Tradução da Oitava Edição

Java: A Referência Completa

Sobre o Autor

Herbert Schildt é uma das principais autoridades nas linguagens Java, C++ e C#. Seus livros sobre programação já venderam milhões de cópias no mundo todo e foram traduzidos para os principais idiomas. Ele é autor de diversos livros sobre Java, incluindo: *Java: A Beginner's Guide*, *Herb Schildt's Java Programming Cookbook*, *Swing: A Beginner's Guide* e *A Arte do Java*. Outros *bestsellers* de Herbert são: *C++: The Complete Reference*™, *C#: The Complete Reference*™ e *C: The Complete Reference*™. Embora tenha interesse em todas as áreas da informática, seu foco principal são as linguagens de programação, incluindo compiladores, interpretadores e linguagens de controle de robótica. Ele também tem bastante interesse na padronização de linguagens. Schildt é graduado e pós-graduado pela Universidade de Illinois. Ele pode ser encontrado em seu escritório de consultoria pelo telefone (+1 217 586 4683). O endereço de seu site é **www.HerbSchildt.com**.

Sobre o Editor Técnico

Dr. Danny Coward vem contribuindo com as plataformas Java desde 1997. Ele foi membro fundador do grupo Java EE quando trabalhou na Sun. É membro do Comitê de Processos Executivos da Comunidade Java e contribuiu bastante em todas as edições da Plataforma Java – Java SE, Java ME e Java EE –, além de ter criado a equipe original do Java FX.

Tradução da Oitava Edição

Java: A Referência Completa

Herbert Schildt

ALTA BOOKS
EDITORA
Rio de Janeiro, 2013

Sumário Resumido

Prefácio

O Java é uma das linguagens de programação mais importantes e mais utilizadas no mundo. Além disso, ele carrega este título há muitos anos. Ao contrário de outras linguagens de programação, cuja influência foi desaparecendo com o passar do tempo, a influência do Java ficou cada vez mais forte. O Java saltou à frente da programação para internet em sua primeira versão. E cada versão subsequente só reforçou esta posição. Atualmente, ele ainda é a melhor opção para desenvolver aplicativos baseados na web. O Java também participou da revolução dos *smartphones*, já que é usado para programar o Android. Simplificando: grande parte do mundo moderno roda em código Java. O Java é realmente importante.

Um grande segredo para o sucesso do Java é a sua agilidade. Desde a versão original 1.0, o Java vem se adaptando continuamente às mudanças no ambiente de programação e às mudanças na forma como os desenvolvedores programam. O mais importante é que o Java não apenas seguiu as tendências, mas, também, ajudou a criá-las. A capacidade do Java de comportar a velocidade das mudanças no mundo da informática é uma parte crucial do motivo pelo qual ele foi e continua sendo um sucesso.

Desde a primeira publicação deste livro, em 1996, o conteúdo passou por diversas edições, cada uma delas refletindo a evolução contínua do Java. Esta é a oitava edição, atualizada com base no Java SE 7. Como resultado, este livro contém uma quantidade considerável de material novo. Por exemplo: ele inclui uma abordagem das melhorias da linguagem Project Coin, os recursos expandidos do NIO (NIO 2) e o Framework Fork/Join. Em geral, as discussões sobre os novos recursos foram integradas aos capítulos já existentes, mas, como o NIO sofreu muitos acréscimos, ele passou a ter um capítulo exclusivo. No entanto, a estrutura geral do livro permanece a mesma. Isso significa que, se você está familiarizado com a versão anterior, irá se sentir em casa com esta nova edição.

Um Livro para Todos os Programadores

Este livro é para todos os programadores, sejam eles novatos ou profissionais experientes. O novato encontrará discussões cuidadosamente detalhadas e muitos exemplos úteis. Sua abordagem profunda dos recursos mais avançados do Java, bem como de suas bibliotecas, é mais direcionada aos profissionais. Em ambos os casos, o livro serve de recurso durável e um guia de referência bastante útil.

O Que Você Vai Encontrar Neste Livro

Este livro é um guia abrangente da linguagem Java, descrevendo sua sintaxe, palavras-chave e princípios básicos de programação. Parcelas significativas da biblioteca API Java também são analisadas. Este livro está dividido em quatro partes, cada uma delas se concentrando em um aspecto diferente do ambiente de programação Java.

A Parte I apresenta um tutorial aprofundado da linguagem. Ela começa com o básico, incluindo assuntos como tipos de dados, operadores, instruções de controle e classes. Em seguida, passa a tratar de herança, pacotes, interfaces, manipulação de exceções e *multithreading* (programação em várias linhas de execução). Os capítulos finais da Parte I descrevem anotações, enumerações, autoboxing e tipos genéricos (generics). Também são apresentados I/O e applets.

A Parte II analisa os principais aspectos da biblioteca padrão da API Java. Os assuntos incluem strings, I/O, comunicação, utilitários padrão, o Framework Collections, applets, controles baseados em GUI, imagens e concorrência (incluindo o novo Framework Fork/Join).

A Parte III fala de três importantes tecnologias Java: Java Beans, servlets e Swing.

A Parte IV tem dois capítulos que mostram exemplos do Java em ação. O primeiro capítulo desenvolve diversos applets, que executam vários cálculos financeiros comuns, como computar o pagamento regular de um empréstimo ou o investimento mínimo necessário para sacar um determinado valor mensal. Este capítulo também mostra como converter estes applets em servlets. O segundo capítulo desenvolve um gerenciador de downloads que supervisiona o download de arquivos. Ele inclui a capacidade de iniciar, parar e retomar uma transferência. Ambos os capítulos foram adaptados de meu livro *A Arte do Java*, que teve como coautor James Holmes.

Lembre-se: Código na Web

Lembre-se, o código fonte de todos os exemplos deste livro está disponível no site da Alta Books: **www.altabooks.com.br**.

Agradecimentos Especiais

Gostaria de agradecer especialmente a Patrick Naughton, Joe O'Neil, James Holmes e Danny Coward.

Patrick Naughton foi um dos criadores da linguagem Java. Ele também ajudou a escrever a primeira edição deste livro. Por exemplo: entre muitas outras contribuições, grande parte do material dos Capítulos 19, 21 e 26 foi inicialmente fornecido por Patrick. Suas ideias, experiência e energia contribuíram imensamente para o sucesso da primeira edição.

Durante a preparação da segunda e terceira edições deste livro, Joe O'Neil forneceu os primeiros rascunhos para o material que agora pode ser encontrado nos Capítulos 28, 29, 31 e 32 desta edição. Joe ajudou em vários dos meus livros e suas ideias sempre foram muito boas.

James Holmes contribuiu com o Capítulo 34. James é um programador e autor extraordinário. Ele foi meu coautor em *A Arte do Java* e é autor de *Struts: The Complete Reference*™ e coautor de *JSF: The Complete Reference*™.

Danny Coward é o editor técnico desta edição. Seus conselhos, ideias e sugestões foram de grande valor e muito apreciados.

HERBERT SCHILDT

Para Aprofundar os Estudos

Java: A Referência Completa, Tradução da Oitava Edição, é a sua porta de entrada para a série de livros de programação de Herb Schildt. A seguir, outros títulos que você pode achar interessantes (alguns deles com conteúdo em inglês).

Para saber mais sobre a programação em Java, recomendamos os seguintes livros:

Herb Schildt's Java Programming Cookbook

Java: A Beginner's Guide

Swing: A Beginner's Guide

A Arte do Java

Para saber sobre C++, você achará estes livros bastante úteis:

C++: The Complete Reference

Herb Schildt's C++ Programming Cookbook

C++: Guia Para Iniciantes

The Art of C++

C++ From the Ground Up

STL Programming From the Ground Up

Para saber sobre C#, sugerimos os seguintes livros de Schildt:

C#: The Complete Reference

C#: A Beginner's Guide

Para saber sobre a linguagem C, o título a seguir é bem interessante:

C: The Complete Reference

Quando precisar de respostas concretas e rápidas, conte com Herbert Schildt, uma autoridade reconhecida em programação.

PARTE I

A Linguagem Java

CAPÍTULO 12
Enumerações, Autoboxing e Anotações (Metadados)

CAPÍTULO 13
I/O, Applets e Outros Tópicos

CAPÍTULO 14
Tipos Genéricos (Generics)

CAPÍTULO 1

História e Evolução do Java

Para entender o Java por completo, é preciso entender os motivos por trás de sua criação, as forças que o moldaram e o legado que ele herda. Assim como as linguagens bem-sucedidas que vieram antes, o Java é uma mistura dos melhores elementos de sua rica herança, combinados com os conceitos inovadores exigidos por sua missão exclusiva. Enquanto o restante dos capítulos deste livro descreve os aspectos práticos do Java – incluindo sua sintaxe, bibliotecas e aplicativos importantes –, este capítulo explica como e por que o Java foi criado, o que faz dele tão importante e como ele evoluiu ao longo dos anos.

Embora o Java esteja ligado de maneira inseparável ao ambiente online da internet, é importante lembrar que, antes de mais nada, ele é uma linguagem de programação. A inovação e o desenvolvimento das linguagens ocorrem por dois motivos fundamentais:

- Para se adaptar às mudanças de ambientes e usos.
- Para implementar melhorias e ajustes na arte de programar.

Como você vai ver, o desenvolvimento do Java foi orientado por estes dois elementos quase que em proporções iguais.

A Linhagem Java

O Java está relacionado ao C++, que é descendente direto do C. Grande parte dos caracteres Java é herdada destas duas linguagens. Do C, o Java tirou sua sintaxe. Muitas das características orientadas a objeto do Java foram influenciadas pelo C++. Na verdade, muitas das características que definem o Java vieram – ou são respostas – de seus predecessores. Além disso, a criação do Java foi profundamente enraizada no processo de refinamento e adaptação que vem ocorrendo nas linguagens nas últimas décadas. Por estes motivos, esta seção irá rever a sequência de acontecimentos e forças que levou à criação do Java. Como você verá, cada inovação no design da linguagem foi orientada pela necessidade de solucionar um problema fundamental que as linguagens anteriores não podiam resolver. O Java não é exceção.

O Nascimento da Programação Moderna: C

A linguagem C sacudiu o mundo da informática. Seu impacto não deveria ser subestimado, pois esta linguagem mudou a forma como a programação era abordada e pensada. A criação do C foi o resultado direto da necessidade de uma linguagem estruturada, eficiente e de alto nível que pudesse substituir o código de montagem na criação de programas para sistemas. Como você provavelmente sabe, quando uma linguagem é projetada, geralmente ocorrem trocas como as seguintes:

- Facilidade de uso *versus* potência.
- Segurança *versus* eficiência.
- Rigidez *versus* extensibilidade.

Antes do C, os programadores geralmente tinham de escolher entre linguagens que otimizavam um ou outro conjunto de características. Por exemplo: embora o FORTRAN pudesse ser usado para escrever programas razoavelmente eficientes para aplicativos científicos, ele não era muito bom para escrever código de sistema. E, enquanto o BASIC era fácil de aprender, ele não era muito potente, e sua falta de estrutura tornou sua utilidade questionável para programas maiores. A linguagem *Assembly* pode ser usada para gerar programas altamente eficientes, mas não é tão fácil de ser aprendida ou não pode ser usada de maneira eficiente. Além disso, depurar um código *assembly* pode ser bem difícil.

Outro problema era que as primeiras linguagens, como BASIC, COBOL e FORTRAN, não eram projetadas a partir de princípios estruturados. Ao invés disso, elas dependiam do GOTO como seu principal meio de controle do programa. Como resultado, os programas escritos com estas linguagens tendiam a produzir um código conhecido como "macarronada" – um emaranhado de saltos e ramificações condicionais que tornam um programa virtualmente impossível de se compreender. Ao mesmo tempo em que linguagens como Pascal eram estruturadas, elas não eram projetadas para serem eficientes e não eram capazes de incluir determinadas características necessárias para que pudessem ser aplicadas em uma grande variedade de programas. (Especificamente, com os dialetos padrão para Pascal disponíveis na época, não era prático considerar o uso do Pascal para códigos em nível de sistema.)

Portanto, pouco antes da invenção do C, nenhuma outra linguagem havia reconciliado os atributos conflitantes que levaram ao fracasso. Ao mesmo tempo, a necessidade por uma linguagem que solucionasse estes conflitos era urgente. No início dos anos 1970, a revolução na informática já estava começando a se estabelecer, e a demanda por software estava cada vez mais superando a capacidade dos programadores de produzí-lo. Foi feito um grande esforço nos círculos acadêmicos em uma tentativa de criar uma linguagem melhor. Mas, e talvez o mais importante, uma força secundária começava a ser sentida. O hardware finalmente se tornava comum o suficiente e um número importante de pessoas estava sendo atingido. Os computadores não eram mais mantidos atrás de portas trancadas. Pela primeira vez, os programadores obtinham acesso praticamente ilimitado às suas máquinas. Isso lhes dava liberdade para experimentar e também permitia a eles começarem a criar suas próprias ferramentas. Às vésperas da criação do C, o palco estava pronto para um grande salto no que dizia respeito às linguagens.

Inventado e implementado pela primeira vez por Dennis Ritchie em um DEC PDP-11 com sistema operacional UNIX, o C foi o resultado de um processo de desenvolvimento que começou com uma linguagem mais antiga, chamada BCPL, desenvolvida por Martin Richards. O BCPL influenciou uma linguagem, chamada B, inventada por Ken Thompson, que levou ao desenvolvimento do C nos anos 1970.

Durante muitos anos, o padrão de fato para C era fornecido pelo sistema operacional UNIX e descrito em 1986, no livro *C: A Linguagem de Programação*, de Brian Kernighan e Dennis Ritchie. A linguagem C foi oficialmente padronizada em dezembro de 1989, quando o padrão ANSI (American National Standards Institute) para C foi adotado.

A criação do C é considerada por muitos como o marco do início da idade moderna das linguagens. Ela sintetizava, de maneira bem-sucedida, os atributos conflitantes que tanto atormentavam as outras linguagens. O resultado foi uma linguagem potente, eficiente e estruturada, e relativamente fácil de usar. Ela também tinha um aspecto quase que intangível: era uma linguagem de *programador*. Antes da invenção do C, as linguagens eram projetadas de maneira geral como exercícios acadêmicos ou por comitês burocráticos. O C era diferente. A linguagem havia sido projetada, implementada e desenvolvida por programadores de verdade, refletindo o modo como eles abordavam a tarefa de desenvolver. Seus recursos foram ajustados, testados, pensados e repensados pelas pessoas que realmente usavam a linguagem. O resultado foi uma linguagem que os programadores gostavam de usar. De fato, o C rapidamente atraiu muitos seguidores que tinham um zelo quase que religioso com a linguagem. Sendo assim, o C encontrou uma aceitação ampla e rápida na comunidade de programadores. Resumindo, o C é uma linguagem criada por e para programadores. Como você vai ver, o Java herdou uma parte desta linguagem.

C++: O Passo Seguinte

No final dos anos 1970 e início dos anos 1980, o C era a linguagem dominante e ainda é bastante utilizada atualmente. Como o C é uma linguagem bem-sucedida e útil, você pode estar se perguntando por que havia a necessidade de algo mais. A resposta é: *complexidade*. Na história da programação, a crescente complexidade dos programas orientou a necessidade por melhores maneiras de gerenciar tal complexidade. O C++ é uma resposta a esta necessidade. Para entender melhor por que gerenciar a complexidade de um programa foi fundamental para a criação do C++, considere o seguinte:

As abordagens com relação à programação mudaram drasticamente desde a invenção do computador. Por exemplo: quando os computadores foram inventados, a programação era feita manualmente através de instruções de uma máquina binária no painel frontal. Desde que os programas tivessem algumas centenas de instruções, esta abordagem funcionava. Com o crescimento dos programas, a linguagem *Assembly* foi inventada para que o desenvolvedor pudesse lidar com programas cada vez maiores e mais complexos, utilizando representações simbólicas das instruções da máquina. Como os programas continuaram crescendo, as linguagens de alto nível foram introduzidas e deram ao desenvolvedor mais ferramentas para lidar com a complexidade.

A primeira linguagem mais difundida, é claro, foi o FORTRAN. Ao mesmo tempo em que o FORTRAN foi um primeiro passo impressionante, não se trata de uma linguagem que incentive programas claros e fáceis de entender. Nos anos 1960 nasceu a *programação estruturada*. Este é o método de programação preferido das linguagens como C. O uso de linguagens estruturadas permitiu aos programadores escrever, de modo relativamente fácil, pela primeira vez, programas moderadamente complexos. No entanto, mesmo com os métodos de programação estruturada, uma vez que um projeto atinja um determinado tamanho, sua complexidade excede o que pode ser gerenciado pelo programador. No início dos anos 1980, muitos projetos forçavam os limites da abordagem estruturada. Para resolver este problema, uma nova maneira de programar foi inventada e chamada de *programação orientada a objetos* (em inglês, *object-oriented programming – OOP*). A

programação orientada a objetos será discutida em detalhes mais adiante neste livro, mas daremos uma breve definição: a programação orientada a objetos (POO) é uma metodologia de programação que ajuda a organizar programas complexos através do uso de herança, encapsulamento e polimorfismo.

Na análise final, embora o C seja uma das maiores linguagens do mundo, existe um limite em sua capacidade de lidar com a complexidade. Quando o tamanho de um programa excede um determinado ponto, ele fica tão complexo que é difícil vê-lo na totalidade. Enquanto o tamanho preciso em que isso ocorre difere dependendo da natureza do programa e do programador, sempre há um ponto em que um programa passa a deixar de ser gerenciável. O C++ agregou recursos que permitiam que este ponto fosse eliminado, possibilitando aos programadores compreenderem e gerenciarem programas maiores.

O C++ foi inventado por Bjarne Stroustrup, em 1979, quando ele trabalhava na Bell Laboratories, em Murray Hill, New Jersey. Stroustrup inicialmente chamou a linguagem de "C com Classes". No entanto, em 1983, o nome foi mudado para C++. O C++ estende o C, adicionando recursos orientados a objetos. Como o C++ é construído com base em C, ele tem todos os recursos, atributos e benefícios da linguagem C. Este é um motivo crucial para o sucesso do C++ como linguagem. A invenção do C++ não foi uma tentativa de criar uma linguagem totalmente nova; ela foi uma melhoria de uma linguagem que já era muito bem-sucedida.

O Palco Está Pronto para o Java

No final dos anos 1980 e início dos anos 1990, a programação orientada a objetos usando o C++ era dominante. Na verdade, por um breve momento, parecia que os programadores haviam finalmente encontrado a linguagem perfeita. Como o C++ misturava os elementos altamente eficientes e o estilo do C com o paradigma da orientação a objetos, era uma linguagem que podia ser usada para criar uma grande variedade de programas. No entanto, assim como havia ocorrido no passado, havia forças trabalhando para, mais uma vez, levar a evolução da linguagem um passo à frente. Em alguns anos, a World Wide Web e a internet atingiriam um público gigantesco. Este evento desencadearia outra revolução na programação.

A Criação do Java

O Java foi concebido por James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan, na Sun Microsystems, Inc., em 1991. Levaram-se 18 meses para desenvolver a primeira versão que funcionasse. Inicialmente, a linguagem se chamava "Oak", mas recebeu o nome "Java", em 1995. Entre a implementação inicial do Oak, na primavera de 1992, e o anúncio público do Java, no outono de 1995, muitas outras pessoas contribuíram para o projeto e evolução da linguagem. , Arthur van Hoff, Jonathan Payne, Frank Yellin e Tim Lindholm foram pessoas muito importantes no amadurecimento do protótipo original.

Surpreendentemente, o objetivo original do Java não era a internet! Ao invés disso, a principal motivação era a necessidade de uma linguagem independente de plataforma (isto é, com arquitetura neutra), que pudesse ser usada para criar software que pudesse ser inserido em vários dispositivos eletrônicos de consumo, como fornos de micro-ondas e controles remotos. Como você deve estar supondo, muitos tipos diferentes de CPUs são usados como controladores. O problema com C e C++ (e a maioria das outras linguagens) é que elas são projetadas para serem compiladas com um objetivo específico. Embora seja possível compilar um programa em C++ para praticamente qualquer tipo de CPU, fazer isso exige um

compilador C++ totalmente dedicado àquela CPU. O problema é que os compiladores são caros e consomem tempo para serem criados. Uma solução mais fácil – e mais eficiente em termos de custo – era necessária. Em uma tentativa de encontrar esta solução, Gosling e outros começaram a trabalhar em uma linguagem portátil, independente de plataforma e que tivesse possibilidade de ser usada para gerar código que pudesse ser executado em diversas CPUs e em diferentes ambientes. Estes esforços acabaram levando à criação do Java.

Quando os detalhes do Java estavam sendo trabalhados, um segundo fator, que acabou sendo mais importante, surgia e teria um papel crucial no futuro do Java. Esta segunda força era, é claro, a World Wide Web. Se a web não tivesse sido criada quase que ao mesmo tempo em que o Java estava sendo implementado, o Java seria uma linguagem útil, porém obscura, para programar aparelhos eletrônicos. No entanto, com o surgimento da World Wide Web, o Java foi colocado em primeiro lugar no projeto de linguagens, pois a web também demandava programas portáteis.

A maioria dos programadores aprende, no início de suas carreiras, que os programas portáteis são tão ilusórios quanto desejáveis. Ao mesmo tempo em que a busca por uma forma de criar programas eficientes e portáteis (independentes de plataforma) é quase tão antiga quanto a disciplina de programação em si, ela acabou pegando carona em outros problemas mais urgentes. Além disso, como (na época) grande parte do mundo da informática havia se dividido em três acampamentos de competição entre Intel, Macintosh e UNIX, a maioria dos programadores permanecia em seus limites protegidos, e a necessidade urgente por um código portátil foi reduzida. No entanto, com o advento da web e da internet, o antigo problema da portabilidade voltou com uma vingança. Afinal de contas, a internet é composta por um universo, diverso e distribuído, ocupado por vários tipos de computadores, sistemas operacionais e CPUs. Embora muitos tipos de plataformas estejam ligados à internet, os usuários gostariam que todas eles fossem capazes de executar o mesmo programa. O que uma vez foi um problema irritante, mas de baixa prioridade, passara a ser uma grande necessidade.

Por volta de 1993, tornou-se óbvio para os membros da equipe do projeto Java que os problemas de portabilidade, frequentemente encontrados durante a criação de código para controladores integrados, também são encontrados quando se tenta criar código para a internet. Na verdade, o mesmo problema para o qual o Java foi inicialmente criado para solucionar em pequena escala também podia ser aplicado à internet, só que em larga escala. Esta percepção fez com que o foco do Java passasse dos eletrônicos de consumo à programação para internet. Portanto, enquanto o desejo por uma linguagem de arquitetura neutra foi a faísca inicial, a internet acabou conduzindo o Java ao sucesso em larga escala.

Como mencionado anteriormente, o Java tem muitas características do C e do C++. Isso é intencional. Os criadores do Java sabiam que, ao usar a sintaxe familiar do C e ao reproduzir os recursos orientados a objetos do C++, sua linguagem chamaria atenção das legiões de programadores experientes de C e C++. Além das similaridades superficiais, o Java compartilha outros atributos que ajudaram no sucesso do C e do C++. Primeiro: o Java foi projetado, testado e refinado por programadores de verdade. É uma linguagem baseada nas necessidades e experiências das pessoas que o criaram. Portanto, o Java é uma linguagem para desenvolvedores. Segundo: o Java é coeso e logicamente consistente. Terceiro: com exceção das restrições impostas pelo ambiente da internet, o Java proporciona, ao programador, controle total. Se você programar bem, seu programa refletirá isto. Se você programar mal, o programa também refletirá. Em outras palavras, o Java não é uma linguagem com “rodinhas” de treinamento; é uma linguagem para programadores profissionais.

Devido às similaridades entre Java e C++, é tentador pensar no Java como a "versão para internet do C++". No entanto, este é um grande erro. O Java tem diferenças práticas e filosóficas significativas. É verdade que a linguagem foi influenciada pelo C++, mas não é uma versão melhorada do C++. Por exemplo: o Java não pode ser comparado superior ou inferiormente com o C++. É claro que as similaridades com o C++ são importantes e, se você é programador de C++, se sentirá em casa com o Java. Outro ponto: o Java não foi criado para substituir o C++, e sim para solucionar um determinado conjunto de problemas. O C++ foi desenvolvido para solucionar um conjunto diferente de problemas. Ambos irão coexistir durante os muitos anos que virão.

Como foi dito no início deste capítulo, as linguagens evoluem por dois motivos: para se adaptar às mudanças no ambiente e para implementar avanços na arte da programação. A mudança ambiental que culminou na criação do Java foi a necessidade de programas independentes de plataforma destinados à distribuição na internet. No entanto, o Java também incorpora mudanças no modo como as pessoas abordam a escrita dos programas. Por exemplo: o Java melhorou e refinou o paradigma da programação orientada a objetos usado pelo C++, acrescentou suporte integrado à programação *multithread* (em várias linhas de execução) e forneceu uma biblioteca que simplifica o acesso à internet. Na análise final, no entanto, não foram as características individuais do Java que o tornaram tão marcante; foi a linguagem como um todo. O Java foi a resposta perfeita para as demandas do então novo e altamente distribuído universo da informática. O Java foi para a programação da internet o que o C foi para a programação de sistemas: uma força revolucionária que mudou o mundo.

A Conexão com o C#

O alcance e o poder do Java continuam sendo sentidos no mundo do desenvolvimento das linguagens. Muitos de seus recursos inovadores, estruturas e conceitos passaram a fazer parte da base de qualquer nova linguagem. O sucesso do Java é simplesmente importante demais para ser ignorado.

Talvez o exemplo mais importante da influência do Java seja o C#. Criado pela Microsoft para suportar o Framework .NET, o C# está intimamente relacionado ao Java. Por exemplo: ambos compartilham a mesma sintaxe geral, suportam programação distribuída e utilizam o mesmo modelo de objeto. É claro que existem diferenças entre o Java e o C#, mas a "aparência" geral destas linguagens é bastante similar. Esta "interpolinização" do Java para o C# é o testemunho mais forte de que o Java redefiniu o modo como pensamos e usamos uma linguagem.

Como o Java Mudou a Internet

A internet ajudou a impulsionar o Java à frente da programação, e a linguagem, por sua vez, teve um efeito profundo na internet. Além de simplificar a programação web como um todo, o Java inovou num novo tipo de programa em cadeia, chamado *applet*, que mudou a maneira como o mundo online pensava sobre conteúdo. O Java também tratou das questões mais espinhosas associadas à internet: portabilidade e segurança. Vamos ver estes dois tópicos com mais detalhes.

Applets Java

Um *applet* é um tipo especial de programa Java criado para ser transmitido pela internet e executado automaticamente por um navegador web compatível com o Java.

Além disso, um applet é baixado sob demanda, sem qualquer interação com o usuário. Se o usuário clica em um link que contém um applet, o mesmo é automaticamente baixado e executado no navegador. Os applets devem ser programas pequenos. Tipicamente, são usados para exibir dados fornecidos pelo servidor, manipular os dados inseridos pelo usuário, ou fornecer funções simples, como uma calculadora de empréstimo, executada localmente e não no servidor. Basicamente, o applet permite alguma funcionalidade para ser movido do servidor para o cliente.

A criação do applet mudou a programação para internet, porque expandiu o universo de objetos que podem se mover livremente no ciberespaço. Em geral, existem duas categorias bastante amplas de objetos que são transmitidos entre o servidor e o cliente: informações passivas e programas dinâmicos, ativos. Por exemplo: quando você lê o seu e-mail, está visualizando dados passivos. Mesmo quando baixa um programa, o código do programa ainda é somente dados passivos até que você o execute. Em contrapartida, o applet é um programa dinâmico, autoexecutável. Um programa assim é um agente ativo no cliente, ainda que seja iniciado pelo servidor.

Mesmo sendo dinâmicos, os programas de comunicação também apresentam sérios problemas nas áreas de segurança e portabilidade. Obviamente, um programa que é baixado e executado automaticamente no cliente deve ser impedido de causar danos. Ele também deve ser capaz de ser executado em uma variedade de ambientes e em sistemas operacionais diferentes. Como você vai ver, o Java solucionou estes problemas de maneira eficiente e elegante. Vamos ver com mais detalhes.

Segurança

Como você deve estar ciente, sempre que baixa um programa "normal", você está se arriscando, pois o código que está baixando pode conter um vírus, um Cavalo de Troia ou qualquer outro código danoso. No centro do problema está o fato de que o código malicioso pode causar danos, porque obteve acesso não autorizado aos recursos do sistema. Por exemplo: um programa de vírus pode reunir informações particulares, como números de cartões de crédito, extratos bancários e senhas, pesquisando o conteúdo do sistema de arquivos locais de seu computador. Para que o Java permita que os applets sejam baixados e executados no computador cliente com segurança, foi necessário evitar que um applet lançasse um ataque.

O Java conseguiu esta proteção, confinando o applet no ambiente de execução do Java e não lhe permitindo acesso a outras partes do computador. (Você verá como se faz isso em instantes.) A capacidade de baixar applets, confiando que nenhum dano será causado e que nenhuma segurança será quebrada, é considerada por muitos como o único aspecto mais inovador do Java.

Portabilidade

A portabilidade é um dos principais aspectos da internet, pois existem muitos tipos diferentes de computadores e sistemas operacionais conectados a ela. Se um programa Java fosse executado em praticamente qualquer computador conectado à internet, precisaria haver alguma maneira de permitir que este programa fosse executado em diferentes sistemas. Por exemplo: no caso de um applet, o mesmo applet deve ser capaz de ser baixado e executado pela grande variedade de CPUs, sistemas operacionais e navegadores conectada à internet. Não é prático ter várias versões do applet para diferentes computadores. O *mesmo* código deve funcionar em *todos* os computadores. Portanto, algum meio de gerar código executável portátil era necessário. Como você verá em breve, o mesmo mecanismo que ajuda a garantir a segurança também ajuda a criar a portabilidade.

A Mágica do Java: O Bytecode

O segredo que permite ao Java solucionar o problema da segurança e da portabilidade descritos acima é que os dados de saída do compilador Java não são um código executável, e sim *bytecode*. O bytecode é um conjunto de instruções, altamente otimizado, criado para ser executado pelo sistema de tempo de execução do Java, chamado de *Java Virtual Machine (JVM), ou máquina virtual Java*. Basicamente, a JVM original foi criado como um *intérprete de bytecode*. Isso pode ser um pouco surpreendente já que muitas linguagens modernas são criadas para serem compiladas em código executável devido à preocupação com o desempenho. No entanto, o fato de um programa Java ser executado pela JVM ajuda a solucionar os principais problemas associados aos programas baseados na web. Eis o motivo.

Traduzir um programa Java para bytecode facilita bastante a execução do programa em uma grande variedade de ambientes, pois somente a JVM precisa ser implementado para cada plataforma. Uma vez que o pacote de tempo de execução exista para um determinado sistema, qualquer programa Java poderá ser executado nele. Lembre-se: embora os detalhes da JVM sejam diferentes de uma plataforma para outra, todos compreendem o mesmo bytecode Java. Se um programa Java fosse compilado em código nativo, precisariam existir diferentes versões do mesmo programa para cada tipo de CPU conectada à internet. Esta, é claro, não é uma solução muito inteligente. Portanto, a execução do bytecode pela JVM é a maneira mais fácil de criar programas realmente portáteis.

O fato de que um programa Java é executado pela JVM também ajuda a torná-lo seguro. Como a JVM tem o controle, ele pode conter o programa e evitar que se criem efeitos colaterais fora do sistema. Como se verá, a segurança também é aprimorada por determinadas restrições existentes na linguagem Java.

Em geral, quando um programa é compilado para uma forma intermediária e, em seguida, interpretado por uma máquina virtual, ele é mais lento do que seria se fosse compilado em código executável. No entanto, com o Java, a diferença entre os dois não é muito grande. Como o bytecode foi altamente otimizado, seu uso permite que a JVM execute programas muito mais rápido do que se espera.

Embora o Java tenha sido criado como uma linguagem interpretada, não há nada nele que evite uma compilação de bytecode para código nativo de última hora para aumentar o desempenho. Por este motivo, a tecnologia HotSpot foi introduzida não muito tempo depois do lançamento do Java. O HotSpot fornece um compilador de bytecode *Just-In-Time* (JIT). Quando um compilador JIT faz parte da JVM, partes selecionadas de bytecode são compiladas em código executável em tempo real e sob demanda, parte por parte. É importante compreender que não é prático compilar todo um programa Java em código executável de uma vez, pois o Java executa diversas verificações em tempo de execução. O compilador JIT compila o código, conforme necessário, durante a execução. Além disso, nem todas as sequências de bytecode são compiladas – somente aquelas que terão benefícios com a compilação. O restante do código é simplesmente interpretado. No entanto, a abordagem *just-in-time* ainda gera uma melhora significativa no desempenho. Mesmo quando a compilação dinâmica é aplicada ao bytecode, os recursos de portabilidade e segurança ainda se aplicam, pois a JVM ainda está encarregada do ambiente de execução.

Servlets: Java no Lado Servidor

Mesmo tendo toda essa utilidade, os applets são apenas metade da equação cliente/servidor. Não muito depois do lançamento do Java, ficou óbvio que a linguagem

também seria útil no lado servidor. O resultado foi o *servlet*. Um servlet é um programa pequeno executado no servidor. Assim como os applets estendem dinamicamente a funcionalidade de um navegador web, os servlets estendem dinamicamente a funcionalidade de um servidor web. Portanto, com a chegada do servlet, o Java abraçou os dois lados da conexão cliente/servidor.

Os servlets são usados para criar conteúdo gerado dinamicamente que, em seguida, é servido para o cliente. Por exemplo: uma loja online poderia usar um servlet para procurar o preço de um item em um banco de dados. Em seguida, a informação do preço é usada para gerar dinamicamente uma página web que é enviada para o navegador. Embora o conteúdo gerado dinamicamente possa ser disponibilizado através de mecanismos como CGI (Common Gateway Interface), o servlet oferece diversas vantagens, incluindo melhoria no desempenho.

Como os servlets (assim como todos os programas Java) são compilados em bytecode e executados pela JVM, eles são altamente portáteis. Portanto, o mesmo servlet pode ser usado em uma variedade de ambientes servidores. Os únicos requisitos são que o servidor suporte a JVM e um recipiente de servlet (servlet container).

Os Termos Java

Nenhuma discussão sobre a história do Java é completa sem uma olhada nos termos desta linguagem. Embora as forças fundamentais que motivaram a invenção do Java sejam portabilidade e segurança, outros fatores também tiveram um papel importante no molde final da linguagem. As principais considerações foram resumidas pela equipe Java na seguinte lista de termos:

- Simples;
- Seguro;
- Portátil;
- Orientado a objetos;
- Robusto;
- Multithreaded;
- Arquitetura neutra;
- Interpretado;
- Alta performance;
- Distribuído;
- Dinâmico.

Dois destes termos já foram discutidos: seguro e portátil. Vamos analisar o que cada um dos outros quer dizer.

Simples

O Java foi criado para facilitar o aprendizado do programador profissional e para ser usado de maneira eficaz. Supondo que você já tenha experiência em programação, você não achará difícil trabalhar com o Java. Se já compreende os conceitos básicos da programação orientada a objetos, aprender Java será ainda mais fácil. O melhor de tudo: se você é um programador experiente de C++, passar para o Java exigirá muito pouco esforço. Como o Java herdou a sintaxe do C/C++ e muitas características da programação orientada a objetos do C++, a maioria dos desenvolvedores não tem problemas em aprender Java.

Orientado a Objetos

Embora seja influenciado por seus antecessores, o Java não foi criado para ser um código fonte compatível com qualquer outra linguagem. Isso deu à equipe Java liberdade para criar a partir do zero. Um dos resultados foi uma abordagem limpa, útil e pragmática com relação aos objetos. Fazendo empréstimos de muitos ambientes orientados a software das últimas décadas, o Java consegue atingir um equilíbrio entre o paradigma purista de que "tudo é um objeto" e o modelo pragmático "saia do meu caminho". O modelo do objeto em Java é simples e fácil de estender, enquanto que os tipos primitivos, como inteiros, são mantidos como não objetos de alta performance.

Robusto

O ambiente multiplataforma da web faz exigências extraordinárias a um programa, pois deve ser executado de maneira confiável em vários sistemas. Portanto, a capacidade de criar programas robustos recebeu alta prioridade na criação do Java. Para ganhar confiança, o Java restringe o usuário em algumas áreas-chave para forçá-lo a encontrar seus erros no início do desenvolvimento do programa. Ao mesmo tempo, o Java o libera de se preocupar com muitas das causas mais comuns de erros de programação. Como o Java é uma linguagem estritamente tipada, ele verifica seu código no momento da compilação. No entanto, ele também verifica seu código no tempo de execução. Muitos bugs difíceis de rastrear, que geralmente aparecem em situações de tempo de execução difíceis de serem reproduzidas, são simplesmente impossíveis de serem criados em Java. Saber que aquilo que você escreveu irá, sob diversas condições, se comportar de maneira previsível é um dos pontos chave do Java.

Para entender melhor como o Java é robusto, considere dois dos principais motivos para o programa falhar: erros no gerenciamento de memória e condições excepcionais mal manipuladas (isto é, erros de tempo de execução). O gerenciamento da memória pode ser uma tarefa difícil e entediante nos ambientes tradicionais de programação. Por exemplo: em C/C++, o programador deve alocar e liberar manualmente toda memória dinâmica. Isto, às vezes, gera problemas, pois os programadores se esquecem de liberar a memória que já havia sido alocada ou, pior ainda, tentam liberar uma memória que outra parte do código ainda está usando. O Java praticamente elimina estes problemas, gerenciando a alocação e desalocação de memória para você. (Na verdade, a desalocação é totalmente automática, pois o Java fornece coleta de lixo para objetos não utilizados, conhecido como Garbage Collection.) As condições excepcionais nos ambientes tradicionais geralmente surgem em situações como a divisão por zero ou "arquivo não encontrado" e devem ser gerenciadas com estruturas desajeitadas e difíceis de ler. O Java ajuda nesta parte, fornecendo manipulação de exceções orientada a objetos. Em um programa Java bem escrito, todos os erros de tempo de execução podem – e devem – ser gerenciados pelo seu programa.

Multithreaded

O Java foi criado para atender ao requisito do mundo real de criar programas interativos e conectados. Para fazer isso, a linguagem suporta programação multithreaded, que permite a você escrever programas que façam muitas coisas ao mesmo tempo. O sistema de tempo de execução do Java tem uma solução elegante, porém sofisticada, para a sincronização de diversos processos, o que permite ao desenvolvedor construir sistemas interativos que rodam perfeitamente. A abordagem fácil de usar do Java para a programação multithreaded permite que você pense no comportamento específico de seu programa, e não no subsistema de tarefas combinadas.