, habilitação /desabilitação de componentes, controle de visibilidade, chamada condicional de métodos, erros etc.
- Suporta Swing, SWT e Thinlet
- Possui documentação em português

Distribuindo a aplicação

- Como instalar e atualizar a aplicação cliente:

  Vou ter que ir instalar na máquina de cada usuário?

  E se quiser fazer upgrade da JVM? Ou usar uma versão específica?

  Não tenho como distribuir a aplicação; é muito grande pra banda que eu tenho

- Há solução...

Java WebStart

- Permite distribuir as atualizações da aplicação de forma simples
- Usuário clica no ícone e tudo funciona :-)
- Permite distribuir apenas as classes modificadas através do versionamento dos jars da aplicação
- Permite distribuir uma versão específica de JVM, se necessário
- Resolve maior limitação na adoção da tecnologia desktop

Como integrar com o backend?

- Muitas alternativas:
  - ***EJB***
  - ***WebServices***
  - ***XML/RPC***
  - ***JBoss Remoting***
  - ***Spring Remoting***
  - ***Rodar em modo local!***
- De preferência, escolha uma tecnologia que permita trocar a comunicação com o servidor sem reescrever o código inteiro

Demonstração prática

- Tecnologias:
  - ***Java 5***
  - ***Swing (Thinlet, se der)***
  - ***Matisse (NetBeans)***
  - ***genesis (binding e integração com backend)***
  - ***JBoss (servidor)***
  - ***Java WebStart***

SUMMA TECHNOLOGIES

Demo

O futuro

- Três grandes esforços de padronização no JCP:
  - ***JSR-295: Beans Binding***
  - ***JSR-296: Swing Application Framework***
  - ***JSR-303: Bean Validation***
- Primeiras JSRs de padronização “alto nível” para tecnologias desktop
- Provavelmente serão integrados no Java 7, se forem concluídas a tempo e o Expert Group da JSR do Java 7 aprovar

JSR-295: Beans Binding

- Padroniza ligação entre JavaBeans
  - ***Observe que não é entre JavaBeans e interfaces gráficas***
  - ***Provavelmente seguirá parte do modelo do JGoodies Binding***
  - ***Status atual desconhecido***

JSR-296: Swing Application Framework

- Padroniza elementos básicos do desenvolvimento Swing
  - ***Infelizmente, o foco é apenas no Swing***
  - ***Gerenciamento do ciclo de vida da aplicação***
  - ***Carregamento de recursos e "branding"***
  - ***Conceito de sessão***
  - ***Suporte a ações (sim, @Action!!!)***

JSR-303: Bean Validation

- Padroniza a definição de validadores, regras de validação, ciclo de validação e afins
- Serve não somente para o desktop, mas também para a web e para a camada de persistência
- Provavelmente não entregará um mecanismo de validação end-to-end, i.e., precisará da implementação de diversos componentes para que funcione

  Intenção é ser usada pelos frameworks, e não substituí-los

genesis 3.x e o futuro

- Versão 3.0 final (provavelmente) entre Dezembro/2006 e Janeiro/2007
- Você pode colaborar:

  Fazendo download

  Fazendo perguntas na lista

  Ajudando a definir os requisitos das novas funcionalidades:

  - **Suporte melhorado a paginação**
  - **Edição e ordenação arbitrária em tabelas**
  - **Modelo assíncrono de ações**

SUMMA
TECHNOLOGIES

Perguntas?

SUMMA TECHNOLOGIES

Obrigado!

https://genesis.dev.java.net

http://blog.michaelnascimento.com.br/

Michael Nascimento Santos

Conexão Java 2006